Non-blocking consistency-preserving subgraph transformation systems via activation diagrams

Jens Kosiol[a], Steffen Zschaler[b]

[a] *Philipps-Universität Marburg, Marburg, Germany*
[b] *King's College London, London, UK*

Abstract

Consistency-preserving graph-transformation rules, which do not worsen the consistency of a graph, are important in many contexts, including model-driven optimisation and model repair. However, while it is comparatively easy to produce rules that are only applicable when they will not introduce new constraint violations—we call such rules *blocking* rules—solutions for producing rules that actively repair any violations they might introduce—*non-blocking* rules—are currently only available for very specific cases. In this paper, we build on recent work on generating consistency-preserving rules for optimal feature configuration and show how this can be generalised to subgraph replacement systems for a broad class of constraints and initial rules. We develop a formal generalisation of activation diagrams for subgraph replacement systems and show how these can be used to read off variability-based rules as a compact encoding of all possible repair options.

Keywords: Graph transformation, Double-pushout rewriting, Consistency preservation

1. Introduction

In the graph-transformation literature, significant research has focused on how to ensure graph transformations keep a graph consistent with a given set of constraints (or make the graph consistent with the constraints if they are not yet consistent)—for example, see [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. However, producing consistency-preserving graph-transformation rules can be challenging. While it is well understood how an existing graph transformation rule can be augmented with additional (negative) application conditions to ensure it can only be applied if it will not invalidate a given constraint [6, 7, 8, 11, 16], such augmentation invariably limits the applicability of the rule. We call such rules *blocking* rules, as they block the application of the transformation if it would lead to inconsistency. Instead, we are interested in how a given rule can be augmented (retaining all of its actions) to a consistency-preserving rule *without restricting its applicability,* by adding actions that will repair any constraint violations introduced by the original rule. We call such rules *non-blocking* rules.

In [10], Horcas et al. generate *consistency-preserving configuration operators* (CPCOs)—an example of non-blocking rules—that are then used (in evolutionary search) to find optimal configurations (i.e., sets of selected features) for (software) product lines, outperforming the previous state of the art. The central new concept is the one of a *feature-activation diagram*, capturing, for every feature, which other features have to be (de-)selected along with the (de-)selection of that feature to obtain a valid configuration. Horcas et al. show that such a feature-activation diagram is a compact and efficiently computable encoding of all relevant information about the interaction between the (de-)selection of the individual features and the validity of configurations. Importantly, for any feature (de-)selection there are usually several ways to amend a feature configuration to preserve consistency; a feature-activation diagram captures all of them compactly for all possible feature (de-)selections. From the information in a feature-activation diagram, CPCOs can be derived

Email addresses: kosiolje@mathematik.uni-marburg.de (Jens Kosiol), szschaler@acm.org (Steffen Zschaler)

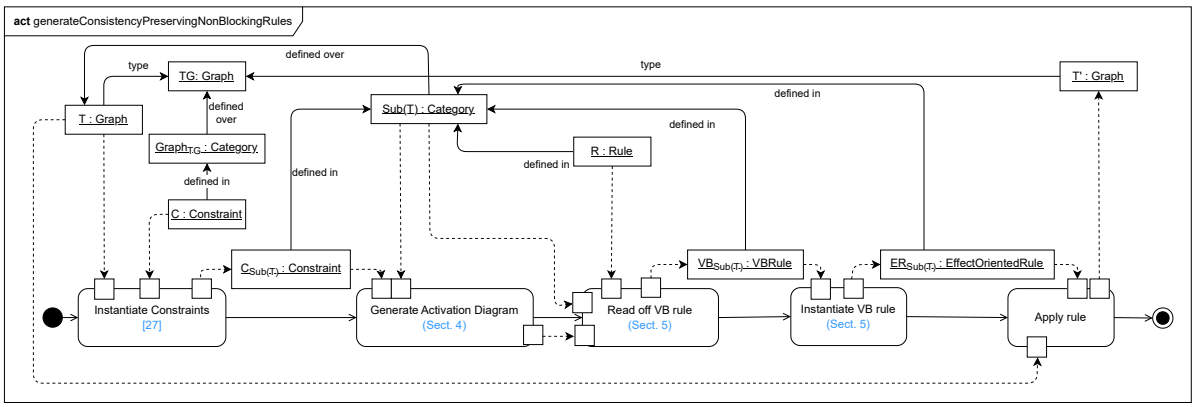


Figure 1: Overview of our approach. The top half of the figure shows different objects being used or produced by our approach, while the bottom half shows the sequence of steps, indicating which section discusses which step.

efficiently. To avoid having to explicitly enumerate all possible ways to repair configurations as individual CPCO rules, Horcas et al. encode all CPCOs for one feature (de-)selection as a single *variability-based rule* (VB rule[1]) [17]; different instantiations of the VB rule then correspond to the different possible ways to repair configurations.

In this paper, we ask

> **RQ:** How, and under what conditions, can the ideas from [10] be generalised to general graph-transformation rules and general graph constraints?

Such a generalisation is, for example, helpful in model-driven optimisation [18], where graph-transformation rules are used to capture the way in which candidate solutions can be changed to derive new solutions; they are, thus, used to characterise the search space. Many approaches (e.g., [13, 19, 20]) use graph-transformation rules directly to define neighbourhoods for evolutionary search, acting as mutation operators. In other approaches [21], matches of the rules are used to generate decision variables for a (M)ILP problem that is then solved by a standard (M)ILP solver. In either scenario, rules that are consistency-preserving [9]—that is, transforming consistent graphs into consistent graphs—are preferable. For search-based solutions, they improve the effectiveness of the search [10, 13, 18, 22], and for (M)ILP-based solvers, they can help reduce the number of matches and decision variables to consider. Similarly, many model-repair approaches [23] use graph-transformation rules to encode the changes to be made to a model to repair inconsistencies (e.g., [1, 2, 24]). Obviously, these graph-transformation rules should be consistency-improving [9].

The key observation for our generalisation is that product line configuration (as discussed in [10]) is essentially what we will call a *selection problem*: A feature model provides the set of available features and a (valid) configuration consists of a subset of these features (that satisfies additional *feasibility constraints*). Accordingly, we will denote as "selection problems" problems where the task is to select a (valid) group of a given set of options.

In this paper, we formally characterise such selection problems and then generalise the approach by Horcas et al. [10] to these. Figure 1 summarises our overall approach. A key idea for our generalisation is that selection problems can be treated in *subgraph transformation systems* [25, 26]. The objects of such a system are all subgraphs of a given finite *container graph* T (where the container graph T, in turn, is typed over a *type graph* TG to be able to associate types/roles with its elements). When compared to the setting of product line configuration, the set of all features corresponds to the container graph, configurations correspond to subgraphs of the container graph and the selection or deselection of features corresponds to incorporating or removing an element of the container graph T into or from a subgraph of T.

[1] A compact way to encode a family of related graph-transformation rules.

The first central step of our approach is to derive an *activation diagram*, similar in spirit to the feature activation diagrams used in [10], from constraints defined over a subgraph transformation system. For this derivation, the constraints have to be given in a suitable normal form. In [27], we have worked out that (i) every nested constraint over a subgraph transformation system can be equivalently transformed into the necessary non-nested normal form and (ii) that every traditional nested constraint over graphs can be equivalently translated into a constraint over a subgraph transformation system. (This translation makes our approach more convenient to use because traditional constraints are much more compact than their counterparts over subgraph transformation systems.) Regarding the activation diagrams we construct, we assess their size (in particular, they are finite; Proposition 4) and characterize in which sense they actually capture the interplay of graph modifications with the validity of constraints (Theorem 5); these contributions on activation diagrams are the content of Section 4.

From the activation diagrams, we can read off new, consistency-preserving transformation rules, which we compactly encode as variability-based rules, to avoid having to explicitly specify every possible alternative repair action for any constraint violations that might be introduced by the original rule; we call the read-off rules *consistency-preserving variability-based extensions* of the originally given rules. To further reduce the number of consistency-preserving transformation rules we need to derive from the VB rules, we derive those as so-called *effect-oriented rules* [28]—these allow for declaring *potential actions*, making it possible to compactly specify a rule for a whole range of situations by, e.g., skipping the deletion of elements that do not exist anyhow. We refer to these resulting rules as *non-blocking consistency-preserving variants*. Our contributions regarding this construction of rules is a series of results characterising their properties; most importantly, the rules are indeed consistency-preserving (Theorem 8), we can give reasonable guarantees for their existence (Theorem 9) and we can provide easy-to-meet circumstances under which our derived consistency-preserving rules can simulate transformation sequences by the original rules that result in valid graphs (Theorem 10). Section 5 presents these constructions of and results about rules.

To illustrate our results and put them into context, we introduce examples (in particular, also our running example) in Sect. 2, recall the necessary formal background in Sect. 3 and discuss related works in Sect. 6. In Sect. 7, we conclude and offer an outlook on potential avenues for further research. In our appendix, we show how our presented approach indeed generalizes the feature activation diagrams of Horcas et al. [10] (Appendix A) and present all proofs, which we defer from the main part of the paper (Appendix B).

2. Examples

In this section, we first briefly discuss several examples of selection problems, before describing one—class responsibility assignment—in more detail so that we can use it as a running example throughout the remainder of the paper.

2.1. Example selection problems

In Sect. 1, we characterised *selection problems* as problems where the task is to select a (valid) group of a given set of options. This characterisation covers a wide range of problems: in particular, any problem where different solutions can be established via the presence or absence of edges between a fixed finite set of nodes, or the presence or absence of nodes (and possibly edges between them) from a fixed and finite set of nodes, is a selection problem. Basically, these are combinatorial problems where we do not allow the creation of new elements. Examples of such problems include:

- The *model-repair problem* [23] aims to transform models that violate some consistency constraint into new models that have fewer or no remaining constraint violations. Not every model-repair scenario is a selection problem, but a significant subset of model-repair problems are. Specifically, repairs that are limited to deleting existing model elements or creating/deleting edges between existing model elements are selection problems: we can imagine rephrasing them into the corresponding situation where we generate all possible edges between the existing model elements and ask which model elements and edges should be selected for a repaired model.

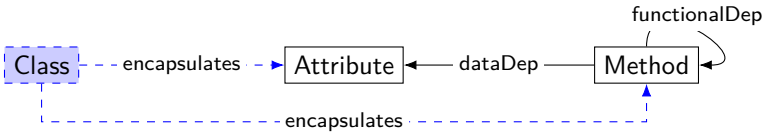


Figure 2: A meta-model (type graph) TG for defining instances of the CRA problem (based on [33])

- *Optimal feature configuration* considers a set of features of a software product line—including constraints capturing interdependencies between these features—and aims to identify a subset of these features that is both valid in terms of the constraints and optimal in terms of other properties (e.g., each feature may make a contribution to overall weight or energy consumption and the goal is to minimise these). This is a selection problem, because the goal is to choose a subset from a given, finite set of features. We have previously presented an approach to optimal feature configuration [10], which we generalise in this paper (cf. Appendix A).
- *Quantum program synthesis* [29], wherein the goal is to determine a sequential combination of quantum operators applied to a finite set of quantum bits in order to implement a particular algorithm. Part of this problem involves determining optimal numeric parameters for individual quantum operators; this part of the problem is clearly not a selection problem. However, a simpler variant of the problem focuses only on the combinatoric optimisation of sequences of standard quantum operators and this clearly conforms to our notion of a selection problem.
- *Resource allocation problems* like nurse rostering [30], surgery scheduling [31], or TA allocation [32] are also all clearly selection problems as they can be conceptualised as establishing 'allocation' edges between fixed sets of resources and resource users.

The above overview provides some evidence that our notion of 'selection problems' is sufficiently broad to allow many different real-world applications. For the purpose of this paper, we will focus on one running example to help illustrate our definitions and results. We discuss this example next.

