King's Research Portal

*Document Version*
Peer reviewed version

Link to publication record in King's Research Portal

*Citation for published version (APA):*
Kosiol, J., Strüber, D., Taentzer, G., & Zschaler, S. (in press). Finding the Right Way to Rome: Effect-oriented Graph Transformation. In *International Conference on Graph Transformations*

**Citing this paper**
Please note that where the full-text provided on King's Research Portal is the Author Accepted Manuscript or Post-Print version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version for pagination, volume/issue, and date of publication details. And where the final published version is provided on the Research Portal, if citing you are again advised to check the publisher's website for any subsequent corrections.

**General rights**
Copyright and moral rights for the publications made accessible in the Research Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

•Users may download and print one copy of any publication from the Research Portal for the purpose of private study or research.
•You may not further distribute the material or use it for any profit-making activity or commercial gain
•You may freely distribute the URL identifying the publication in the Research Portal

**Take down policy**
If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 12. Jan. 2025

# Finding the Right Way to Rome: Effect-oriented Graph Transformation

Jens Kosiol[1], Daniel Strüber[2,3], Gabriele Taentzer[1], and Steffen Zschaler[4]

[1] Philipps-Universität Marburg, Marburg, Germany
{taentzer,kosiolje}@mathematik.uni-marburg.de
[2] Chalmers | University of Gothenburg, Gothenburg, Sweden
danstru@chalmers.se
[3] Radboud University, Nijmegen, Netherlands
[4] King's College London, London, UK
szschaler@acm.org

**Abstract.** Many applications of graph transformation require rules that change a graph without introducing new consistency violations. When designing such rules, it is natural to think about the desired outcome state, i.e., the desired *effect*, rather than the specific steps required to achieve it; these steps may vary depending on the specific rule-application context. Existing graph-transformation approaches either require a separate rule to be written for every possible application context or lack the ability to constrain the maximal change that a rule will create. We introduce *effect-oriented graph transformation*, shifting the semantics of a rule from specifying actions to representing the desired effect. A single effect-oriented rule can encode a large number of *induced* classic rules. Which of the *potential* actions is executed depends on the application context; ultimately, all ways lead to Rome. If a graph element to be deleted (created) by a potential action is already absent (present), this action need not be performed because the desired outcome is already present. We formally define effect-oriented graph transformation, show how matches can be computed without explicitly enumerating all induced classic rules, and report on a prototypical implementation of effect-oriented graph transformation in Henshin.

**Keywords:** Graph transformation · Double-pushout approach · Consistency-preserving transformations

## 1 Introduction

Applications of graph transformation such as model synchronisation [14,13,20] or search-based optimisation [5,19] require graph-transformation rules that combine a change to the graph with repair [28] operations to ensure transformations are consistency sustaining or even improving [21]. For any given graph constraint, there are typically many different ways in which it can be violated, requiring slightly different specific changes to repair the violation. As a result, it is often

easier to think about the desired *effect* of a repairing graph transformation rule rather than the specific transformations required. We would like to be able to reach a certain state of the graph—defined in terms of the presence or absence of particular graph elements (the *effect*)—even if, in different situations, a different set of specific changes is required to achieve this.

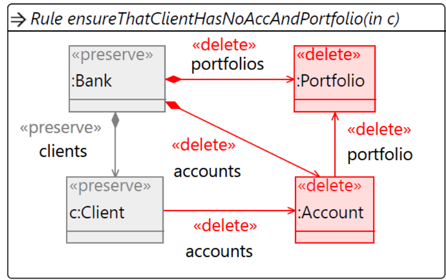
Existing approaches to graph transformation make it difficult to precisely capture the effect of a rule without explicitly specifying the specific set of changes required. For example, the *double-pushout approach (DPO)* to graph transformation [10,11] has gained acceptance as an underlying formal semantics for *graph* and *model transformation rules* in practice as a simple and intuitive approach: A transformation rule simply specifies which graph elements are to be deleted and created when it is applied; that is, a rule prescribes exactly all the *actions* to be performed. For graph repair, this effectively forces one to specify every way in which a constraint can be violated and the specific changes to apply in this case, so that the right rule can be applied depending on context. On the other end of the spectrum, the *double-pullback approach* [18] is much more flexible. Here, rules only specify minimal changes. However, there is no way of operationally constraining the maximal possible change.

There currently exists no approach to graph transformation that allows the *effect* of rules to be specified concisely and precisely without specifying every action that needs to be taken. In this paper, we introduce the notion of *effect-oriented graph transformations.* In this approach, graph-transformation rules encode a, potentially large, number of *induced rules.* This is achieved by differentiating basic actions that have to be performed by any transformation consistent with the rule and *potential* actions that only have to be performed if they are required to achieve the intended rule *effect.* Depending on the application context, a different set of actions will be executed—all ways lead to Rome. We provide an algorithm for selecting the right set of actions depending on context, without having to explicitly enumerate all possibilities—we efficiently find the right way to Rome.

Thus, the paper makes the following contributions:

1. We define the new notion of effect-oriented graph transformation rules and discuss different notions of consistent matches for these;
2. We provide an algorithm for constructing a complete match and a transformation given a partial match for an effect-oriented transformation rule. The algorithm is efficient in the sense that it avoids computing and matching all induced rules explicitly;
3. We report on a prototypical implementation of effect-oriented transformations in Henshin; and
4. We compare our approach to existing approaches to graph transformation showing that it does indeed provide new expressivity.

The rest of this paper is structured as follows. First, we introduce a running example (Sect. 2) and briefly recall basic preliminaries (Sect. 3). Section 4 introduces effect-oriented rules and transformations and several notions of constructing matches. Section 5 explains in more detail one algorithm for constructing matches



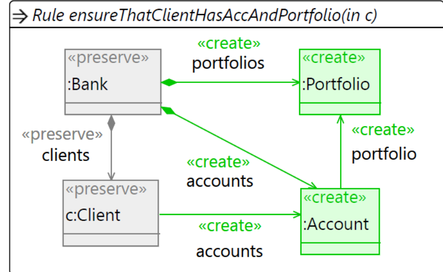




**Fig. 1.** Example rules, shown in the integrated visual syntax of Henshin [1,29]. The LHS of a rule consists of all red and grey elements (additionally annotated with delete or preserve), the RHS of all green (additionally annotated as create) and grey elements.

for effect-oriented rules and reports on a prototype implementation in Henshin. In Section 6, we discuss how existing applications can benefit from effect-oriented rules and transformations, and compare our new approach to graph transformation with other approaches that could be used to achieve these goals. We conclude in Sect. 7. We provide some additional results and explanations and proofs of our formal statements in an extended version of this paper [22].

## 2 Running Example

We use the well-known banking example [23] and adapt it slightly to illustrate our newly introduced concept of effect-oriented transformations. Assume that the context of this example is specified in a meta-model formalised as a type graph (not shown) for the banking domain in which a Bank has Clients, Accounts and Portfolios. A Client may have Accounts which may be associated with a Portfolio.

Imagine a scenario where it is to be ensured that a Client has an Account with a Portfolio. To realise this condition in a rule-based manner so that no unnecessary elements are created, at least three rules (and a programme to coordinate their application) are required: A rule that checks whether a Client already has an Account with a Portfolio, a rule that adds a Portfolio to an existing Account of the Client, and a rule that creates all the required structure; this last rule is shown as the rule ensureThatClientHasAccAndPortfolio in Fig. 1.

An analogous problem exists if the Accounts and Portfolios of a Client are to be removed. The rule ensureThatClientHasNoAccAndPortfolio in Fig. 1 is the rule that deletes the entire structure, and additional rules are needed to delete Accounts that are not associated with Portfolios. In general, the number of rules needed and the complexity of their coordination depend on the size of the structure to occur together and hence, to be created (deleted).

With our new notion of *effect-oriented rules* and *transformations*, we provide the possibility to use a single rule to specify *all* desired behaviours by making the rule's semantics dependent on the context in which it is applied. Specifically,

if the rule ensureThatClientHasAccAndPortfolio is applied to a Client in effect-oriented semantics, this allows for matching an existing Account and/or Portfolio (rather than creating them) and only creating the remainder. Therefore, we call the creation actions for Account and Portfolio *potential actions* that are only executed if the corresponding elements do not yet exist. Similarly, applying the rule ensureThatClientHasNoAccAndPortfolio in effect-oriented semantics allows deleting nothing (if the matched Client has no Account and no Portfolio) or only an Account (that does not have a Portfolio). So here the deletion actions are potential actions that are only executed if the corresponding elements are present. We will allow for some of the actions of an effect-oriented rule to be mandatory. Note that in a Henshin rule, we would need to provide additional annotation to differentiate mandatory from potential actions. We will define different strategies for this kind of matching that may be appropriate for different application scenarios.