2.2. A running example: Class Responsibility Assignment

In the *class–responsibility assignment (CRA) problem* [33], a set of methods and attributes must be allocated to a (to be determined) number of classes to optimise cohesion and coupling metrics of the resulting software design. It is not immediately obvious that this is a selection problem, as it might involve the generation of an unspecified number of new classes (which are nodes in a graph representing allocations of methods / attributes to classes). However, once one realises that any meaningful solution contains at most as many classes as there are features (methods and attributes), it becomes clear that the selection problem is about determining the correct set of edges between features and classes.

Figure 2 shows the metamodel (type graph) for the CRA problem—a class diagram whose instances describe specific sets of classes, attributes, and methods for which to determine an allocation. Some nodes and edges are shown in blue and with dashed rather than solid lines. These are intended to be normal parts of the metamodel, but have been highlighted to indicate that only the corresponding parts of any model can be changed when deciding on an allocation of methods/attributes to classes. That is, we are allowed to create and delete classes, and we are allowed to change the encapsulates relationship between classes and methods/attributes. We consider the black elements (solid lines) as part of the *problem description,* while we think of the blue elements (dashed lines) as part of the *solution description.*

Figure 3 then shows a model that instantiates the meta-model in Figure 2 and describes a concrete problem with 3 methods and 3 attributes that need to be allocated to classes. Figure 4 shows a corresponding instance of the metamodel that represents one possible solution to the CRA problem. Here, we have introduced two classes and made specific assignments of methods and attributes to these classes.

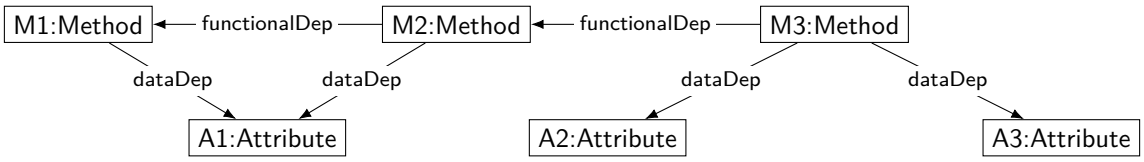


Figure 3: Example instance P of the CRA problem

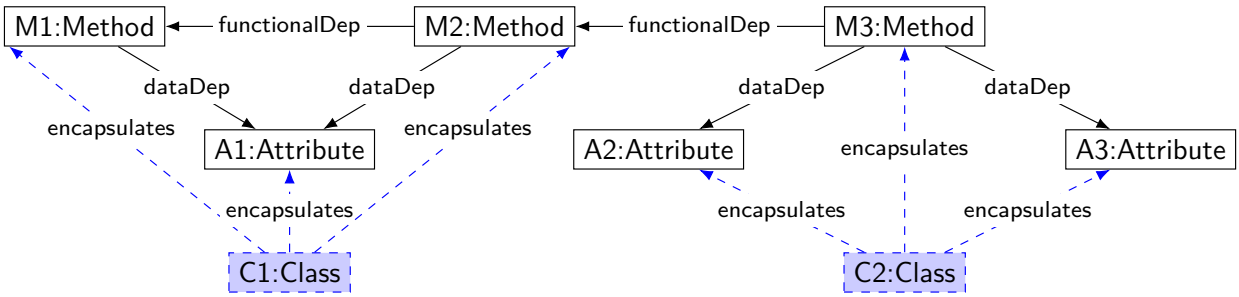


Figure 4: Example of a solution S to the CRA problem instance depicted in Fig. 3

3. Preliminaries

In this section, we introduce our preliminaries. As our general background, we recall *typed graphs* and their transformations via *rules* applied according to the *double-pushout approach* [34] and *nested constraints* and *conditions* [11]. We present *subgraph transformation systems* (as a special case of *subobject transformation systems* [25]) as the kind of transformation system for which we are able to construct our non-blocking consistency-preserving rules. Furthermore, we recall variability-based rules [17] as a mechanism to compactly represent a set of rules. Finally, we discuss and extend *effect-oriented transformations* [28], a recent extension of the double-pushout semantics, as the semantics we use to apply our constructed consistency-preserving rules.

3.1. Graph transformation foundations

Graphs are sets of nodes with edges between them. Typing offers a way to equip the individual elements of a graph with meaning; a *type graph* provides the available types for elements, and morphisms assign the elements of *typed graphs* to those types.

Definition 1 (Graph. Typed graph)**.** A *graph* $G = (V_G, E_G, src_G, tar_G)$ consists of a set of nodes V_G, a set of edges E_G, and source and target functions $src_G, tar_G \colon E_G \to V_G$. A *graph morphism* $f \colon G \to H$ between two graphs is a pair of functions $f_V \colon V_G \to V_H$ and $f_E \colon E_G \to E_H$ that is compatible with the source and target functions, such that

$$src_H \circ f_E = f_V \circ src_G \text{ and } tar_H \circ f_E = f_V \circ tar_G.$$

A *type graph* $TG = (V_{TG}, E_{TG}, src_{TG}, tar_{TG})$ is a fixed graph. A *graph typed over a fixed type graph* TG consists of a graph G and a graph morphism $type_G \colon G \to TG$. A *typed graph morphism* $f \colon G \to H$ between graphs G and H typed over TG is a graph morphism that is compatible with typing, i.e., such that $type_H \circ f = type_G$.

A typed graph morphism is *injective/surjectiv/bijective/an inclusion* if both of its components are. We denote injective morphisms via hooked arrows $f \colon G \hookrightarrow H$ and regularly denote inclusions as $\iota^G_H \colon G \subseteq H$, where the indices denote the involved graphs. ♢

Remark 1. To ease language and notation, we adopt the following conventions and assumptions.

- In this paper, we work with typed graphs and always assume a fixed type graph TG over which all graphs are typed. Except for the type graph, we will just speak of *graphs*, and we do not introduce a special notation for typed graphs. Thus, speaking of a graph G, we tacitly assume a type graph TG and a type morphism $type_G \colon G \to TG$ to exist. Similarly, we also regularly just speak of morphisms (and not of typed ones).
- Via $\emptyset$ we denote the *initial* or *empty graph*, also when it is considered to be typed over the given type graph TG (via the empty function).
- We furthermore assume that all our graphs are finite. By $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$ we denote the category of all finite graphs typed over the graph TG. ♢

Example 1. The graph TG from Fig. 2 can serve as a type graph, modelling the domain of the CRA problem. The nodes provide the types Class, Attribute and Method. Methods can have dependencies on Methods (functionalDep) and on Attributes (dataDep). Classes can encapsulate Methods and Attributes.

The further graphs from Sect. 2.2, namely graph P (Fig. 3) and graph S (Fig. 4), are typed over that type graph TG (and the typing morphisms are indicated by the annotations of nodes and edges). Recall that the blue, dashed elements are ordinary graph elements and that the visual difference just highlights that these elements belong to the *solution* and not the *definition* of a problem instance of the CRA problem. ♢

3.2. Subgraph transformation

Subobject transformation systems have been developed as a tool to study different notions of causality between rule applications in (double-pushout) rewriting systems [25, 26]. Intuitively, one studies transformations of subobjects of a given object from an ($\mathcal{M}$-)adhesive category. In the following, we instantiate the definitions for the case where the object whose subobjects are considered is a (typed) graph. For simplicity of presentation, we work with subgraphs instead of injective morphisms.

Definition 2 (Subgraph)**.** Given a typed graph G, a *subgraph* of it is a typed graph H, such that $V_H \subseteq V_G$, $E_H \subseteq E_G$ and src_H, tar_H, and $type_H$ are the restrictions of the according functions of G (in particular, with every edge $e \in E_H$ its source and target nodes in V_G belong to V_H). Via $H \subseteq G$ we denote the subgraph relationship between graphs.

Given a typed graph T, the *category of subgraphs of* T, $\mathbf{Sub}(T)$, has subgraphs $G \subseteq T$ as objects and inclusions $\iota^G_H \colon G \subseteq H$ between such subgraphs as morphisms. We call T the *container graph* of that category. ♢

Remark 2. What we call the container graph would be called *type graph* in the terminology of [25]. We deviate from that terminology to distinguish the container graph T from the type graph TG over which T and all graphs from $\mathbf{Sub}(T)$ are typed. ♢

Example 2. Given the type graph TG from Fig. 2 (see Sect. 2.2 and Example 1), also the graph T depicted in Fig. 5 is typed over it.

Considering the CRA problem instance defined by the graph P shown in Fig. 3, $\mathbf{Sub}(T)$ defines a category that contains all interesting solutions to that problem instance (disregarding solutions with more than six Classes of which several will be necessarily empty). The graph S depicted in Fig. 4 is one of the elements of $\mathbf{Sub}(T)$. Note that $\mathbf{Sub}(T)$ is actually too large to be considered the search space for finding solutions for the problem instance defined by the graph P from Fig. 3: we are only interested in graphs also containing that instance P; otherwise the problem definition would be changed (containing a different set of Methods and Attributes and/or other dependencies). Later on, we will address that problem by only considering graphs $G \in \mathbf{Sub}(T)$ for which $P \subseteq G$, where $P \subseteq T$ is considered to define a problem instance. ♢

In contrast to standard double pushout rewriting [34], where a rewrite might create "new" elements, a *rule* (or *production*) in a subobject transformation system itself consists of subobjects of the given container. This has the effect that subobject rewriting can rather be understood as *displaying* and *hiding* elements from the container object.

Transformations in subobject transformation systems are based on lattice-theoretical concepts.

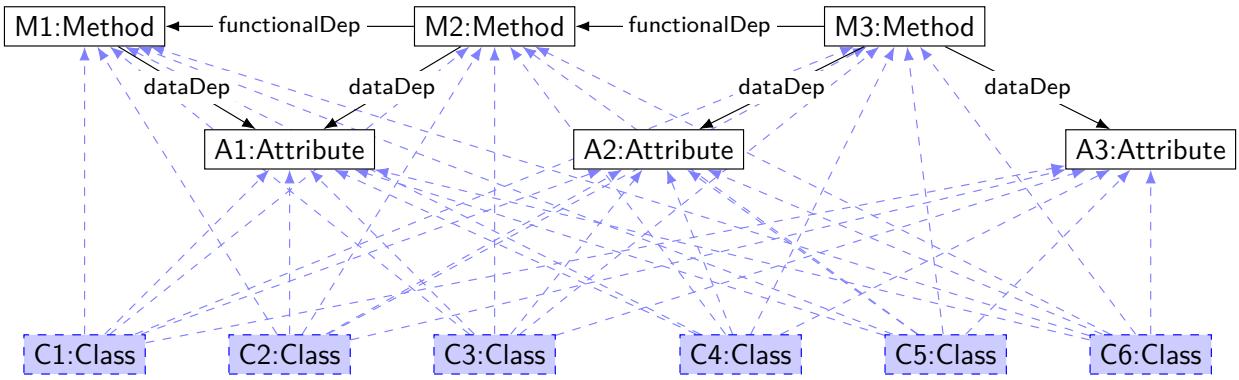


Figure 5: A container graph T defining a category $\mathbf{Sub}(T)$ that includes all interesting solutions to the CRA problem instance defined in Fig. 3. Recall that the blue, dashed elements are ordinary elements of the graph and that the visual difference just highlights that these elements belong to the solution rather than the definition of a problem instance of the CRA problem. In this graph, we additionally omit the encapsulates annotation and draw these edges paler to not further clutter the figure.

Remark 3. In any category of subobjects, if there exists a morphism between subobjects A and B, this morphism is unique (by left-cancelability of monomorphisms).

Moreover, given an adhesive category $\mathcal{C}$ (such as the category of finite graphs typed over a given type graph TG), for every object T of $\mathcal{C}$, $\mathbf{Sub}(T)$ constitutes a distributive lattice [35, Theorem 5.1 and Corollary 5.2]. Given two subobjects $A, B \subseteq T$, their *meet* $A \cap B$ in $\mathbf{Sub}(T)$ arises from pulling back $A \to T \leftarrow B$ in $\mathcal{C}$ and their *join* $A \cup B$ in $\mathbf{Sub}(T)$ from pushing out $A \leftarrow A \cap B \to B$ in $\mathcal{C}$. (Categorically speaking, the meet $A \cap B \subseteq T$ is the *product* of $A \subseteq T$ and $B \subseteq T$ in $\mathbf{Sub}(T)$, and the join $A \cup B \subseteq T$ their *coproduct* [25].) In our setting, where we work with subgraphs instead of equivalence classes of injections, this means that the meet $A \cap B$ is the intersection of A and B as graphs while the join $A \cup B$ is their union. ♢

Definition 3 (Rule. Direct derivation)**.** Given a container graph T (typed over a type graph TG), a *rule* (or *production*) p is a span in $\mathbf{Sub}(T)$, i.e., consists of subgraphs $L \supseteq K \subseteq R$ of T; the subgraphs are called the *left-hand side* (LHS), the *interface*, and the *right-hand side* of the rule, respectively. A rule $p = L \supseteq K \subseteq R$ is called *pure* if $L \cap R = K$.

Given a graph G of $\mathbf{Sub}(T)$ and a rule $p = L \supseteq K \subseteq R$, there exists a (*direct*) *derivation* from G to H via p, denoted as $G \Longrightarrow_p H$, if there exists a graph D in $\mathbf{Sub}(T)$ such that

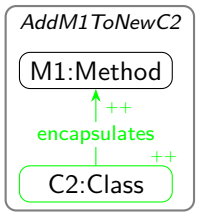
(a) $L \cup D = G$; (c) $D \cup R = H$;
(b) $L \cap D = K$; (d) $D \cap R = K$.

In case D exists, it is called the *context* of the direct derivation $G \Longrightarrow_p H$. ♢

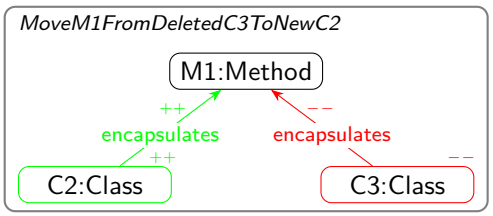
Intuitively, in a derivation from G to H via a rule $p = L \supseteq K \subseteq R$, first those elements from T that are present in L but not in K are removed from G, resulting in D. If p would require to remove elements from G that G does not contain (i.e., if $L \setminus K$ contains elements that G does not contain)[2], no derivation exists as property (a) from the definition cannot be fulfilled. Then, those elements from T that are present in R but not in K are added to D, resulting in H. If p would require to add elements to D that are already present (i.e., if $R \setminus K$ contains elements that also D—and, consequently, G—contains), no derivation exists as property (d) would be violated. From this description, it also becomes clear that non-pure rules are rules that delete an element that they immediately re-create. We therefore mostly focus on pure rules.

Fact 1. *Let a container graph T be given. Then the following important properties for derivations hold.*

[2]Here, and in the following, "\" denotes the componentwise set-theoretic differences on the sets of nodes and edges.



(a) Rule assigning Method M1 to newly added Class C2



(b) Rule removing Class C3 and moving M1 from it to newly created Class C2

Figure 6: Two example rules in integrated visual syntax

- *By [25, Proposition 6] and [34, Fact 3.11] it holds that a rule $p = L \supseteq K \subseteq R$ is* applicable *to a subgraph G of T if and only if:*
  - $L \subseteq G$,
  - $G \cap R \subseteq L$, *and*
  - *for every node n in $L \setminus K$ all edges that are adjacent to n in G are also part of $L \setminus K$.*
- *By condition (c), i.e., by requiring $D \cup R = H$, the graph H resulting from a direct derivation is itself an object in* $\mathbf{Sub}(T)$. ♢

Example 3. Figure 6 shows two (pure) rules in an integrated, visual syntax that is often used for displaying graph transformation rules (e.g., in tools like Groove [36] or Henshin [37]). The three graphs L, K and R are displayed as a single graph, with colour coding (and annotations) indicating the different roles of elements. The graph K consists of the black elements (without further annotation), the graph L consists of K and the red elements (additionally annotated with $--$), and the graph R consists of K and the green elements (additionally annotated with $++$). Thus, the red elements, i.e., elements from $L \setminus K$, are to be deleted upon application of a rule, and green elements, i.e., elements from $R \setminus K$, are to be created.

A crucial difference to ordinary graph transformation is that we do not need to find a match but that such a match (if it exists) is uniquely determined by identity of elements. Thus, the first rule *AddM1ToNewC2* does not specify to create some Class and assign some Method to it but concretely specifies to *add* the Class C2 from graph T (see Fig. 5) to some graph $G \subseteq T$. It is applicable to every graph $G \subseteq T$ that contains Method M1 but does not yet contain Class C2. Thus, it is applicable to graph P from Fig. 3 but not to graphs S and T from Figs. 4 and 5 ($R \cap S \not\subseteq K$ resp. $R \cap T \not\subseteq K$, where R and K denote the RHS and interface of the rule). Similarly, the second rule, *MoveM1FromDeletedC3ToNewC2*, specifies to remove Class C3 and the edge encapsulating M1 and create Class C2 and an according encapsulates edge instead. This rule is not applicable to any of the example instances considered so far (because in none of these, Class C3 encapsulates M1). ♢

3.3. Nested conditions and constraints

Next, we define *nested conditions* and *constraints* [11, 38, 39]. Nested conditions provide a logic to express properties of morphisms in a category; constraints express properties of objects. For (typed) graphs, nested conditions are expressively equivalent to standard first-order logic on graphs [38, 11]. Here, every level of nesting increases the expressivity [38].

In this work, given a type graph TG and a container graph T that is typed over TG, we can consider two different categories: The category of finite graphs typed over TG, $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$, and the category $\mathbf{Sub}(T)$ of subgraphs of T (with inclusions as morphisms). Since every graph from $\mathbf{Sub}(T)$ is also a graph typed over TG (by composition of the inclusion with the typing of T), we obtain two variants of defining formulas for it.

We define conditions and constraints category-theoretically and afterwards illustrate the definitions for $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$ and $\mathbf{Sub}(T)$.

Definition 4 ((Nested) conditions and constraints)**.** In a category $\mathcal{C}$, given an object C_0, a *(nested) condition* over C_0 is defined recursively as follows:

- true is a condition over C_0.
- If $a_1 \colon C_0 \hookrightarrow C_1$ is a monomorphism and d is a condition over C_1, $\exists\,(a_1 \colon C_0 \hookrightarrow C_1, d)$ is a condition over C_0.
- Moreover, Boolean combinations of conditions over C_0 are conditions over C_0.

In a category with a strict initial object $\emptyset$, a *(nested) constraint* is a condition over $\emptyset$.

We recursively define the *nesting level* of a condition c as

$$\begin{aligned} \mathrm{nl}(\mathsf{true}) &= 0; \\ \mathrm{nl}(\neg d) &= \mathrm{nl}(d); \\ \mathrm{nl}(d_1 \odot d_2) &= \max\{\mathrm{nl}(d_1), \mathrm{nl}(d_2)\}, \text{ where } \odot \in \{\vee, \wedge\}; \\ \mathrm{nl}(\exists(a_1 \colon C_0 \hookrightarrow C_1, d)) &= \mathrm{nl}(d) + 1. \end{aligned}$$

Satisfaction of a nested condition c over C_0 for a morphism $g \colon C_0 \to G$, denoted as $g \models c$, is defined as follows:

- Every morphism satisfies true.
- The morphism g satisfies a condition of the form $c = \exists\,(a_1 \colon C_0 \hookrightarrow C_1, d)$ if there exists a monomorphism $q \colon C_1 \hookrightarrow G$ such that $g = q \circ a_1$ and $q \models d$.
- For Boolean operators, satisfaction is defined as usual.

An object G satisfies a constraint c, denoted as $G \models c$, if the unique monomorphism $\iota_G \colon \emptyset \hookrightarrow G$ does so.

An object G satisfies a set of constraints $\mathcal{C}$, denoted as $G \models \mathcal{C}$, if G satisfies each individual constraint $c \in \mathcal{C}$. ♢

Remark 4. In the notation of conditions, we drop the domains of the involved morphisms and occurrences of true whenever they can be unambiguously inferred. Moreover, we introduce $\forall\,(C_1, d)$ as an abbreviation for the condition $\neg\exists\,(C_1, \neg d)$. For example, we write $\forall\,(C_1, \exists\, C_2)$ instead of $\neg\exists\,(a_1 \colon \emptyset \hookrightarrow C_1, \neg\exists\,(a_2 \colon C_1 \hookrightarrow C_2, \mathsf{true}))$. At other places, especially where it is convenient for compact presentation of computations, we will not display the graphs of a condition but merely the identifiers of the involved morphisms, writing, e.g., $\forall\,(a_1, \exists\, a_2)$.

To be able to notationally distinguish between satisfaction of constraints and conditions in the category $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$ from the one in $\mathbf{Sub}(T)$ (where T is typed over TG), we use the symbol $\models_{\mathcal{I}}$ to denote satisfaction in $\mathbf{Sub}(T)$ and $\equiv_{\mathcal{I}}$ to denote equivalence in $\mathbf{Sub}(T)$. Moreover, we denote the morphisms in conditions over graphs from $\mathbf{Sub}(T)$ as $b_i \colon B_{i-1} \subseteq B_i$ to visually differentiate these from (i) morphisms $a_i \colon C_{i-1} \hookrightarrow C_i$ in conditions over a graph from $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$ and (ii) from inclusions $\iota^{C_i}_G : C_i \subseteq G \in \mathbf{Sub}(T)$ we use to check the satisfaction of such conditions. ♢

Example 4. The definition of the CRA problem comes with additional constraints (compare Sect. 2.2): As a minimum, one requires that every Attribute and every Method is encapsulated by exactly one Class. In this paper, for illustrative purposes, we additionally add the constraint that every Attribute is private, which can be expressed by requiring that every Method that has a data dependency to an Attribute is encapsulated by the same Class encapsulating the Attribute.

Figure 7 depicts in $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$ the constraint expressing the *lower bound* on encapsulation of Methods, namely stating that every Method shall be encapsulated by (at least) one Class (the names of the elements in the graphs indicate the homomorphism from C_1 to C_2); in the following, we refer by c_{lb} to that constraint. In $\mathbf{Sub}(T)$, in contrast, we can only express properties of concrete elements (as the constraints in $\mathbf{Sub}(T)$ are defined in terms of subgraphs of T); thus, the constraint in Fig. 8 expresses that the concrete Method M1

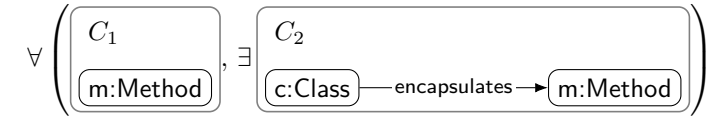


Figure 7: Constraint c_{lb} in $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ requiring every Method to be assigned to at least one Class

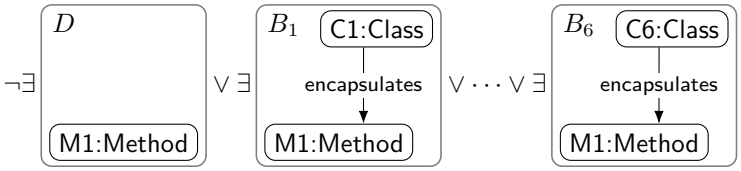


Figure 8: One conjunct of constraint $c_{\text{lb}}^{\mathbf{Sub}(T)}$ in $\mathbf{Sub}(T)$, requiring Method M1 to be assigned to at least one of Classes C1, ..., C6 from T (in case it exists)

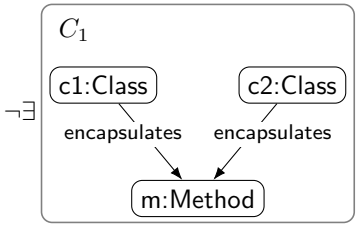


Figure 9: Constraint c_{ub} in $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ forbidding any Method to be assigned to more than one Class