## 3 Preliminaries

In this section, we briefly recall basic preliminaries. Throughout our paper, we work with *typed graphs* and leave the treatment of attribution and type inheritance to future work. For brevity, we omit the definitions of *nested graph conditions* and their *shift* along morphisms [16,12]. We also omit basic notions from category theory; in particular, we omit standard facts about adhesive categories [24,11] (of which typed graphs are an example). While these are needed in our proofs, the core ideas in this work can be understood without their knowledge.

**Definition 1 (Graph. Graph morphism).** *A* graph $G = (V_G, E_G, src_G, tar_G)$ *consists of a set of* nodes *(or* vertices*)* $V_G$*, a set of* edges $E_G$*, and* source *and* target functions $src_G, tar_G \colon E_G \to V_G$ *that assign a* source *and a* target node *to each edge.*

*A* graph morphism $f = (f_V, f_E)$ *from a graph* $G$ *to a graph* $H$ *is a pair of functions* $f_V \colon V_G \to V_H$ *and* $f_E \colon E_G \to E_H$ *that both commute with the source and target functions, i.e., such that* $src_H \circ f_E = f_V \circ src_G$ *and* $tar_H \circ f_E = f_V \circ tar_G$*. A graph morphism is* injective/surjective/bijective *if both* $f_V$ *and* $f_E$ *are. We denote injective morphisms via a hooked arrow, i.e., as* $f \colon G \hookrightarrow H$*.*

Typing helps equip graphs with meaning; a *type graph* provides the available types for elements and morphisms assign the elements of *typed graphs* to those.

**Definition 2 (Type graph. Typed graph).** *Given a fixed graph* $TG$ *(the* type graph*), a* typed graph $G = (G, type_G)$ *(over* $TG$*) consists of a graph* $G$ *and a morphism* $type_G \colon G \to TG$*. A* typed morphism $f \colon G \to H$ *between typed graphs* $G$ *and* $H$ *(typed over the same type graph* $TG$*) is a graph morphism that satisfies* $type_H \circ f = type_G$*.*

Throughout this paper, we assume all graphs to be typed over a given type graph and all morphisms to be typed morphisms. However, for (notational) simplicity, we let this typing be implicit and just speak of graphs and morphisms. Moreover, all considered graphs are finite.

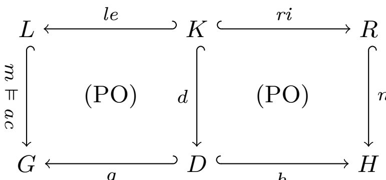



**Fig. 2.** A rule-based transformation in the Double-Pushout approach

**Definition 3 (Rules and transformations).** *A* rule $\rho = (p, ac)$ *consists of a* plain rule $p$ *and an* application condition $ac$*. The plain rule is a span of injective morphisms of typed graphs* $p = (L \overset{le}{\hookleftarrow} K \overset{ri}{\hookrightarrow} R)$*; its graphs are called* left-hand side *(LHS),* interface*, and* right-hand side *(RHS), respectively. The application condition $ac$ is a nested condition [16] over $L$.*

*Given a rule* $\rho = (L \overset{le}{\hookleftarrow} K \overset{ri}{\hookrightarrow} R, ac)$ *and an injective morphism* $m \colon L \hookrightarrow G$*, a* (direct) transformation $G \Longrightarrow_{\rho,m} H$ *from $G$ to $H$ (in the Double-Pushout approach) is given by the diagram in Fig. 2 where both squares are pushouts and $m$ satisfies the application condition $ac$, denoted as* $m \models ac$*. If such a transformation exists, the morphism $m$ is called a* match *and rule $\rho$ is* applicable *at match $m$; in this case, $n$ is called the* comatch *of the transformation. An injective morphism* $m \colon L \hookrightarrow G$ *with* $m \models ac$ *from the LHS of a rule to some graph $G$ is called a* pre-match*.*

For a rule to be applicable at a pre-match $m$, there must exist a pushout complement for $m \circ le$; in categories of graph-like structures, an elementary characterisation can be given in terms of the *dangling condition* [11, Fact 3.11]: A rule is applicable at a pre-match $m$ if and only if $m$ does not map a node to be deleted in $L$ to a node in $G$ with an incident edge that is not also to be deleted.

Application conditions can be 'shifted' along morphisms in a way that preserves their semantics [12, Lemma 3.11]. We presuppose this operation in our definition of *subrules* without repeating it. Our notion of a subrule is a simplification of the concept of *kernel* and *multi-rules* [15].

**Definition 4 (Subrule).** *Given a rule* $\rho = (L \overset{le}{\hookleftarrow} K \overset{ri}{\hookrightarrow} R, ac)$*, a* subrule *of $\rho$ is a rule* $\rho' = (L' \overset{le'}{\hookleftarrow} K' \overset{ri'}{\hookrightarrow} R', ac')$ *together with a subrule embedding* $\iota \colon \rho' \hookrightarrow \rho$ *where* $\iota = (\iota_L, \iota_K, \iota_R)$ *and* $\iota_X \colon X' \hookrightarrow X$ *is an injective morphism for* $X \in \{L, K, R\}$ *such that both squares in Fig. 3 are pullbacks and* $ac \equiv \mathrm{Shift}(\iota_L, ac')$*.*

## 4 Effect-oriented Rules and Transformations

The intuition behind effect-oriented semantics is that a rule prescribes the state that should prevail after its application, not the actions to be performed. In this section, we develop this approach. We introduce *effect-oriented rules* as a

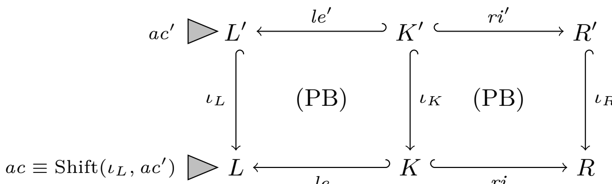



**Fig. 3.** Subrule $\rho'$ of a rule $\rho$

compact way to represent a whole set of *induced rules*. All induced rules share a common *base rule* as subrule (prescribing actions to be definitively performed) but implement different choices of the *potential actions* allowed by the effect-oriented rule. In a second step, we develop a semantics for effect-oriented rules; it depends on the larger context of effect-oriented transformations which of the induced rules is actually applied. Here, we implement the idea that potential deletions of an effect-oriented rule are to be performed if a suitable element exists but can otherwise be skipped. In contrast, a potential creation is only to be performed if there is not yet a suitable element. This maximises the number of deletions to be made while minimising the number of creations. We propose two ways in which this 'maximality' and 'minimality' can be formally defined.

### 4.1 Effect-oriented Rules as Representations of Rule Sets

In an *effect-oriented rule*, a *maximal rule* extends a *base rule* by *potential actions*. Here, and in all of the following, we assume that the left and right morphisms of rules and morphisms between rules (such as subrule embeddings) are actually inclusions. This does not lose generality (as the desired situation can always be achieved via renaming of elements) but significantly eases the presentation.

**Definition 5 (Effect-oriented rule).** *An* effect-oriented rule $\rho_\mathrm{e} = (\rho_\mathrm{b}, \rho_\mathrm{m}, \iota)$ *is a rule* $\rho_\mathrm{m} = (L_\mathrm{m} \xhookleftarrow{le_\mathrm{m}} K_\mathrm{m} \xhookrightarrow{ri_\mathrm{m}} R_\mathrm{m}, ac_\mathrm{m})$*, called* maximal rule*, together with a subrule* $\rho_\mathrm{b} = (L_\mathrm{b} \xhookleftarrow{le_\mathrm{b}} K_\mathrm{b} \xhookrightarrow{ri_\mathrm{b}} R_\mathrm{b}, ac_\mathrm{b})$*, called* base rule*, and a subrule embedding* $\iota \colon \rho_\mathrm{b} \hookrightarrow \rho_\mathrm{m}$ *such that* $K_\mathrm{b} = K_\mathrm{m}$ *(and* $\iota_K$ *is an identity).*

*The* potential deletions *of the maximal rule* $\rho_\mathrm{m}$ *are the elements of* $(L_\mathrm{m} \setminus K_\mathrm{m}) \setminus L_\mathrm{b} = L_\mathrm{m} \setminus L_\mathrm{b}$*; analogously, its* potential creations *are the elements of* $(R_\mathrm{m} \setminus K_\mathrm{m}) \setminus R_\mathrm{b} = R_\mathrm{m} \setminus R_\mathrm{b}$*. Here, and in the following, '$\setminus$' denotes the componentwise difference on the sets of nodes and edges.*

While requiring $K_\mathrm{b} = K_\mathrm{m}$ restricts the expressiveness of effect-oriented rules, it suffices for our purposes and allows for simpler definitions of their matching. If, during matching, potential actions would compete with potential interface elements for elements to which they can be mapped, developing notions of maximality of matches becomes more involved.