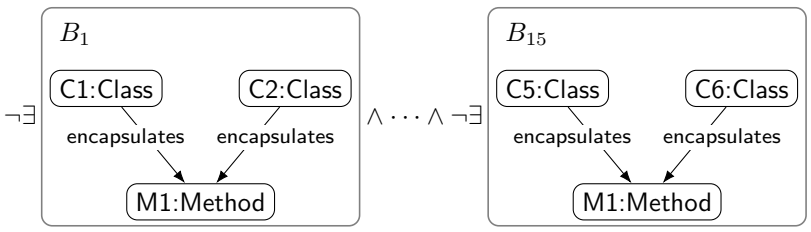


Figure 10: Conjunct of constraint $c_{\text{lb}}^{\mathbf{Sub}(T)}$ in $\mathbf{Sub}(T)$, forbidding Method M1 to be assigned to more than one of the Classes C1, ..., C6 from T

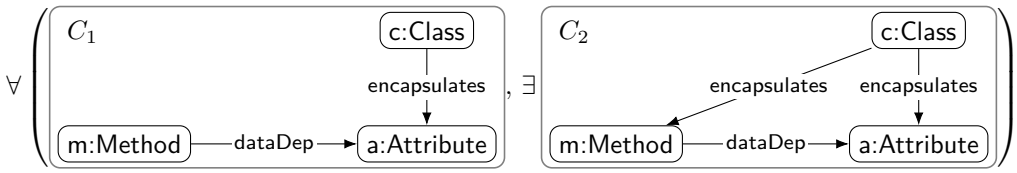


Figure 11: Constraint c_{priv} in $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ requiring the sources of data dependencies (Methods) to be assigned to the same Classes as their targets (Attributes)

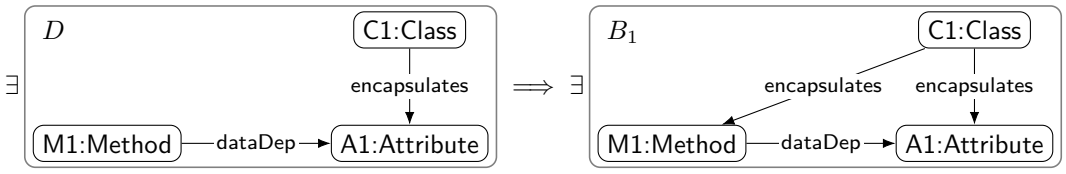


Figure 12: Conjunct of constraint $c_{\text{priv}}^{\mathbf{Sub}(T)}$ in $\mathbf{Sub}(T)$, requiring Method M1 to be assigned to Class C1 from T in case Attribute A1 (the target of the data dependency) is

is encapsulated by *a* Class—either C1, C2, ... or C6. The same constraint has to be added for Methods M2 and M3. By $c_{\text{lb}}^{\mathbf{Sub}(T)}$, we refer to the conjunction of these constraints.

Continuing, the constraint from Fig. 9 expresses in $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ that no Method is encapsulated by more than one Class, that is, it states an *upper bound* on these encapsulations; in the following, we refer by c_{ub} to that constraint. Again, in $\mathbf{Sub}(T)$, we have to express this for every Method individually and exclude any possible pair of Classes; the resulting set of constraints for Method M1 is shown in Fig. 10. By $c_{\text{ub}}^{\mathbf{Sub}(T)}$, we refer to the conjunction of these constraints.

Finally, Fig. 11 shows a constraint in $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ that—together with the requirement that every Attribute is encapsulated—expresses that Methods are encapsulated together with the Attributes to which they have data dependencies, i.e., the constraint that Attributes are *private*. We refer via c_{priv} to this constraint in the following. Figure 12 shows the same constraint for the data dependency between Method M1 and Attribute A1 for Class C1; we have to add such a constraint for every data dependency in T for every Class. By $c_{\text{priv}}^{\mathbf{Sub}(T)}$ we refer to the conjunction of these constraints. ♢

Assumption on presentation of constraints in $\mathbf{Sub}(T)$. In a category of subgraphs $\mathbf{Sub}(T)$ (where T is finite), we can transform every nested condition or constraint into an equivalent and non-nested condition or constraint. In particular, we can assume every constraint to be given in one of the normal forms well-known from propositional logics, with *literals* being nested constraints of nesting level ≤ 1. We prove this in [27]. Moreover, in that note we also show how to systematically translate the far more succinct constraints from $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ to semantically equivalent constraints in $\mathbf{Sub}(T)$.

Therefore, without loss of expressivity, we make the following assumption for the rest of the paper.

Assumption 1. We assume that, given a type graph TG, a finite graph T typed over it, and a set $\mathcal{C}$ of conditions in $\mathbf{Sub}(T)$, all the conditions from $\mathcal{C}$ are individual clauses of literals over $\mathbf{Sub}(T)$ (i.e., we assume these to be given in conjunctive normal form (CNF) but split up the different clauses of a condition into different conditions).

- the *trivial* cases: $c = \mathsf{true}$ or $c = \mathsf{false}$;
- the *purely positive* case: $c = \exists\,(B_1, \mathsf{true}) \vee \cdots \vee \exists\,(B_t, \mathsf{true})$;
- the *purely negative* case: $\neg\exists\,(D, \mathsf{true})$;

- the *mixed* or *implication* case:

$$\begin{aligned} &\neg\exists\,(D, \mathsf{true}) \vee \exists\,(B_1, \mathsf{true}) \vee \cdots \vee \exists\,(B_t, \mathsf{true}) \\ \equiv &\exists\,(D, \mathsf{true}) \implies \exists\,(B_1, \mathsf{true}) \vee \cdots \vee \exists\,(B_t, \mathsf{true}). \end{aligned}$$ ♢

3.4. Variability-based rules

A *variability-based rule* (VB rule) [17] is a compact representation of a family of related, individual rules. We present VB rules since in our approach we will first derive a VB rule from an activation diagram. This VB rule will encode different ways of augmenting an originally given rule in a consistency-preserving fashion. Individual consistency-preserving rules can then be derived from the VB rules as instances. Instead of introducing VB rules in full generality [17], we present a simplified definition inspired by [10] that suffices for our purposes.

We start by concisely introducing *feature models* and their *configurations* as the definition of a VB rule makes use of these. In our presentation of feature models and VB rules we closely follow [10]. Feature models [40] allow specifying the variant space of a variant-rich system in a hierarchical form, graphically commonly represented as a feature diagram. A feature model comprises a set of features and relations amongst those, such as parent-to-child, alternative groups, and exclusion between features. Each possible variant of the system arises from a subset of its features, called a *configuration* of the feature model.

Definition 5 (Feature model. Configuration)**.** A *feature model fm* consists of a set of literals $\mathcal{F}$, called *features*, a strict partial order $<$ defining a parent–child relationship over the features, and a propositional formula Q over elements from $\mathcal{F}$, called *cross-tree constraints* and expressing relationships between features. The partial order has a minimal element, the *root element*. A feature g is considered a *child* of feature f if $f < g$ and there exists no feature $h \in \mathcal{F}$ such that $f < h < g$. Child features can be *mandatory* or *optional*. Moreover, a feature can be a *group feature*, namely an *or* or a *xor* group; in that case, it must have at least two children.

Given a feature model *fm*, a *configuration* F is a subset of $\mathcal{F}$; the features of F are called *active*. A configuration F is *valid* if it satisfies the following properties:

1. CRoot: If f is the root feature of *fm*, $f \in F$.
2. CPar: If g is the parent of f and $f \in F$, then $g \in F$.
3. CMand: If g is a mandatory child of f and $f \in F$, then $g \in F$.
4. COr: If f is an or group and $f \in F$, then at least one of f's children is also in F.
5. CXor: If f is a xor group and $f \in F$, then exactly one of f's children is also in F.
6. CRel: The features selected in F satisfy the relations required by the formula Q, i.e., if χ_F is the characteristic function of F, then $\chi_F \models Q$. ♢

Example 5. The feature diagram in Fig. 13 represents the *MobileMedia* SPL, a standard example for variability-rich systems [41]. *MobileMedia* is the root feature and has three mandatory children, of which *MediaSelection* is an "or" group, *Screensize* is an "xor" group, and *MediaManagement* has three optional children. The implication cross-tree constraint at the bottom represents a *requires* relationship exposing the need of selecting *CopyMedia* when *SMSTransfer* is selected. An example for a valid configuration is c_1 = *{MobileMedia, MediaSelection, Music, MediaManagement, ScreenSize, Screen3}*. Adding *Screen1* to c_1 leads to an invalid configuration, because having two active children of the same "xor" group violates an underlying validity constraint of feature models. In the following, we use the graphical notation from Fig. 13 to display feature diagrams. ♢

We now introduce VB rules as we use them in Section 5.

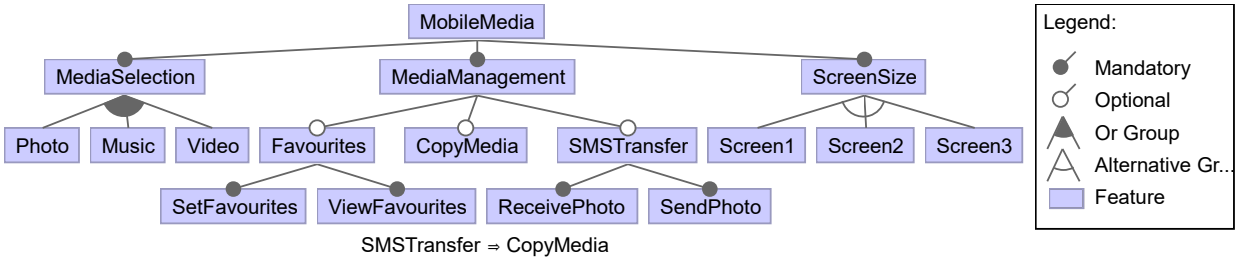


Figure 13: Example feature diagram of Mobile Media [41]

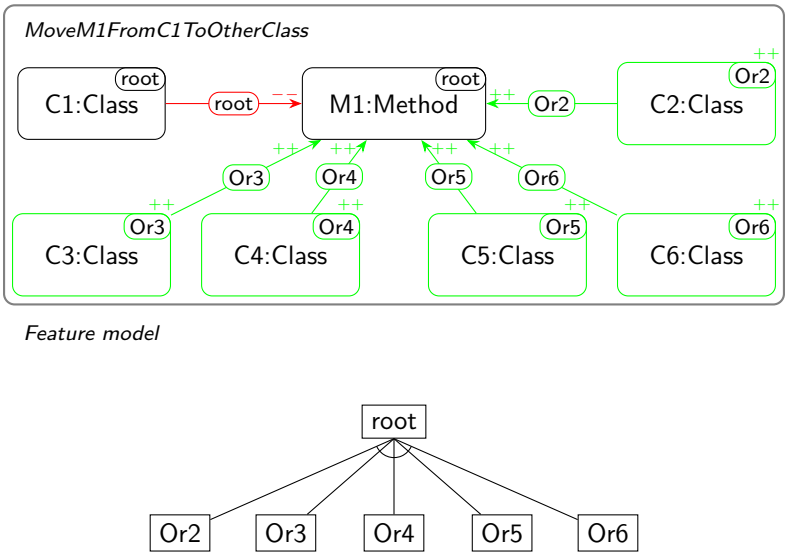


Figure 14: Example VB rule capturing the different possibilities to move Method M1 from C1 to another Class

Definition 6 (Variability-based rule)**.** Given a type graph TG and a graph T typed over it, a *variability-based rule* (VB rule) $pv = (p, fm, b_L, b_K, b_R)$ in $\mathbf{Sub}(T)$ consists of a rule $p = L \supseteq K \subseteq R$ in $\mathbf{Sub}(T)$, a feature model fm, and functions b_L, b_K, b_R that map the elements of L, K and R, respectively, each to a propositional formula over the features $\mathcal{F}$ of fm, called *presence condition*.