*Example 1.* We consider the rules from Fig. 1 as the maximal rules of effect-oriented rules. In each case, there are different possibilities as to which subrule

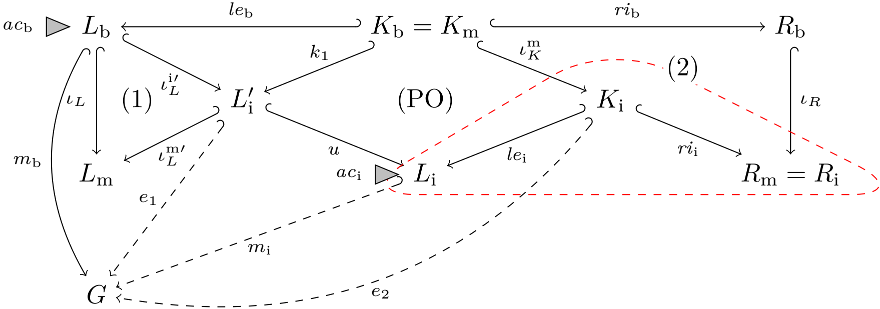



**Fig. 4.** Construction of an induced rule $\rho_{\mathrm{i}}$ (indicated via the red, dashed border) and of its match

of the rule to choose as the base rule. Our convention that the interfaces of the base and maximal rules of an effect-oriented rule coincide specifies that in each case the base rule contains at least the interface that is to be preserved. This minimal choice renders all deletions and creations potential.

Specifically, for the rule ensureThatClientHasAccAndPortfolio, one can assume that all elements to be created belong only to the maximal rule and represent potential creations. A possible alternative is to consider as the base rule the rule that creates an Account (together with its incoming edges), making the creation of a Portfolio and its incoming edges potential. Further combinations are possible.

An effect-oriented rule $\rho_{\mathrm{e}}$ represents a set of *induced rules*. The induced rules are constructed by extending the base rule of $\rho_{\mathrm{e}}$ with potential deletions and creations from the maximal rule. However, we require every induced rule to have the same RHS as the maximal rule. Potential creations are omitted in an induced rule by also incorporating them into the interface. This ensures that the state that is represented by the RHS of the maximal rule holds after applying an induced rule, even if not all potential creations are performed.

**Definition 6 (Induced rules).** *Given an effect-oriented rule $\rho_{\mathrm{e}} = (\rho_{\mathrm{b}}, \rho_{\mathrm{m}}, \iota)$, every rule $\rho_{\mathrm{i}} = (L_{\mathrm{i}} \xhookleftarrow{le_{\mathrm{i}}} K_{\mathrm{i}} \xhookrightarrow{ri_{\mathrm{i}}} R_{\mathrm{i}}, ac_{\mathrm{i}})$ is one of its* induced rules *if it is constructed in the following way (see (the upper part of) Fig. 4):*

1. *There is a factorisation (1) $\iota_L = \iota_L^{\mathrm{m}\prime} \circ \iota_L^{\mathrm{i}\prime}$ of $\iota_L \colon L_{\mathrm{b}} \hookrightarrow L_{\mathrm{m}}$ into two inclusions $\iota_L^{\mathrm{m}\prime}$ and $\iota_L^{\mathrm{i}\prime}$.*
2. *There is a factorisation (2) $\iota_R \circ ri_{\mathrm{b}} = ri_{\mathrm{m}} = ri_{\mathrm{i}} \circ \iota_K^{\mathrm{m}}$ of $ri_{\mathrm{m}} \colon K_{\mathrm{m}} \hookrightarrow R_{\mathrm{m}}$ into two inclusions $ri_{\mathrm{i}}$ and $\iota_K^{\mathrm{m}}$ such that the square (2) is a pullback.*
3. *$(L_{\mathrm{i}}, u, le_{\mathrm{i}})$ are computed as pushout of the pair of morphisms $(k_1, \iota_K^{\mathrm{m}})$, where $k_1 \coloneqq \iota_L^{\mathrm{i}\prime} \circ le_{\mathrm{b}}$; in that, we choose $L_{\mathrm{i}}$ such that $u$ and $le_{\mathrm{i}}$ become inclusions (employing renaming if necessary).*
4. *The application condition $ac_{\mathrm{i}}$ is computed as $\mathrm{Shift}(\iota_L^{\mathrm{i}}, ac_{\mathrm{b}})$, where $\iota_L^{\mathrm{i}} \coloneqq u \circ \iota_L^{\mathrm{i}\prime}$.*

*The* size *of an induced rule $\rho_{\mathrm{i}}$ is defined as $|\rho_{\mathrm{i}}| \coloneqq |L_{\mathrm{i}}' \setminus L_{\mathrm{b}}| + |K_{\mathrm{i}} \setminus K_{\mathrm{b}}|$.*

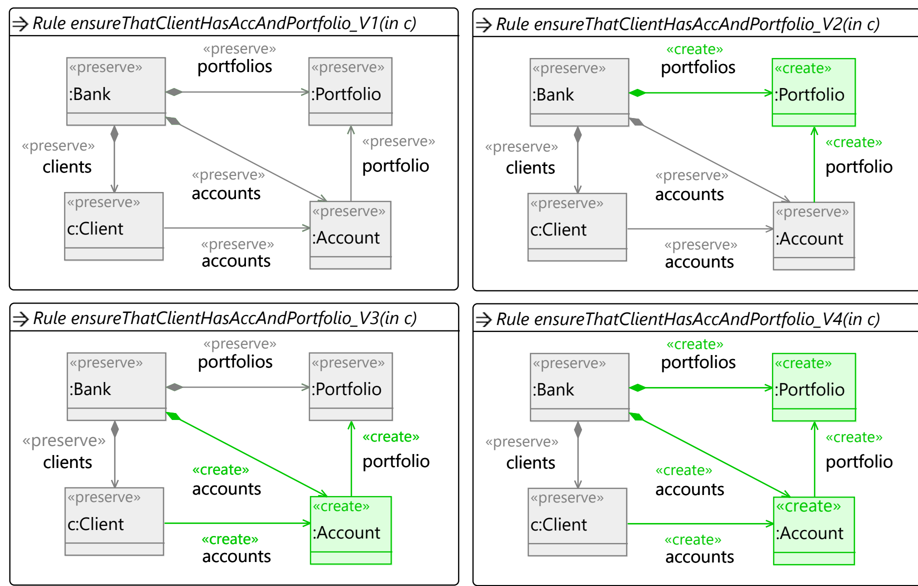



**Fig. 5.** Four induced rules arising from rule ensureThatClientHasAccAndPortfolio

*Example 2.* Consider again rule ensureThatClientHasAccAndPortfolio as a maximal rule and its preserved elements as the base rule. This effect-oriented rule has 26 induced rules, namely all rules that stereotype some of the «create»-elements of ensureThatClientHasAccAndPortfolio as «preserve». Figure 5 shows a selection of those, namely induced rules that reduce undesired reuse of elements. These are the rules where, *together with a node that is to be preserved (instead of being created), all adjacent edges that lead to preserved elements are also preserved.* We call this the *weak connectivity condition* and discuss and formalise it in [22]. An example for an induced rule that is not depicted is the rule that creates a new portfolio-edge between existing Accounts and Portfolios.

Next, consider the effect-oriented rule where the maximal rule ensureThatClientHasAccAndPortfolio is combined with the base rule that already creates an Account with its two incoming edges. Here, the Account cannot become a context in any induced rule because its creation is already required by the base rule. (This is ensured by the factorisation (2) in Fig. 4 being a pullback.) The induced rules of ensureThatClientHasNoAccAndPortfolio are obtained in a similar way.