Given a valid configuration F of fm, let X_F, where $X \in \{L, K, R\}$, denote the respective sets of elements whose presence conditions evaluate to true under F (i.e., for each $x \in X$ it holds that $x \in X_F \iff \chi_F \models b_X(x)$, where χ_F is the characteristic function of $F \subseteq \mathcal{F}$). The configuration F *induces* a rule $p_F = L_F \supseteq K_F \subseteq R_F$, also referred to as a *variant* of pv, if L_F, K_F and R_F are graphs and $K_F \subseteq L_F$ and $K_F \subseteq R_F$. ♢

We will later construct presence conditions in such a way that every valid configuration of our VB rules induces a rule.

Example 6 (VB rule)**.** Figure 14 depicts a VB rule in an integrated fashion[3]. Its interface consists of the nodes C1 and M1; its LHS additionally contains the edge between these two nodes. Beyond the interface, its RHS contains Classes C2, ..., C6 and their outgoing edges. All edges are of type encapsulates, which is omitted from the figure since it is the only available type. The rule's feature model is depicted at the bottom, consisting of a root feature and five children Or2, ..., Or6, grouped in a Xor-group. The presence conditions

[3]Note that our integrated illustration does not work for VB rules in general because b_L, b_R, b_K can differ on elements and the underlying rule of a VB rule does not need to be pure.

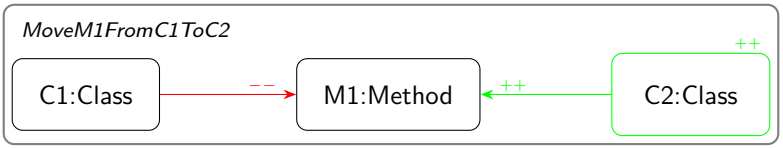


Figure 15: Instantiation of *MoveM1FromC1ToC2* of *MoveM1FromC1ToOtherClass* induced by configuration {root, Or2}

of the rule elements are indicated by annotations; b_L and b_K are constant, mapping every element contained in the LHS resp. interface to root. The function b_R also maps C1 and M1 to root and every further Class, together with its incident edge, to OrX, where X coincides with the number of the Class.

Because the children of root constitute a Xor-group (here indicated by standard concrete syntax for feature models), the rule's feature model has five valid configurations, namely {root, OrX} for X ∈ {2, . . . , 6}. Thus, the VB rule encodes five rules, namely the five rules moving M1 from C1 to one of the other Classes available in T; Fig. 15 depicts the instance moving M1 to C2. Note that those are the potential reactions to maintain validity when removing M1 from C1 (because every Method should be assigned to a Class). We will systematically construct VB rules that capture all ways to maintain consistency with regard to a given set of constraints in Sect. 5. ♢

3.5. Effect-oriented transformations

In [28], we introduced *effect-oriented graph transformation* as a means to deal with the problem that application context for rules might be unknown at their design time. In particular, when designing rules that are intended to preserve the validity of given constraints, it is regularly the case that it is known that a change to an element x (creation or deletion) necessitates a change also to element y (creation or deletion)—*in case y does(n't) exist in the first place*. However, whether the respective element y exists (or not) at runtime is unknown at design time. In effect-oriented graph transformations, we equip *effect-oriented rules* with a kind of skip semantics that allows defining selected actions (creations and deletions) as *potential* ones: A potential creation is only performed if the respective element does not already exist and a potential deletion is only performed if the respective element exists. In contrast to traditional graph transformation, existence resp. non-existence of the elements concerning potential creations and deletions does not stop the rule from being applied; unnecessary actions are just skipped, and remaining actions are performed upon application of the rule according to effect-oriented semantics. The whole approach drastically reduces the number of rules one needs to specify to cover all possible scenarios for which one wants to implement a certain action. The intended semantics of an effect-oriented transformation is the following: Formally, rendering actions potential is realised by defining an effect-oriented rule as a pair of rules with one rule being a subrule of the other. The subrule, called *base rule*, specifies the mandatory actions; the additional actions defined in the super rule, called *maximal rule*, are the potential ones. The extension relationship between the two rules is formally captured by the base rule being a *subrule* of the maximal rule (the definition of which simplifies the concept of *kernel* and *multi-rules* [42]).

In the following, we formally introduce *effect-oriented graph transformation* in the context of a category $\mathbf{Sub}(T)$, where T is a typed graph. Compared to our approach to ordinary graph transformation in [28], we do not have to consider a matching process since in $\mathbf{Sub}(T)$ every part of a rule is a subgraph of T; this considerably simplifies the definitions. However, for our application later in this paper (constructing consistency-preserving rules), we need to extend the original approach and also allow skip-operations for elements from the interface of the rule. This leads to a potential problem for the semantics if defined via the subrule relationship: If a node in the interface is allowed to be skipped for which the rule also declares a potential creation of an adjacent edge, the semantics becomes unclear for applications to graphs in which the to-be-skipped interface node does not exist: Should this node then also be created (even though it is nowhere defined to be created in the rule) or should the creation of the adjacent edge be omitted (even though it does not yet exist, violating the intended semantics of effect-oriented transformations)? To avoid this, we extend the definition of effect-oriented rules by what we call *context extensions* that serve to provide the skip-actions for interface elements.

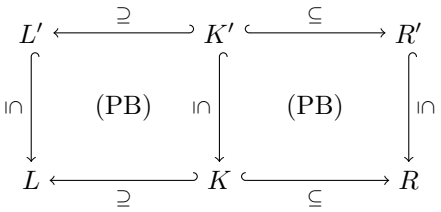


Figure 16: Subrule p' of a rule p

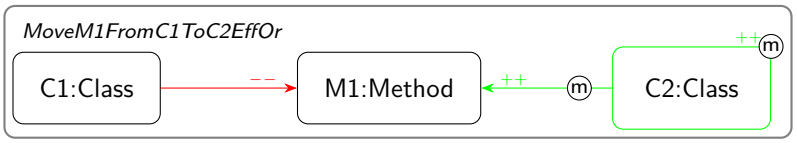


Figure 17: Effect-oriented rule moving Method M1 from Class C1 to Class C2 (also in case C2 already exists)

Definition 7 (Subrule. Effect-oriented rule)**.** Given a rule $p = (L \supseteq K \subseteq R)$ in $\mathbf{Sub}(T)$, a *subrule* of p is a rule $p' = (L' \supseteq K' \subseteq R')$ such that $X' \subseteq X$ for $X \in \{L, K, R\}$ and $K' = L' \cap K$ and $K' = R' \cap K$ in $\mathbf{Sub}(T)$, i.e., both squares in Fig. 16 are pullbacks in $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ (by $L \subseteq T$). We denote such a subrule relationship between rules as $p' \subseteq p$.

An *effect-oriented rule* p_e is a rule together with a subrule and a *context extension* for the rule, i.e., a triple $(p_b = (L_b \supseteq K_b \subseteq R_b), p_m = (L_m \supseteq K_b \subseteq R_m), k_{ext} \colon K_b \subseteq K_m)$ such that (i) p_b is pure and $L_m \cap R_m \subseteq K_m$; (ii) p_b and p_m have an equal interface K_b and $p_b \subseteq p_m$; (iii) k_{ext} factors through $K_b \subseteq L_m$ inducing a pullback, i.e., $K_m \subseteq L_m$ and $L_b \cap K_m = K_b$. The rules p_b and p_m are called the *base rule* and the *maximal rule*, respectively. The *potential deletions* of the maximal rule are the elements of $(L_m \setminus K_m) \setminus L_b$; similarly, its *potential creations* are the elements of $(R_m \setminus K_b) \setminus R_b$. ♢

Example 7. The rule *MoveM1FromC1ToC2* is one of the instantiations of *MoveM1FromC1ToOtherClass* (see Figs. 14 and 15 in Example 6). Figures 17 and 18 show two different ways to interpret that rule as an effect-oriented rule. First, in Fig. 17, we interpret the instantiated rule *MoveM1FromC1ToC2* as the maximal rule of the effect-oriented rule. The base rule merely consists of the deletion of the encapsulates-edge between C1 and M1, the context extension is the identity morphism in this case. That an element belongs to the maximal rule is indicated by the index m in the figure.

In Fig. 18, in contrast, the base rule is the constant rule with each graph merely consisting of M1 (and performing no action). All actions are part of the maximal rule (again indicated by the index m on elements) and Class C1 becomes part of K_m, i.e., of the target graph of the context extension (and, consequently, also part of the LHS L_m of the maximal rule). ♢

An *effect-oriented transformation*, i.e., the application of an effect-oriented rule, first consists of the application of the base rule. Subsequently, all applicable potential actions (deletions and creations) are performed. Technically, we separate the computation of applicable potential deletions and applicable potential creations (combining them into a single rule in a second step). This allows us to consider the context extension only for computing applicable potential deletions, leading to the desired effect that we can use the context

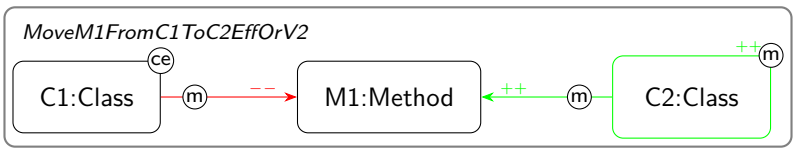


Figure 18: Effect-oriented rule moving Method M1 from Class C1 to Class C2 (also in case C2 already exists or C1 does not)

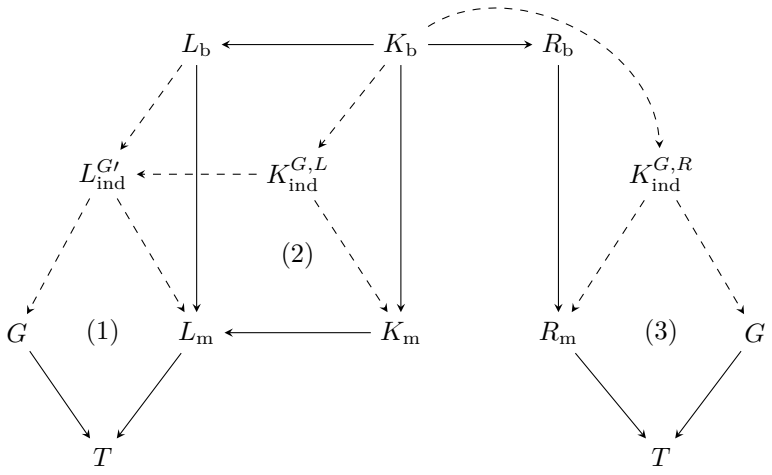


Figure 19: First part of construction of the induced rule

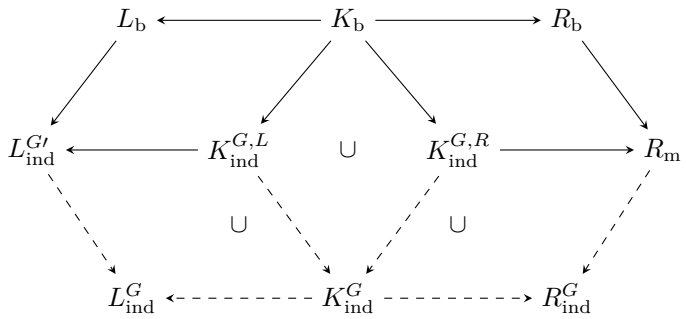


Figure 20: Second part of construction of the induced rule

extension to define adjacent nodes for potentially to be deleted edges, without having to care about what this means for potential creations. The following definition captures this formally.

Definition 8 (Effect-oriented transformation)**.** Given a typed graph T, an effect-oriented rule $p_\mathrm{e} = (p_\mathrm{b}, p_\mathrm{m}, k_\mathrm{ext})$ with $p_\mathrm{b} = (L_\mathrm{b} \supseteq K_\mathrm{b} \subseteq R_\mathrm{b})$, $p_\mathrm{m} = (L_\mathrm{m} \supseteq K_\mathrm{b} \subseteq R_\mathrm{m})$, and $k_\mathrm{ext}\colon K_\mathrm{b} \subseteq K_\mathrm{m}$ in $\mathbf{Sub}(T)$ and a graph G in $\mathbf{Sub}(T)$ such that $L_\mathrm{b} \subseteq G$ and $R_\mathrm{b} \cap G \subseteq K_\mathrm{b}$, the rule *induced by* p_e *for* G, denoted as $p^G_\mathrm{ind} = (L^G_\mathrm{ind} \supseteq K^G_\mathrm{ind} \subseteq R^G_\mathrm{ind})$, is computed in several steps, which are overviewed in Figs. 19 and 20 (where dashed arrows indicate morphisms that are computed during the construction):

(i) Determining deletions to be performed: We set $L^{G\prime}_\mathrm{ind} \coloneqq G \cap L_\mathrm{m}$ and $K^{G,L}_\mathrm{ind} \coloneqq L^{G\prime}_\mathrm{ind} \cap K_\mathrm{m}$ in $\mathbf{Sub}(T)$, i.e., we compute the squares (1) and (2) in Fig. 19 as pullbacks in $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$.

(ii) Determining creations to be performed: We set $K^{G,R}_\mathrm{ind} \coloneqq R_\mathrm{m} \cap G$ in $\mathbf{Sub}(T)$, i.e., we compute square (3) in Fig. 19 as pullback in $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$.

(iii) Combining deletions and creations into rule: We set $K^G_\mathrm{ind} \coloneqq K^{G,L}_\mathrm{ind} \cup K^{G,R}_\mathrm{ind}$ and then $L^G_\mathrm{ind} \coloneqq L^{G\prime}_\mathrm{ind} \cup K^G_\mathrm{ind}$ and $R^G_\mathrm{ind} \coloneqq R_\mathrm{m} \cup K^G_\mathrm{ind}$; this is illustrated in (the bottom row of) Fig. 20.

Given an effect-oriented rule p_e and a graph G, an *effect-oriented transformation* $t\colon G \Longrightarrow H$ via p_e is the derivation $G \Longrightarrow_{p^G_\mathrm{ind}} H$ via the induced rule p^G_ind. ♢

Remark 5. Note that, by construction, $K^G_\mathrm{ind} \subseteq L^G_\mathrm{ind}, R^G_\mathrm{ind}$. We require $L_\mathrm{b} \subseteq G$ to ensure that the deletions prescribed by the base rule p_b are performed when applying p^G_ind. Analogously, requiring $R_\mathrm{b} \cap G \subseteq K_\mathrm{b}$ ensures this for the creations. ♢

Example 8. Consider the rule *MoveM1FromC1ToC2* from Example 6 (see Fig. 15) and its two effect-oriented adaptations we introduced in Example 7 (Figs. 17 and 18). By definition, the rule *MoveM1FromC1ToC2* is applicable to graphs in $\mathbf{Sub}(T)$ where M1 is actually encapsulated by C1 and where C2 does not yet exist. Hence, it is not applicable to any of the graphs from $\mathbf{Sub}(T)$ we have considered so far (which either already contain C2 or do not contain C1).

In contrast, *MoveM1FromC1ToC2EffOr* (Fig. 17) is applicable to any graph in $\mathbf{Sub}(T)$ in which C1 encapsulates M1. The creation of Class C2 and its outgoing encapsulates-edge to M1 are rendered potential. In particular, *MoveM1FromC1ToC2EffOr* is applicable to the graph S (Fig. 4), deleting the encapsulates-edge between C1 and M1, creating a new such edge between C2 and M1 and skipping the creation of C2, which already exists in S. Similarly, *MoveM1FromC1ToC2EffOr* is applicable to T, with the only effect of removing the edge from C1 to M1. Note how the rule captures different situations and ways to move M1 from C1 to C2 in a single rule.

However, *MoveM1FromC1ToC2EffOr* is not applicable to graph P (Fig. 3), because its base rule requires the deletion of an encapsulates-edge between M1 and C1, which is not present in P. The rule *MoveM1FromC1ToC2EffOrV2* (Fig. 18), rendering this action also potential, is also applicable to P, omitting the deletion of the encapsulates-edge between M1 and C1, and creating C2 together with the outgoing encapsulates-edge to M1. Note that this is only possible because the node C1 in *MoveM1FromC1ToC2EffOrV2* is part of the *context extension*, allowing to skip its matching. Note also that applying rule *MoveM1FromC1ToC2EffOrV2* may not result in a model satisfying all constraints. For example, if Class C3 already encapsulates Method M1, then applying rule *MoveM1FromC1ToC2EffOrV2* would lead to a situation where the method is encapsulated by two classes, contrary to the constraint shown in Fig. 10. ♢

Properties of effect-oriented transformations. We ascertain that important properties of effect-oriented rules, originally proven in [28], are also true for our adapted and extended definition in $\mathbf{Sub}(T)$.

First, the base rule of an effect-oriented rule is a subrule of all its induced instances. This is an analogue to Prop. 1 in [28].

Proposition 2 (Base rule as subrule of induced rule)**.** *Given a typed graph T and an effect-oriented rule $p_\mathrm{e} = \big(p_\mathrm{b} = (L_\mathrm{b} \supseteq K_\mathrm{b} \subseteq R_\mathrm{b}), p_\mathrm{m} = (L_\mathrm{m} \supseteq K_\mathrm{b} \subseteq R_\mathrm{m}), k_\mathrm{ext}\big)$ in $\mathbf{Sub}(T)$ then, for any graph G in $\mathbf{Sub}(T)$ such that $L_\mathrm{b} \subseteq G$ and $R_\mathrm{b} \cap G \subseteq K_\mathrm{b}$, the induced rule $p^G_\mathrm{ind} = (L^G_\mathrm{ind} \supseteq K^G_\mathrm{ind} \subseteq R^G_\mathrm{ind})$ is pure and p_b is a subrule of p^G_ind.* ♢

Secondly, after an effect-oriented transformation, we have certain guarantees about the (non-)existence of elements. This is the analogue to Theorem 1 in [28]; however, the situation is much simpler in $\mathbf{Sub}(T)$.

Proposition 3 (Characterising effect-oriented transformations)**.** *Given a typed graph T, an effect-oriented rule $p_\mathrm{e} = \big(p_\mathrm{b} = (L_\mathrm{b} \supseteq K_\mathrm{b} \subseteq R_\mathrm{b}), p_\mathrm{m} = (L_\mathrm{m} \supseteq K_\mathrm{b} \subseteq R_\mathrm{m}), k_\mathrm{ext}\colon K_\mathrm{b} \subseteq K_\mathrm{m}\big)$ in $\mathbf{Sub}(T)$ and a graph G in $\mathbf{Sub}(T)$ such that $L_\mathrm{b} \subseteq G$, $R_\mathrm{b} \cap G \subseteq K_\mathrm{b}$, and the induced rule $p^G_\mathrm{ind} = (L^G_\mathrm{ind} \supseteq K^G_\mathrm{ind} \subseteq R^G_\mathrm{ind})$ is applicable to G, then*

1. *no element of $L_\mathrm{m} \setminus K_\mathrm{m}$ appears in H; and*
2. *every element of R_m appears in H,*

where $G \Longrightarrow_{p^G_\mathrm{ind}} H$ is the effect-oriented transformation induced by p_e on G. ♢

In contrast to the case of $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$, where the matching process for effect-oriented transformations ensures that an effect-oriented transformation is always possible when a match for the base rule exists, this is not longer true for our definition of effect-oriented transformation in $\mathbf{Sub}(T)$: The application of the rule p^G_ind induced by an effect-oriented rule p_e for a graph $G \in \mathbf{Sub}(T)$ might be hindered by the dangling-edge condition. In our application in Section 5, we prevent this by ensuring that in the maximal rules we construct, together with a node all potential adjacent edges are declared to be deleted.

4. Activation Diagrams and their Construction

In this section, we introduce our construction of *activation diagrams*. These capture, in a compact way, the interaction of the validity of constraints with atomic changes that are performed on graphs. From the activation diagrams, we derive consistency-preserving rules in Sect. 5.

Setting. We start by specifying the setting we are going to work in this and the following section. Compared to the previous sections, we consider one extension to the situation considered so far. In our envisioned application of solving selection problems (see Sects. 1 and 2), we actually do not work in a category $\mathbf{Sub}(T)$. Instead, we are only interested in graphs $G \in \mathbf{Sub}(T)$ that additionally contain a given graph P as subgraph, encoding the concrete problem instance that is to be solved. Only graphs containing P constitute solution candidates for that problem. We denote as $\mathbf{Sub}(T \supseteq P)$ this subcategory of $\mathbf{Sub}(T)$ that consists of graphs G with $P \subseteq G \subseteq T$ as objects and their inclusions as morphisms.

For illustration, consider again our running example of class responsibility assignment (Sect. 2.2) and, in particular, the type graph depicted in Fig. 2. The elements depicted in black and with solid borders serve to define *problem instances*: graphs containing sets of Attributes and Methods (and their interdependencies) that have to be assigned to Classes. Given such a problem instance P (i.e., a graph not containing Classes) as depicted in Fig. 3, we are interested in adding and removing Classes and encapsulates-edges to construct a design that optimally assigns Attributes and Methods to Classes (according to some metric). Thus, given P, fixing as container graph T some meaningful upper bound on allowed Classes and all potential assignments these allow for (as, e.g., in Fig. 5), solution candidates for this selection problem are graphs G such that $P \subseteq G \subseteq T$.

For rules and constraints, this has the following ramifications. Rules that only modify elements from $T \setminus P$ (i.e., Classes and their assignments) transform solution candidates into solution candidates. Rules that modify elements from P (i.e., change the given set of Attributes, Methods or their interdependencies), in contrast, would change the problem instance that is to be solved. Similarly, constraints where every occurring graph contains at least one element from $T \setminus P$ express desired properties of solutions (like every Method being assigned to a Class) whereas other constraints would express properties of problem instances. Moreover, strictly speaking, for rules and constraints to actually belong to $\mathbf{Sub}(T \supseteq P)$ each of their occurring graphs needs to contain P. Note that for $P = \emptyset$ we work in $\mathbf{Sub}(T)$; i.e., this is indeed a generalisation of the setting considered so far.

Summarizing, we work in the following setting.

Assumption 2. For the whole of Sects. 4 and 5, we assume to be given a type graph TG, a finite graph T, the *container graph*, typed over it, a graph $P \subseteq T$, the *problem graph*, and a set of constraints $\mathcal{C}$. Throughout, we work in $\mathbf{Sub}(T \supseteq P)$, which means the following: For any given rule $p = L \supseteq K \subseteq R$ we consider, it holds that $P \subseteq L, K, R$; in particular, this implies $L \setminus K \subseteq T \setminus P$ and $R \setminus K \subseteq T \setminus P$, i.e., p only deletes or creates elements from $T \setminus P$. Regarding constraints we assume that

- Every constraint $c \in \mathcal{C}$ is in one of the formats from Assumption 1, i.e., it is either *purely positive*, *purely negative* or in *implication format*.
- For any graph B occurring in any of the constraints $c \in \mathcal{C}$ it holds that $B \cap (T \setminus P) \neq \emptyset$, i.e., every such graph contains at least one element from $T \setminus P$. ♢

Remark 6. Note that in $\mathbf{Sub}(T \supseteq P)$, since P is a graph, for every node x of some graph $G \in \mathbf{Sub}(T \supseteq P)$, if x stems from $T \setminus P$ all of its incident edges (in G) also stem from $T \setminus P$. We will regularly silently use that in the following.