Our first result states that an induced rule actually contains the base rule of its effect-oriented rule as a subrule. In particular, this ensures that, if an induced rule is applied, all actions specified by the base rule are performed.

**Proposition 1 (Base rule as subrule of induced rule).** *If $\rho_{\mathrm{i}}$ is an induced rule of the effect-oriented rule $\rho_{\mathrm{e}} = (\rho_{\mathrm{b}}, \rho_{\mathrm{m}}, \iota)$, then $\rho_{\mathrm{b}}$ is a subrule of $\rho_{\mathrm{i}}$ via the embedding $(\iota^{\mathrm{i}}_L, \iota^{\mathrm{i}}_K, \iota^{\mathrm{i}}_R)$, where $\iota^{\mathrm{i}}_L := u \circ \iota^{\mathrm{i}\prime}_L$, $\iota^{\mathrm{i}}_K := \iota^{\mathrm{m}}_K$ and $\iota^{\mathrm{i}}_R := \iota_R$ (compare Fig. 4).*

Next, we show that an effect-oriented rule indeed compactly represents a potentially large set of rules. The exact number of induced rules depends on how the edges of the maximal rule are connected.

**Proposition 2 (Number of induced rules).** *For an effect-oriented rule $\rho_\mathrm{e} = (\rho_\mathrm{b}, \rho_\mathrm{m}, \iota)$, the number $n$ of its induced subrules (up to isomorphism) satisfies:*

$$2^{|V_{L_\mathrm{m}} \setminus V_{L_\mathrm{b}}| + |V_{R_\mathrm{m}} \setminus V_{R_\mathrm{b}}|} \leq n \leq 2^{|V_{L_\mathrm{m}} \setminus V_{L_\mathrm{b}}| + |E_{L_\mathrm{m}} \setminus E_{L_\mathrm{b}}| + |V_{R_\mathrm{m}} \setminus V_{R_\mathrm{b}}| + |E_{R_\mathrm{m}} \setminus E_{R_\mathrm{b}}|} \ .$$

While our definition of induced rules is intentionally liberal, in many application cases it may be sensible to limit the kind of considered induced rules to avoid undesired reuse (e.g., connecting an Account to an already existing Portfolio of another Client). In [22], we provide a definition that enables that.

### 4.2 Matching Effect-oriented Rules

In this section, we develop different ways to match effect-oriented rules. *Effect-oriented transformations* get their semantics from 'classical' Double-Pushout transformations using the induced rules of the applied effect-oriented rule. We will develop different ways in which an existing context determines which induced rule of an effect-oriented rule should be applied at which match.

We assume a pre-match for the base rule of an effect-oriented rule to be given and try to to extend this pre-match to a match for an appropriate induced rule. The notion of *compatibility* captures this extension relationship. In [22], we present two technical lemmas that further characterise compatible matches.

**Definition 7 (Compatibility).** *Given an effect-oriented rule $\rho_\mathrm{e} = (\rho_\mathrm{b}, \rho_\mathrm{m}, \iota)$, a pre-match $m_\mathrm{b} \colon L_\mathrm{b} \hookrightarrow G$ for its base rule and a match $m_\mathrm{i} \colon L_\mathrm{i} \hookrightarrow G$ for one of its induced rules $\rho_\mathrm{i}$ are* compatible *if $m_\mathrm{i} \circ \iota^\mathrm{i}_L = m_\mathrm{b}$, where $\iota^\mathrm{i}_L = u \circ \iota^{\mathrm{i}\prime}_L \colon L_\mathrm{b} \hookrightarrow L_\mathrm{i}$ stems from the subrule embedding of $\rho_\mathrm{b}$ into $\rho_\mathrm{i}$ (compare Fig. 4 and Proposition 1).*

*An induced rule $\rho_\mathrm{i}$ can be* matched compatibly to $m_\mathrm{b}$ *if it has a match $m_\mathrm{i}$ such that $m_\mathrm{b}$ and $m_\mathrm{i}$ are compatible.*

Given an effect-oriented rule and a pre-match for its base rule, there can be many different induced rules for which there is a compatible match. The following definition introduces different strategies for selecting such a rule and match, so that the corresponding applications form transformations that are complete in terms of deletion and creation actions to achieve the intended effect, which is why they are called effect-oriented transformations. Their common core is that in any effect-oriented transformation, a pre-match of a base rule is extended by potential creations and deletions from the maximal rule such that no further extension is possible. Intuitively, this ensures that all possible potential deletions but only necessary potential creations are performed (in a sense we make formally precise in Theorem 1). A stricter notion is to maximise the number of reused elements.

**Definition 8 (Local completeness. Maximality. Effect-oriented transformation).** *Given a pre-match $m_\mathrm{b} \colon L_\mathrm{b} \to G$ for the base rule $\rho_\mathrm{b}$ of an effect-oriented rule $\rho_\mathrm{e} = (\rho_\mathrm{b}, \rho_\mathrm{m}, \iota)$, and a match $m_\mathrm{i}$ for one of its induced rules $\rho_\mathrm{i}$ that is compatible with $m_\mathrm{b}$, $\rho_\mathrm{i}$ and $m_\mathrm{i}$ are* locally complete *w.r.t. $m_\mathrm{b}$ if (see Fig. 4):*

1. Local completeness of additional deletions: *Any further factorisation* $\iota_L = \iota_L^{\mathrm{m}\prime\prime} \circ \iota_L^{\mathrm{i}\prime\prime}$ *into inclusions (with domain resp. co-domain* $L_{\mathrm{i}}''$*) such that there exists a non-bijective inclusion* $j\colon L_{\mathrm{i}}' \hookrightarrow L_{\mathrm{i}}''$ *with* $j \circ \iota_L^{\mathrm{i}\prime} = \iota_L^{\mathrm{i}\prime\prime}$ *and* $\iota_L^{\mathrm{m}\prime\prime} \circ j = \iota_L^{\mathrm{m}\prime}$ *meets one of the following two criteria.*
   - Not matchable: *There is no injective morphism* $e_1'\colon L_{\mathrm{i}}'' \hookrightarrow G$ *with* $e_1' \circ \iota_L^{\mathrm{i}\prime\prime} = m_{\mathrm{b}}$*.*
   - Not applicable: *Such an* $e_1'$ *exists, but the morphism* $m_{\mathrm{i}}'\colon L_{\mathrm{i}}'' \to G$ *which it induces together with the* right extension match $e_2$ *of* $m_{\mathrm{i}}$ *(where* $L_{\mathrm{i}}''$ *is the LHS of the induced rule that corresponds to this further factorisation) is not injective.*
2. Local completeness of additional creations: *Any further factorisation* $ri_{\mathrm{m}} = ri_{\mathrm{i}}' \circ \iota_K^{\mathrm{m}\prime}$ *into inclusions (with domain resp. co-domain* $K_{\mathrm{i}}'$*) such that there is a non-bijective inclusion* $j\colon K_{\mathrm{i}} \hookrightarrow K_{\mathrm{i}}'$ *with* $j \circ \iota_K^{\mathrm{m}} = \iota_K^{\mathrm{m}\prime}$ *and* $ri_{\mathrm{i}}' \circ j = ri_{\mathrm{i}}$ *meets one of the following two criteria.*
   - Not matchable: *There is no injective morphism* $e_2'\colon K_{\mathrm{i}}' \hookrightarrow G$ *with* $e_2' \circ \iota_K^{\mathrm{m}\prime} = m_{\mathrm{b}} \circ le_{\mathrm{b}}$*.*
   - Not applicable: *Such an* $e_2'$ *exists, but the morphism* $m_{\mathrm{i}}'\colon L_{\mathrm{i}}'' \to G$ *which it induces together with the* left extension match $e_1$ *of* $m_{\mathrm{i}}$ *(where* $L_{\mathrm{i}}''$ *is the LHS of the induced rule that corresponds to this further factorisation) is not injective.*

*An* effect-oriented transformation $t\colon G \Longrightarrow H$ *via* $\rho_{\mathrm{e}}$ *is a double-pushout transformation* $t\colon G \Longrightarrow_{\rho_{\mathrm{i}}, m_{\mathrm{i}}} H$*, where* $\rho_{\mathrm{i}}$ *is an induced rule of* $\rho_{\mathrm{e}}$ *and* $\rho_{\mathrm{i}}$ *is locally complete w.r.t.* $m_{\mathrm{b}} \coloneqq m_{\mathrm{i}} \circ \iota_L^{\mathrm{i}}$*, the induced pre-match for* $\rho_{\mathrm{b}}$*. The* semantics *of an effect-oriented rule is the collection of all of its effect-oriented transformations.*