Regarding the condition on constraints that each of their graphs contains at least one element from $T \setminus P$, we can trivially achieve this in the following way. Per se, we do not expect this to hold for user-defined constraints (or for constraints resulting from the translation we develop in [27]). However, in any case we can assume a constraint to be given in normal form according to Assumption 1. In such constraints, still literals $\exists\, B$ or $\neg\exists\, D$ might occur such that $B \cap (T \setminus P) = \emptyset$ or $D \cap (T \setminus P) = \emptyset$, respectively. This implies $B \subseteq P$ or $D \subseteq P$, respectively. Importantly, this means that for every graph $G \in \mathbf{Sub}(T \supseteq P)$, $G \models_{\mathcal{I}} \exists\, B$

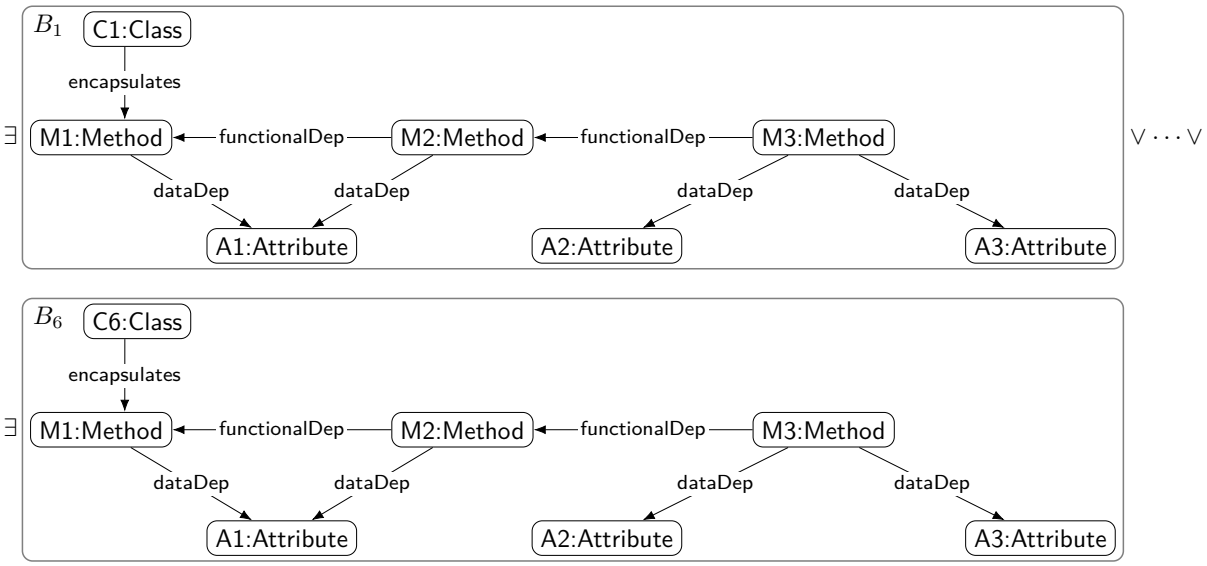


Figure 21: Simplified conjunct of constraint $c_{\text{lb}}^{\mathbf{Sub}(T)}$ in $\mathbf{Sub}(T)$, requiring Method M1 to be assigned to at least one of Classes C1, ..., C6 from T

and $G \not\models_{\mathcal{I}} \neg\exists\, D$. Therefore, we can drop every constraint c containing such a literal $\exists\, B$ from our constraint set $\mathcal{C}$—the constraint c evaluates to true for every graph $G \in \mathbf{Sub}(T \supseteq P)$. Similarly, we can omit every such negative literal $\neg\exists\, D$ from its constraint c—no graph $G \in \mathbf{Sub}(T \supseteq P)$ can satisfy c by satisfying this literal. (More formally, in the second case, we substitute $\neg\exists\, D$ by false in c, which does not change the semantics for graphs $G \in \mathbf{Sub}(T \supseteq P)$, and then omit false by applying the identity rule for $\vee$.) In summary, this requirement on constraints does not pose any real restriction but serves to ensure that the given constraints express properties of graphs $G \in \mathbf{Sub}(T \supseteq P)$ and not just properties of P. Similarly, we can achieve that every graph X occurring in a constraint or rule contains P by just considering $X \cup P$; it is also easily verified that for the just discussed rules and constraints this does not change the semantics (in $\mathbf{Sub}(T \supseteq P)$). ♢

Example 9 (Representation of the CRA problem in terms of assumption)**.** We illustrate the above assumption using the CRA problem. Thus, let the graph TG from Fig. 2 be our type graph, T from Fig. 5 be our container graph, the graph P from Fig. 3 our problem graph, and let the constraint set $\mathcal{C}$ consist of the constraints requiring all Methods and Attributes to be allocated to exactly one Class and all Attributes being private. In accordance with Assumption 2, these constraints are given in the following form. First, the constraints requiring every Method and every Attribute to be assigned to at least one Class are purely positive. Originally, when translating the according constraints from $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$, we receive constraints in implication format as depicted in Fig. 8. Following the above remark, however, we can omit the negative literal forbidding the existence of a Method (or Attribute) since, as elements belonging to P, these are bound to exist. Figure 21 shows the resulting simplified constraint for Method M1, the clause consisting of six positive literals giving the six different possibilities to assign M1 to one of the available Classes. Analogous constraints (clauses) are given for the other five Methods and Attributes. *As a convention, to reduce the size of our figures, we refrain from always displaying the whole graph P for constraints as well as for rules, but in the following only display the elements from $X \setminus P$ (where X is some graph from a rule or a constraint) and those elements from P that are necessary to complete $X \setminus P$ into a graph (i.e., nodes from P that are adjacent to an edge from $X \setminus P$).* For instance, we compactly represent the constraint from Fig. 21 as shown in Fig. 22.

Figure 10 shows the fifteen clauses that together forbid Method M1 to be assigned to more than one Class by explicitly excluding every possible pair of assignments. We consider these fifteen clauses as individual negative constraints. In total, we have 90 such constraints—fifteen for every Method or Attribute from T.

The constraint ensuring Attributes being private is displayed in Fig. 12 for the data dependency between

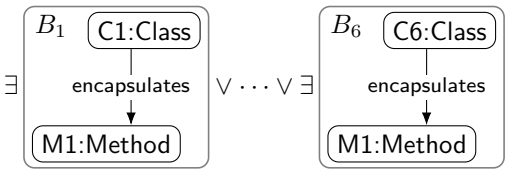


Figure 22: Compact version of the simplified conjunct of constraint $c_{\text{lb}}^{\mathbf{Sub}(T)}$ in $\mathbf{Sub}(T)$, requiring Method M1 to be assigned to at least one of Classes C1, ..., C6 from T

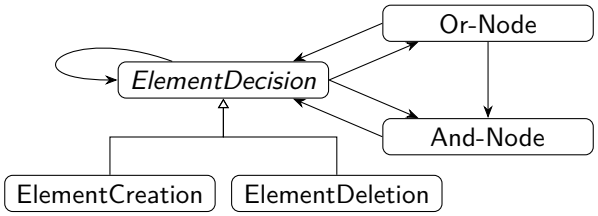


Figure 23: Type graph for activation diagrams (making use of inheritance for compactness)

M1 and A1 and the case that A1 is assigned to C1. This constraint is an implication constraint. In total, we have 24 of such constraints, namely one for each pair of data dependency and Class.

Finally, regarding rules, we assume that all rules we encounter only manipulate Classes and/or encapsulates-edges. In particular, the rules do not create or delete Methods, Attributes and their dependencies. All rules we have seen so far satisfy this assumption. ♢

In the following, we first abstractly define activation diagrams and then describe how we derive activation diagrams from given container and problem graph and a set of constraints. The process is a generalisation of the process developed in [10] for the specific case of feature configuration.

Definition 9 (Activation diagram)**.** An *activation diagram* is a directed graph that is typed over the type graph displayed in Fig. 23 (where we make use of inheritance for a more compact representation). This means, an activation diagram has nodes that represent the creation of elements, nodes that represent the deletion of elements and nodes that group actions, namely Or-Nodes and And-Nodes. And-Nodes can only reference ElementDecisions, i.e., nodes of type ElementCreation or ElementDeletion. ♢

We build activation diagrams to capture the interaction of the creation or deletion of elements with the validity of constraints. Intuitively, the outgoing edges of an ElementDecision node capture what actions can or must be taken so that the given constraints remain valid after that action. For instance, given the decision to create the element x and a constraint that forbids the graph D to exist of which x is a part, we know we need to guarantee the non-existence of at least one $y \in D$ with $y \neq x$. We represent this by creating edges to the ElementDeletion nodes for all elements of $D \setminus x$, grouped by an Or-Node to indicate that it suffices to perform one of these deletions. The building process for activation diagrams is driven by *interaction principles* that capture the interaction of the creation and deletion of elements and the validity of constraints, and we check for all possible such interactions. The following construction of activation diagrams abstracts from algorithmic details because these are not relevant for the goal of this paper, which is to establish the correctness of the construction. However, Horcas et al. have presented an efficient implementation of their feature activation diagrams via depth-first search [10] and we are confident that their algorithm can be adapted to our case.

Construction 1 (Activation diagram)**.** Given a type graph TG, graphs $T \supseteq P$ typed over it, serving as container and as problem graph, and a finite set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ in normal form according to Assumption 2, the activation diagram for T, P and $\mathcal{C}$, denoted as $\text{AD}_{T \supseteq P, \mathcal{C}}$, is constructed as follows:

1. For each element x of $T \setminus P$ (node or edge), add one corresponding ElementCreation node and one corresponding ElementDeletion node to $\text{AD}_{T \supseteq P, \mathcal{C}}$. (Given an element x of $T \setminus P$, we will always refer

via $+x$ to its corresponding ElementCreation node and via $-x$ to its corresponding ElementDeletion node in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$.)

2. For each ElementCreation node n of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ that corresponds to an edge e of $T \setminus P$, add edges from n to the ElementCreation nodes that correspond to the nodes adjacent to e in $T \setminus P$.

3. For each ElementDeletion node n of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ that corresponds to a node a of $T \setminus P$, add edges from n to the ElementDeletion nodes that correspond to the edges adjacent to a in $T \setminus P$.

4. For each ElementDecision node $n \in \mathrm{AD}_{T \supseteq P, \mathcal{C}}$, extend $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ by applying all interaction principles from Definition 10 to n that are applicable given $\mathcal{C}$. ♢

Before presenting the interaction principles that drive the construction of activation diagrams, we remark on two technical details.

Remark 7.

1. Points 2. and 3. in the above construction serve to ensure that following ElementDecisions actually results in a graph: Together with an edge, its adjacent nodes are created, and together with a node, its adjacent edges are deleted.

2. For simplicity of notation, we use $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ to denote the final activation diagram as well as its intermediate states during the iterative construction. ♢

The interaction between decisions to create or delete elements and constraints is captured by four interaction principles that underlie the construction of activation diagrams. We formulate these principles with an eye towards minimality: For instance, if the existence of a certain graph necessitates a certain reaction, we only have to trigger this reaction if edges or isolated nodes of the respective graph are added (the creation of a node without its incident edge can never complete the graph). Similarly, since the deletion of a node triggers the deletion of its adjacent edges anyhow, it suffices to delete edges or isolated nodes whenever we have to destroy a graph. To be precise, by *isolated node* in the following we refer to a node $x \in T \setminus P$ such that x has no incident edge in the concerned graph.

Definition 10 (Interaction principles)**.** Let a type graph TG, graphs $T \supseteq P$ typed over it and a set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ in normal form according to Assumption 2 be given, and let $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ denote the corresponding activation diagram (in its current state during the construction). We have the following *interaction principles*.

- **Creation principles:** Let n be an ElementCreation node from $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$, corresponding to the node or edge $x \in T \setminus P$.

  1. CreationNeg: *Intuition: For each purely negative constraint $c = \neg\exists\,(D, \mathsf{true})$ from $\mathcal{C}$, if $x \in D$ and x is an edge or an isolated node of D, we need to ensure that at least one element of $D \setminus x$ does* not *exist.*

     Therefore, if $x \in D$ and x is an edge or an isolated node of D, we add the ElementDeletion nodes corresponding to other edges and isolated nodes of D as alternative consequences to the ElementCreation node $+x$. That is, we add to $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ the Or node and edges that are depicted dashed in Fig. 24 (we later explain the labelling of Or nodes and edges). There, the ElementDeletion nodes $-y_1, \ldots, -y_u$ correspond to those elements of $(D \cap (P \setminus T)) \setminus x$ that are edges or nodes isolated in D.