*A transformation* $t\colon G \Longrightarrow_{\rho_{\mathrm{i}}, m_{\mathrm{i}}} H$ *via an induced rule* $\rho_{\mathrm{i}}$ *of a given effect-oriented rule* $\rho_{\mathrm{e}}$ *is* globally maximal *(w.r.t. G) if for any other transformation* $t'\colon G \Longrightarrow_{\rho_{\mathrm{i}}', m_{\mathrm{i}}'} H'$ *via an induced rule* $\rho_{\mathrm{i}}'$ *of* $\rho_{\mathrm{e}}$*, it holds that* $|\rho_{\mathrm{i}}| \geq |\rho_{\mathrm{i}}'|$*. Such a transformation t is* locally maximal *if for any other transformation* $t'\colon G \Longrightarrow_{\rho_{\mathrm{i}}', m_{\mathrm{i}}'} H'$ *via an induced rule* $\rho_{\mathrm{i}}'$ *of* $\rho_{\mathrm{e}}$ *where the induced pre-matches* $m_{\mathrm{b}}$ *and* $m_{\mathrm{b}}'$ *for the base rule coincide, it holds that* $|\rho_{\mathrm{i}}| \geq |\rho_{\mathrm{i}}'|$*. In all of these situations, we also call the match* $m_{\mathrm{i}}$ *and the rule* $\rho_{\mathrm{i}}$ locally complete *or* locally/globally maximal.

*Example 3.* To illustrate the different kinds of matching for effect-oriented rules, we again consider ensureThatClientHasAccAndPortfolio as a maximal rule whose «preserve»-elements form the base rule and apply it according to different semantics to the example instance depicted in Fig. 6. First, we consider the base match that maps the Client-node of the rule to Client c1 in the instance. Extending this base match in a *locally complete* fashion requires one to reuse the existing Portfolio and one of the existing Accounts. Choosing Account a2 leads to induced rule ensureThatClientHasAccAndPortfolio_V1 (Fig. 5) because there already exists an edge to Portfolio p. In contrast, choosing Account a1 leads to a transformation that creates a portfolio-edge from a1 to p (where the underlying induced rule is not depicted in Fig. 5). Both transformations are locally complete; in particular, locally complete matching is not deterministic. If, for semantic reasons, one wants to avoid transformations like the second one and only allows the induced rules

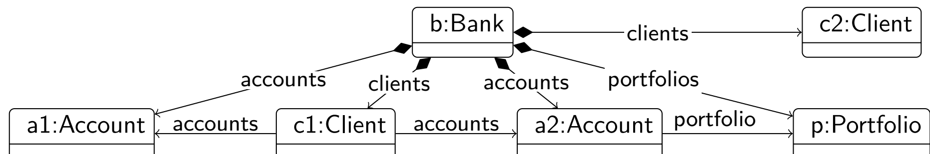



**Fig. 6.** Tiny example instance for the banking domain

that are depicted in Fig. 5, applying ensureThatClientHasAccAndPortfolio_V2 at Account a1 becomes locally complete.

The unique match for ensureThatClientHasAccAndPortfolio_V1 is the only locally maximal match compatible with the chosen base match in our example and is also globally maximal. To see a locally maximal match that is not globally maximal, we consider the base match that maps to Client c2 instead of c1. Here, the locally maximal match reuses a2, p and the portfolio-edge between them and creates the missing edges from c2 to a2 and p. Choosing Account a1 instead of a2 does not provide a locally maximal match as one cannot reuse a portfolio-edge then (reducing the size of the induced rule by 1). The globally maximal match remains unchanged as, by definition, it does not depend on a given base match. It is that evident that in a larger example, every Client with an Account with Portfolio constitutes a globally maximal match. Thus, also globally maximal matching is non-deterministic. In fact, one can even construct examples where globally maximal matches for different induced rules (of equal size) exist.

Unlike potential creations, potential deletions may require backtracking to find a match. To see this, consider the rule ensureThatClientHasNoAccAndPortfolio as a maximal rule whose «preserve»-elements form the base rule. Assuming that an Account can be connected to multiple Clients, a locally complete pre-match for an induced rule is not automatically a match for it. One has to look for an Account that is only connected to the matched Client.

The above example shows that none of the defined notions of transformation is deterministic. The situation is similar to Double-Pushout transformations in general, where the selection of the match is usually non-deterministic; however, in our case, there are different possible outcomes for the same base match. For the applications we are aiming at, such as rule-based search, graph repair or model synchronisation (see Sect. 6.1), this is not a problem. In these, it is often sufficient to know that, for instance, the selected Client has an Account and Portfolio after applying the rule ensureThatClientHasAccAndPortfolio but not necessarily important which ones.

It is easy to see that every globally maximal transformation via an induced rule is also locally maximal, and that every locally maximal transformation is locally complete. The definition of an effect-oriented transformation thus captures the weakest case and also covers locally and globally maximal transformations.

**Proposition 3 (Relations between different kinds of effect-oriented transformations).** *Given an effect-oriented rule* $\rho_{\mathrm{e}} = (\rho_{\mathrm{b}}, \rho_{\mathrm{m}}, \iota)$ *and a graph* $G$*,*

*every transformation* $t\colon G \Longrightarrow_{\rho_\mathrm{i},m_\mathrm{i}} H$ *via an induced rule* $\rho_\mathrm{i}$ *of* $\rho_\mathrm{e}$ *that is globally maximal is also locally maximal. Every locally maximal transformation is also locally complete for its induced pre-match* $m_\mathrm{b}$ *for the base rule* $\rho_\mathrm{b}$*.*

Intuitively, the maximal rule of an effect-oriented rule specifies a selection of potential actions. The induced rules result from the different possible combinations of potential actions. The next theorem clarifies which effects can ultimately occur *after* an effect-oriented transformation: If an element remains for which a matching potential deletion was specified by the effect-oriented rule, one of two alternatives took place: Either, the potential deletion was performed but on a different element (*alternative action*)—if there is more than one way to match an element potentially to be deleted. Or, that element was matched to by a potential creation (*alternative creation*). The latter can happen when a rule specifies potential creations and deletions for elements of the same type (at comparable positions). Similarly, if $x$ denotes a performed potential creation but there had been an element $y$ to which $x$ could have been matched, $y$ was used by another potential creation or deletion (*alternative action*). In particular, Theorem 1 also shows that effect-oriented rules can specify alternative actions. Their application can be non-deterministic, where one of several possible actions is chosen at random.

**Theorem 1 (Characterising effect-oriented transformations).** *Let* $\rho_\mathrm{e} = (\rho_\mathrm{b}, \rho_\mathrm{m}, \iota)$ *be an effect-oriented rule and* $t\colon G \Longrightarrow_{\rho_\mathrm{i},m_\mathrm{i}} H$ *an effect-oriented transformation via one of its induced rules* $\rho_\mathrm{i}$ *(compare Fig. 4 for the following).*

*Let* $x \in L_\mathrm{m} \setminus L_\mathrm{b}$ *be an element that represents a potential deletion of* $\rho_\mathrm{e}$ *and let* $K_\mathrm{b}^+$ *be the extension of* $K_\mathrm{b}$ *with* $x$ *(if defined as graph) and* $\iota^+\colon K_\mathrm{b} \hookrightarrow K_\mathrm{b}^+$ *the corresponding inclusion. If there exists an injective morphism* $m^+\colon K_\mathrm{b}^+ \hookrightarrow H$ *with* $m^+ \circ \iota^+ = n_\mathrm{i} \circ \iota_R \circ ri_\mathrm{b}$*, where* $n_\mathrm{i}$ *is the comatch of* $t$*, then either*

1. (Alternative action): *the element* $x$ *belongs to* $L_\mathrm{i}$*; in particular, an element of the same type as* $x$ *(and in comparable position) was deleted from* $G$ *by* $t$*; or*
2. (Alternative creation): *the element* $m^+(x)$ *of* $H$ *has a pre-image from* $R_\mathrm{i}$ *under* $n_\mathrm{i}$*, i.e., it was first created by* $t$ *or matched by a potential creation.*

*Similarly, let* $x \in R_\mathrm{i} \setminus (K_\mathrm{i} \cup R_\mathrm{b})$ *represent one of the potential creations of* $\rho_\mathrm{e}$ *that have been performed by* $t$*. Let* $K_\mathrm{b}^+$ *be the extension of* $K_\mathrm{b}$ *with* $x$ *(if defined as graph) and* $\iota^+\colon K_\mathrm{b} \hookrightarrow K_\mathrm{b}^+$ *the corresponding inclusion. Then either*

1. (Alternative action): *for every injective morphism* $m_\mathrm{b}^+\colon K_\mathrm{b}^+ \hookrightarrow G$ *with* $m_\mathrm{b}^+ \circ \iota^+ = m_\mathrm{i} \circ le_\mathrm{i} \circ \iota_K^\mathrm{m}$*, the element* $m_\mathrm{b}^+(x) \in G$ *has a pre-image from* $L_\mathrm{i}$ *under* $m_\mathrm{i}$ *(i.e., it is already mapped to by another potential action); or*
2. (Non-existence of match): *no injective morphism* $m_\mathrm{b}^+\colon K_\mathrm{b}^+ \hookrightarrow G$ *with* $m_\mathrm{b}^+ \circ \iota^+ = m_\mathrm{i} \circ le_\mathrm{i} \circ \iota_K^\mathrm{m}$ *exists.*

## 5 Locally Complete Matches—Algorithm and Implementation

In this section, we present an algorithm for the computation of a locally complete match (and a corresponding induced rule) from an effect-oriented rule and a

pre-match for its base rule. Starting with such a pre-match is well-suited for practical applications such as model repair and rule-based search, where a match of the base rule is often already fixed and needs to be complemented by (some of) the actions of the maximal rule. Note that this pre-match is common to all matches of all induced rules. Searching for the pre-match once and extending it contextually is generally much more efficient than searching for the matches of the induced rules from scratch. Moreover, we simultaneously compute a locally complete match and its corresponding induced rule and thus avoid first computing all induced rules (of which there can be many, cf. Proposition 2) and trying to match them in order of their size. The correctness of our algorithm is shown in Theorem 2 below. Note that we have focussed on the correctness of the algorithm and, apart from the basic efficiency consideration above, further optimisations for efficiency (incorporating ideas from [3]) are reserved for future work. We provide a short comment on our use of backtracking in [22]. In Sec. 5.2 we report on a prototype implementation of effect-oriented transformations in Henshin using this algorithm.

### 5.1 An Algorithm for Computing Locally Complete Matches

We consider the problem of finding a locally complete match from a given pre-match. So-called *rooted rules*, i.e., rules where a partial match is fixed (or at least can be determined in constant time), have been an important part of the development of rule-based algorithms for graphs that run in linear time [3,6]. Note that we are not looking for an induced rule and match that lead to a maximal transformation but only to a non-extensible, i.e., locally complete one. This has the effect that the dangling-edge condition remains the only possible source of backtracking in the matching process.

In Algorithm 1, we outline a function that extends a pre-match for a base rule of an effect-oriented rule to a compatible, locally complete match for a corresponding induced rule. The input to our algorithm is an effect-oriented rule $\rho_{\mathrm{e}} = (\rho_{\mathrm{b}}, \rho_{\mathrm{m}}, \iota)$ (the parameter *rule*), a graph $G$ (the parameter *graph*) and a pre-match $m_{\mathrm{b}} \colon L_{\mathrm{b}} \hookrightarrow G$ for $\rho_{\mathrm{b}}$ (the parameter *currentMappings*). It returns a match $m_{\mathrm{i}}$ for an induced rule $\rho_{\mathrm{i}}$ of $\rho_{\mathrm{e}}$ such that $m_{\mathrm{i}}$ and $m_{\mathrm{b}}$ are compatible and $m_{\mathrm{i}}$ is locally complete; it returns null if and only if no such compatible, locally complete match exists. We outline the matching of nodes and consequently consider *currentMappings* to be a list of node mappings; from this, one can infer the matching of edges. The computed match also represents the corresponding induced rule. We provide the details for these conventions in [22].

The search for a match starts with initialising *unboundNodes* with the potential actions of the given effect-oriented rule (line 5). Then the function *findExtension* recursively tries to match those, extending the pre-match (line 6). Function *findExtension* works as follows. For the unbound node $n$ at the currently considered *position*, all available candidates, i.e., all nodes $x$ in the graph $G$ to which $n$ can be mapped, are collected with the function *findExtensionCandidates* (line 10). A candidate $x$ must satisfy the following properties: (i) no other node may already map to $x$, i.e., $x$ does not yet occur in *currentMappings* (injectivity condition)

and (ii) the types of $n$ and $x$ must coincide (consistency condition). If candidates exist, for each candidate $x$, the algorithm tries to map $n$ to $x$ until a solution is found. To do this, the set of current mappings is extended by the pair $(n, x)$ (line 13). If $n$ was the last node to be matched (line 14) and the morphism defined by *currentMappings* satisfies the dangling-edge condition (line 15), the result is returned as solution (line 16). If the dangling-edge condition is violated, the selected candidate is removed and the next one is tried (line 17). If $n$ was not the last node to be mapped (line 18), the function findExtension is called for the extended list of current mappings and the next unmatched node from *unboundNodes* (line 19). If this leads to a valid solution, this solution is returned (lines 20–21). Otherwise, the pair $(n, x)$ is removed from *currentMappings* (line 23), and the next candidate is tried. If candidates exist but none of them lead to a valid solution, null is returned (line 33). If no candidate exists (line 25), either the current mapping or null is returned as the solution (if $n$ was the last node to assign; lines 26–29) or *findExtension* is called for the next *position* (line 31), i.e., the currently considered node $n$ is omitted from the mapping (and, hence, from the induced rule).

**Algorithm 1.** Computation of a locally complete match

```
1  input: effect−oriented rule (ρ_b, ρ_m, ι), graph G, and a pre−match m_b
2  output: locally complete match m_i compatible with m_b
3
4  function findLocallyCompleteMatch(rule, graph, currentMappings)
5    unboundNodes = V_{L_m} \ V_{L_b} ⊔ V_{R_m} \ V_{R_b};
6    return findExtension(currentMappings, graph, unboundNodes, 0);
7
8  function findExtension(currentMappings, graph, unboundNodes, position)
9    n = unboundNodes.get(position);
10   candidates = findExtensionCandidates(currentMappings, graph, n);
11   if (!candidates.isEmpty())
12     for each x in candidates
13       currentMappings.put(n,x);
14       if (position == unboundNodes.size() − 1) //last node to be matched
15         if (danglingEdgeCheck(graph, currentMappings))
16           return currentMappings;
17         else currentMappings.remove(n,x);
18       else //map next unbound node
19         nextSolution = findExtension(currentMappings, graph, unboundNodes,
               position + 1);
20         if (nextSolution != null)
21           return nextSolution;
22         else //try next candidate
23             currentMappings.remove(n,x);
24     end for //no suitable candidate found
25   else //there is no candidate for the current node
26     if (position == unboundNodes.size() − 1) //last node to be matched
27       if (danglingEdgeCheck(graph, currentMappings))
28         return currentMappings;
```

```
29       else return null;
30     else //map next unbound node
31       return findExtension(currentMappings, graph, unboundNodes, position+1);
32   //no candidate led to a valid mapping
33   return null;
```

**Theorem 2 (Correctness of Algorithm 1).** *Given an effect-oriented rule $\rho_e = (\rho_b, \rho_m, \iota)$, a graph $G$, and a pre-match $m_b \colon L_b \hookrightarrow G$, Algorithm 1 terminates and computes an induced rule $\rho_i$ with a match $m_i$ such that $m_i$ and $m_b$ are compatible and $\rho_i$ is locally complete w.r.t. $m_b$. In particular, Algorithm 1 returns* null *if and only if no induced rule of $\rho_e$ can be matched compatibly with $m_b$ and returns $\rho_b$ as induced rule with match $m_b$ if and only if $m_b$ is locally complete and a match.*

### 5.2 Implementation

In this section, we present a prototypical implementation of effect-oriented transformations in Henshin [1,29], a model transformation language based on graph transformation. From the user perspective, the implementation includes two major classes for applying a given effect-oriented rule $\rho_e = (\rho_b, \rho_m, \iota)$ to a host graph $G$: the class `LocallyCompleteMatchFinder` for finding a locally complete match, and the class `EffectOrientedRuleApplication` to apply a rule $\rho_e = (\rho_b, \rho_m, \iota)$ at such a match. We assume that $\rho_e = (\rho_b, \rho_m, \iota)$ is provided as simple Henshin rule representing the maximal rule $\rho_m$, where the base rule $\rho_b$ is implicitly represented by the preserved part of the rule. The host graph $G$ is provided, as usual in Henshin, in the form of a model instance for a given meta-model (representing the type graph).

The implementation follows the algorithm presented in Section 5.1. Our main design goal was to reuse the existing interpreter core of Henshin, with its functionalities for matching and rule applications, as much as possible. In particular, in `LocallyCompleteMatchFinder`, we derive the base rule $\rho_b$ by creating a copy of $\rho_m$ with creations and deletions removed, and feeding it into the interpreter core to obtain a pre-match $m_b$ on $G$. For cases where a pre-match $m_b$ can be found, we provide an implementation of Algorithm 1 that produces a partial match $\hat{m}_m$ incorporating the mappings of $m_b$ and additional mappings for elements to be deleted and elements not to be created. In order to treat elements of different actions consistently, we perform these steps on an intermediate rule $\rho_{gr}$, called the *grayed* rule, in which creations are converted to preserve actions. In `EffectOrientedRuleApplication`, we first derive the induced rule $\rho_i$ from $\hat{m}_m$, such that $\hat{m}_m$ is a complete match for $\rho_i$. For the actual rule application, we feed $\rho_i$ together with $\hat{m}_m$ into the Henshin interpreter core using a classical rule application.

We have tested the implementation using our running example. For this purpose, we specified all rules and an example graph. Our implementation behaved completely as expected. The source code of the implementation and the example are available online at `https://github.com/dstrueber/effect-oriented-gt`.

# 6 Related Work

In this section, we describe how existing practical applications could benefit from the use of effect-oriented transformations (Sect. 6.1) and relate effect-oriented graph transformations to other graph transformation approaches (Sect. 6.2).

## 6.1 Benefiting from Effect-oriented Graph Transformation

There are several application cases in the literature where graph transformation has been used to achieve certain states. In the following, we recall graph repair, where a consistent graph is to be achieved, and model synchronisation, where consistent model relations are to be achieved after one of the models has been changed. A slightly different case is service matching, where a specified service should be best covered by descriptions of existing services.

In their rule-based approach to *graph repair* [28], Sandmann and Habel repair (sub-)conditions of the form $\exists(a\colon B \hookrightarrow C)$ using the (potentially large) set of rules that, for every graph $B'$ between $B$ and $C$, contains the rule that creates $C$ from $B'$. Negative application conditions (NACs) ensure that the rule with the largest possible $B'$ as LHS is selected during repair. With effect-oriented graph transformation, the entire set of rules derived by them can be represented by a single effect-oriented rule that has the identity on $B$ as base rule and $B \hookrightarrow C$ as maximal rule. Moreover, we do not need to use NACs, since locally complete matching achieves the desired effect.

In the context of model synchronisation, a very similar situation occurs in [26]. There, Orejas et al. define consistency between pairs of models via *patterns*. For synchronisation, a whole set of rules is derived from a single pattern to account for the different ways in which consistency might be restored (i.e., to create the missing elements). Again, NACs are used to control the application of the rules. As above, we can represent the whole set of rules as a single effect-oriented rule.

Fritsche, Kosiol, et al. extend *TGG-based model synchronisation processes* to achieve higher incrementality using special repair rules [14,13,20]. Elements to be deleted according to these rules may be deleted for other reasons during the synchronisation process, destroying the matches needed for the repair rules. In [14], this problem is avoided by only considering edits where this cannot happen. In [13], Fritsche approaches this problem pragmatically by omitting such deletions on-the-fly—if an element is already missing that needs be deleted to restore consistency, the consistency has already been restored locally and the deletion can simply be skipped. More formally, Kosiol in [20] presents a set of subrules of a repair rule, where the maximal one matchable can always be chosen to perform the propagation. This whole set of rules can also be elegantly represented by a single effect-oriented rule.

In [2], Arifulina addresses the heterogeneity of *service specifications and descriptions*. She develops a method for matching service specifications by finding a maximal partial match for a rule that specifies a service. Apart from the fact that Arifulina allows the partial match to also omit context elements, the problem

can also be formulated as finding a globally maximal match (in our sense) of the service-specifying rule in the LHS of available service description rules.

### 6.2 Relations to Other Graph Transformation Approaches

There are several approaches to graph transformation that take the variability of the transformation context into account. We recall each of them briefly and discuss the commonalities and differences to effect-oriented graph transformation.

**Other semantics for applying single transformation rules.** Graph transformation approaches such as the single pushout approach [25], the sesqui-pushout approach [9], AGREE [7], PBPO [8], and PBPO$^+$ rewriting [27] are more expressive than DPO rewriting, as they allow some kind of copying or merging of elements or (implicitly specified) side effects. Rules are defined as (extended) spans in all these approaches. Therefore, they also specify sets of actions that must be executed in order to apply a rule. AGREE, PBPO, and PBPO$^+$ would enable one to specify what we call potential deletions; however, in these approaches their specification is far more involved. None of the mentioned approaches supports specifying potential creations that can be omitted depending on the currently considered application context.

In the *Double-Pullback approach*, 'a rule specifies only what at least has to happen on a system's state, but it allows to observe additional effects which may be caused by the environment' [18, p. 85]. In effect-oriented graph transformation, the base rule also specifies what has to happen as a minimum. But the additional effects are not completely arbitrary, as the maximal rule restricts the additional actions. Moreover, these additional actions are to be executed only if the desired state that they specify does not yet exist. This suggests that double-pullback transformations are a more general concept than effect-oriented transformations with locally complete matching.

**Effect-oriented transformations via multiple transformation rules.** In the following, we discuss how graph transformation concepts that apply several rules in a controlled way can be used to emulate effect-oriented transformations.

*Graph programs* [17,6] usually provide control constructs for rule applications such as sequential application, conditional and optional applications, and application loops. To emulate effect-oriented graph transformations, the base rule would be applied first and only once. For each induced rule, we would calculate the remainder rule, which is the difference to the base rule, i.e., it specifies all actions of the induced rule that are not specified in the base rule. To choose the right remainder rule, we would need a set of additional rules that check which actions still need to be executed in the given instance graph. Depending on these checks, the appropriate remainder rule is selected and applied.

*Amalgamated transformations* [4,12] are useful when graph transformation with universally quantified actions are required. They provide a formal basis for *interaction schemes* where a *kernel rule* is applied exactly once and additional

*multi-rules*, extending the kernel rule, are applied as often as possible at matches extending the one of the kernel rule. To emulate the behaviour of an effect-oriented transformation by an interaction scheme, the basic idea is to generate the set of multi-rules as all possible induced rules. Application conditions could be used to control that the 'correct' induced rule is applied.

A compact representation of a rule with several variants is given by *variability-based (VB) rules* [30]. A VB rule represents a set of rules with a common core. Elements that only occur in a subset of the rules are annotated with so-called *presence conditions*. A VB rule could compactly represent the set of induced rules of an effect-oriented rule, albeit in a more complicated way, by explicitly defining a list of features and using them to annotate variable parts. An execution semantics of VB rules has been defined for single graphs [30] and sets of variants of graphs [**?**]. However, for VB rules, the concept of driving the instance selection by the availability of a match with certain properties has not been developed.

## 7 Conclusion

Effect-oriented graph transformation supports the modelling of systems in a more declarative way than the graph transformation approaches in the literature. The specification of basic actions is accompanied by the specification of desired states to be achieved. Dependent on the host graph, the application of a base rule is extended to the application of an induced rule that performs exactly the actions required to achieve the desired state. We have discussed that effect-oriented transformations are well suited to specify graph repair and model synchronisation strategies, since change actions can be accompanied by actions that restore consistency within a graph or between multiple (model) graphs. We have outlined how existing approaches to graph transformation can be used to emulate effect-oriented transformation but lead to accidental complexity that effect-oriented transformations can avoid.

In the future, we are especially interested in constructing effect-oriented rules that induce consistency-sustaining and -improving transformations [21]. Examining the computational complexity of different approaches for their matching, developing efficient algorithms for the computation of their matches, elaborating conflict and dependency analysis for effect-oriented rules, and combining effect-orientation with multi-amalgamation are further topics of theoretical and practical interest.

**Acknowledgements**

We thank the anonymous reviewers for their constructive feedback. Parts of the research for this paper have been performed while J. K. was a Visiting Research Associate at King's College London. This work has been partially supported by the Deutsche Forschungsgemeinschaft (DFG), grant TA 294/19-1.

## References


1. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: Advanced concepts and tools for in-place EMF model transformations. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) Model Driven Engineering Languages and Systems – 13th International Conference, MODELS 2010, Oslo, Norway, October 3–8, 2010, Proceedings, Part I. Lecture Notes in Computer Science, vol. 6394, pp. 121–135. Springer (2010). https://doi.org/10.1007/978-3-642-16145-2_9, https://doi.org/10.1007/978-3-642-16145-2_9
2. Arifulina, S.: Solving heterogeneity for a successful service market. Ph.D. thesis, University of Paderborn, Germany (2017). https://doi.org/10.17619/UNIPB/1-13
3. Bak, C., Plump, D.: Rooted graph programs. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. **54** (2012). https://doi.org/10.14279/tuj.eceasst.54.780, https://doi.org/10.14279/tuj.eceasst.54.780
4. Boehm, P., Fonio, H., Habel, A.: Amalgamation of graph transformations with applications to synchronization. In: Ehrig, H., Floyd, C., Nivat, M., Thatcher, J.W. (eds.) Mathematical Foundations of Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Berlin, Germany, March 25-29, 1985, Volume 1: Colloquium on Trees in Algebra and Programming (CAAP'85). Lecture Notes in Computer Science, vol. 185, pp. 267–283. Springer (1985). https://doi.org/10.1007/3-540-15198-2_17, https://doi.org/10.1007/3-540-15198-2_17
5. Burdusel, A., Zschaler, S., John, S.: Automatic generation of atomic multiplicity-preserving search operators for search-based model engineering. Software Systems Modelling (2021). https://doi.org/10.1007/s10270-021-00914-w
6. Campbell, G., Courtehoute, B., Plump, D.: Fast rule-based graph programs. Sci. Comput. Program. **214**, 102727 (2022). https://doi.org/10.1016/j.scico.2021.102727, https://doi.org/10.1016/j.scico.2021.102727
7. Corradini, A., Duval, D., Echahed, R., Prost, F., Ribeiro, L.: AGREE – algebraic graph rewriting with controlled embedding. In: Parisi-Presicce, F., Westfechtel, B. (eds.) Graph Transformation – 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21–23, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9151, pp. 35–51. Springer (2015). https://doi.org/10.1007/978-3-319-21145-9_3, https://doi.org/10.1007/978-3-319-21145-9_3
8. Corradini, A., Duval, D., Echahed, R., Prost, F., Ribeiro, L.: The PBPO graph transformation approach. J. Log. Algebraic Methods Program. **103**, 213–231 (2019). https://doi.org/10.1016/j.jlamp.2018.12.003, https://doi.org/10.1016/j.jlamp.2018.12.003
9. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17–23, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4178, pp. 30–45. Springer (2006). https://doi.org/10.1007/11841883_4, https://doi.org/10.1007/11841883_4
10. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation – Part I: basic concepts and double pushout approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, pp. 163–246. World Scientific (1997)

11. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2006). https://doi.org/10.1007/3-540-31188-2, https://doi.org/10.1007/3-540-31188-2
12. 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. Math. Struct. Comput. Sci. **24**(4) (2014). https://doi.org/10.1017/S0960129512000357, https://doi.org/10.1017/S0960129512000357
13. Fritsche, L.: Local Consistency Restoration Methods for Triple Graph Grammars. Ph.D. thesis, Technical University of Darmstadt, Germany (2022), http://tuprints.ulb.tu-darmstadt.de/21443/
14. Fritsche, L., Kosiol, J., Schürr, A., Taentzer, G.: Avoiding unnecessary information loss: correct and efficient model synchronization based on triple graph grammars. Int. J. Softw. Tools Technol. Transf. **23**(3), 335–368 (2021). https://doi.org/10.1007/s10009-020-00588-7, https://doi.org/10.1007/s10009-020-00588-7
15. Golas, U., Habel, A., Ehrig, H.: Multi-amalgamation of rules with application conditions in $\Updownarrow$-adhesive categories. Math. Struct. Comput. Sci. **24**(4) (2014). https://doi.org/10.1017/S0960129512000345
16. Habel, A., Pennemann, K.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. **19**(2), 245–296 (2009). https://doi.org/10.1017/S0960129508007202
17. Habel, A., Plump, D.: Computational completeness of programming languages based on graph transformation. In: Honsell, F., Miculan, M. (eds.) Foundations of Software Science and Computation Structures, 4th International Conference, FOSSACS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2–6, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2030, pp. 230–245. Springer (2001). https://doi.org/10.1007/3-540-45315-6_15, https://doi.org/10.1007/3-540-45315-6_15
18. Heckel, R., Ehrig, H., Wolter, U., Corradini, A.: Double-pullback transitions and coalgebraic loose semantics for graph transformation systems. Appl. Categorical Struct. **9**(1), 83–110 (2001). https://doi.org/10.1023/A:1008734426504
19. Horcas, J.M., Strüber, D., Burdusel, A., Martinez, J., Zschaler, S.: *We're Not Gonna Break It!* Consistency-Preserving Operators for Efficient Product Line Configuration. IEEE Transactions on Software Engineering (2022). https://doi.org/10.1109/TSE.2022.3171404
20. Kosiol, J.: Formal Foundations for Information-Preserving Model Synchronization Processes Based on Triple Graph Grammars. Ph.D. thesis, University of Marburg, Germany (2022), https://archiv.ub.uni-marburg.de/diss/z2022/0224
21. Kosiol, J., Strüber, D., Taentzer, G., Zschaler, S.: Sustaining and improving graduated graph consistency: A static analysis of graph transformations. Science of Computer Programming **214** (2021)
22. Kosiol, J., Strüber, D., Taentzer, G., Zschaler, S.: Finding the right way to rome: Effect-oriented graph transformation (2023), https://arxiv.org/abs/2305.03432
23. Krause, C.: Bank accounts example. Online (2023), https://wiki.eclipse.org/Henshin/Examples/Bank_Accounts
24. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. RAIRO Theor. Informatics Appl. **39**(3), 511–545 (2005). https://doi.org/10.1051/ita:2005028, https://doi.org/10.1051/ita:2005028
25. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theor. Comput. Sci. **109**(1&2), 181–224 (1993). https://doi.org/10.1016/0304-3975(93)90068-5, https://doi.org/10.1016/0304-3975(93)90068-5

26. Orejas, F., Guerra, E., de Lara, J., Ehrig, H.: Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009, Udine, Italy, September 7–10, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5728, pp. 383–397. Springer (2009). https://doi.org/10.1007/978-3-642-03741-2_26, https://doi.org/10.1007/978-3-642-03741-2_26
27. Overbeek, R., Endrullis, J., Rosset, A.: Graph rewriting and relabeling with pbpo$^{+}$. In: Gadducci, F., Kehrer, T. (eds.) Graph Transformation – 14th International Conference, ICGT 2021, Held as Part of STAF 2021, Virtual Event, June 24–25, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12741, pp. 60–80. Springer (2021). https://doi.org/10.1007/978-3-030-78946-6_4, https://doi.org/10.1007/978-3-030-78946-6_4
28. Sandmann, C., Habel, A.: Rule-based graph repair. In: Echahed, R., Plump, D. (eds.) Proceedings Tenth International Workshop on Graph Computation Models, GCM@STAF 2019, Eindhoven, The Netherlands, 17th July 2019. EPTCS, vol. 309, pp. 87–104 (2019). https://doi.org/10.4204/EPTCS.309.5, https://doi.org/10.4204/EPTCS.309.5
29. Strüber, D., Born, K., Gill, K.D., Groner, R., Kehrer, T., Ohrndorf, M., Tichy, M.: Henshin: A usability-focused framework for emf model transformation development. In: ICGT'17: International Conference on Graph Transformation. pp. 196–208. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61470-0_12, https://doi.org/10.1007/978-3-319-61470-0_12
30. Strüber, D., Peldszus, S., Jürjens, J.: Taming multi-variability of software product line transformations. In: FASE. pp. 337–355 (2018). https://doi.org/10.1007/978-3-319-89363-1_19, https://doi.org/10.1007/978-3-319-89363-1_19
31. Strüber, D., Rubin, J., Arendt, T., Chechik, M., Taentzer, G., Plöger, J.: Variability-based model transformation: formal foundation and application. Formal Aspects Comput. **30**(1), 133–162 (2018). https://doi.org/10.1007/s00165-017-0441-3