  2. CreationImpl: *Intuition: For each implication condition $c = \exists\,(D, \mathsf{true}) \implies \vee_{i=1,\ldots,t} \exists\,(B_i, \mathsf{true})$, if $x \in D$ and x is an edge or an isolated node of D, we need to either ensure that another element of $D \setminus x$ does not exist or that also one of the graphs B_i exists.*

     Therefore, if $x \in D$ and x is an edge or an isolated node of D, we add the deletion of other edges and isolated nodes from D and the creations of the graphs $B_1, \ldots B_t$ as alternative consequences to

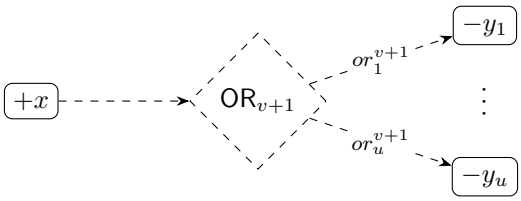


Figure 24: Illustration of the interaction principle CreationNeg. Dashed elements are newly inserted in $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ whereas elements with solid borders (the ElementDecision nodes) already exist.

the ElementCreation node $+x$. For the creation of a graph, we group the creation of all its elements using an And node. That is, we add to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ the Or node, the And nodes and the edges that are depicted dashed in Fig. 25, where $\{y_1,\dots,y_u\}$ are those elements from $(D\cap(T\setminus P))\setminus\{x\}$ that are edges or nodes isolated in D and analogously $\{z_1^i,\dots,z_{w_i}^i\}$ are those elements from $(B_i\cap(T\setminus P))\setminus\{x\}$ that are edges or nodes isolated in B_i.

- **Deletion principles:** Let n be an ElementDeletion node from $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$, corresponding to the node or edge $x\in T\setminus P$.

  1. DeletionPos: *Intuition: For each purely positive constraint $c=\vee_{i=1,\dots,t}\exists(B_i,\mathsf{true})$, if x occurs in a graph B_{i_0} with $i_0\in\{1,\dots,t\}$, then we need to ensure that one of the graphs B_j exists, where $j\in\{1,\dots,t\}$, $j\neq i_0$ and x does not occur in B_j.*
     Therefore, we add the creation of these graphs as alternative consequences to the ElementDeletion node $-x$. That is, if x is an edge or an isolated node of such a graph B_{i_0}, we add to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ the Or node and And nodes depicted dashed in Fig. 26, where $\{z_1^j,\dots,z_{w_j}^j\}$ are the edges and nodes isolated in $B_j\cap(T\setminus P)$ and $j=1,\dots,q$ denote those graphs B_j from c that do not contain x.
  2. DeletionImpl: *Intuition: For each implication condition $c=\exists(D,\mathsf{true})\implies\vee_{i=1,\dots,t}\exists(B_i,\mathsf{true})$, if x is deleted, $x\in B_{i_0}\cap(T\setminus P)$, where $i_0\in\{1,\dots,t\}$, and $x\notin D$, we need to ensure that D does* not exist *or that one of the graphs B_j exists, where $j\in\{1,\dots,t\}$, $j\neq i_0$ and $x\notin B_j$.*
     Therefore, if $x\notin D$ and x is an edge or an isolated node of one such graph B_{i_0}, we add the deletion of an edge or an isolated node of D or the creation of one of the graphs B_j as alternative consequences to the ElementDeletion node $-x$. That is, we add to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ the Or node and the And nodes depicted dashed in Fig. 27. There, $\{y_1,\dots,y_u\}$ are the edges and isolated nodes of $D\cap(T\setminus P)$, $\{z_1^j,\dots,z_{w_j}^j\}$ are the edges and isolated nodes of $B_j\cap(T\setminus P)$ and $j=1,\dots,q$ denote those graphs B_j from c that do not contain x. ♢

Remark 8 (Labelling of activation diagrams and applicability and non-minimality of interaction principles).

- Note that every application of any interaction principle inserts exactly one Or node into the activation diagram; we refer to that node as *the Or node of the application of the principle.*
- For the later derivation of constraint-preserving rules, we enumerate Or nodes and outgoing edges of Or and And nodes; this can be performed during construction. That is, if there are already v Or nodes in $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ upon the application of an interaction principle, we label the newly inserted Or node with $v+1$ and enumerate its outgoing edges accordingly as $or_1^{v+1},\dots,or_w^{v+1}$, where w corresponds to the number of outgoing edges. The outgoing edges of And nodes receive the label of its incoming edge (starting at an Or node) that is guaranteed to exist by construction.
- We can safely assume that our interaction principles are always applicable, i.e., the sets of nodes to which we insert edges are never empty. For example, in case of our first principle CreationNeg, if x is the only element of $D\cap(T\setminus P)$ or, in case x is an edge, $D\cap(T\setminus P)$ only consists of x and its adjacent node(s), no graph $G\in\mathbf{Sub}(T\supseteq P)$ can satisfy the given negative constraint c and simultaneously contain the element x, i.e., the element x is a "dead" element. Thus, the given problem could be

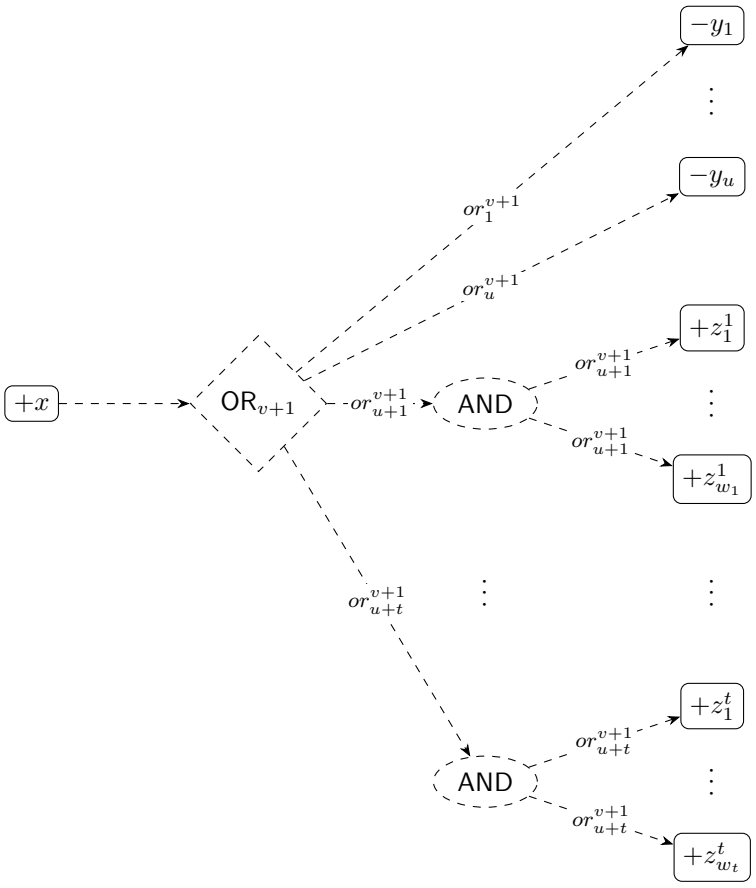


Figure 25: Illustration of the interaction principle CreationImpl. Dashed elements are newly inserted in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ whereas elements with solid borders (the ElementDecision nodes) already exist. Note that the ElementCreation nodes in the right column do not need to be distinct because the same element can occur in several of the involved graphs.

remodelled by removing x from T and the constraint c from the constraint set $\mathcal{C}$. For the other interaction principles, similar arguments can be made. Note, however, that in the following we do not make use of this assumption. Instead, if an interaction principle is not applicable, this just leads to the result that, at the end, we cannot derive a consistency-preserving rule for the case where the originally given rule performs the action for which the interaction principle was not applicable. This is a correct result because inapplicability of an interaction principle basically means that performing the action in question inevitably causes a violation of the considered constraint (as will become clear by our proofs).

- Note that CreationNeg is a subprinciple of CreationImpl and, analogously, DeletionPos is a subprinciple of DeletionImpl. That is, we do not present our approach in the minimal number of interaction principles but specifically present the interaction principles for the special cases of purely positive and purely negative constraints. Note, however, that this does not affect the resulting activation diagrams: The principle CreationNeg just explicitly shows to what the application of CreationImpl amounts when applied to a purely negative constraint and, analogously, DeletionPos explicitly shows to what the application of DeletionImpl amounts when applied to a purely positive constraint. In this way, we explicitly show how we treat each case of a rather natural presentation of constraints. ♢

Example 10 (Snippet of activation diagram and illustration of interaction principles)**.** Figure 28 shows a snippet of the activation diagram resulting for our CRA running example as introduced in Example 9. We start the construction by adding to $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ an ElementCreation node +Ci and an ElementDeletion node -Ci for

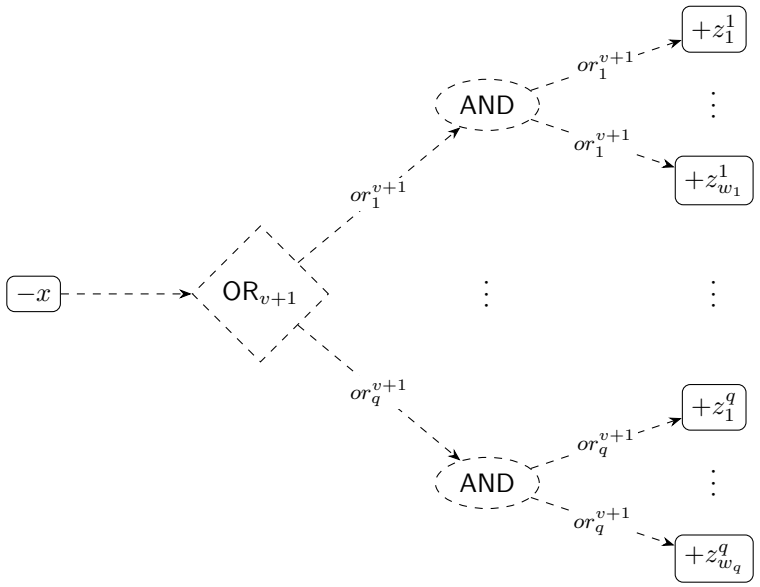


Figure 26: Illustration of the interaction principle DeletionPos. Dashed elements are newly inserted in $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ whereas elements with solid borders (the ElementDecision nodes) already exist. Note that the ElementCreation nodes in the right column do not need to be distinct because the same element can occur in several of the involved graphs.

every Class Ci, $i = 1, \ldots, 6$. Likewise, we add one ElementCreation and one ElementDeletion node +eCiFj resp. -eCiFj for every encapsulates-edge of T (where, again, Ci denotes the Class and Fj $\in$ {A1, A2, A3, M1, M2, M3}, i.e., Fj denotes the feature). This gives us one ElementCreation and one ElementDeletion node per element of $T \setminus P$ (i.e., in total, we have 6 + 36 ElementCreation and the same amount of ElementDeletion nodes). For space reasons, we only show a snippet of the resulting diagram and, moreover, omit And nodes that are unnecessary because they only point to one node.

Centrally, the snippet displays the consequences of node +eC2M1, i.e., of the node corresponding to the creation of the encapsulates-edge between Class C2 and Method M1. First, this node references the ElementCreation node corresponding to C2: In the second step of Construction 1, for every edge from $T \setminus P$ (i.e., for every encapsulates-edge in our example) we add edges from the corresponding ElementCreation node to the ElementCreation nodes corresponding to the adjacent nodes. Figure 29a illustrates that for the encapsulates-edge eC2M1 (here, and in the following diagrams, the elements newly inserted into the activation diagram are indicated as dashed). Similarly, in Step 3 of the construction, for every node from $T \setminus P$ (i.e., for every Class in our example), we add in $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ an edge from the ElementDeletion node corresponding to that node to the ElementDeletion nodes corresponding to incident edges. Figure 29b illustrates that for Class C1 and its outgoing encapsulates-edges (not part of the snippet of the activation diagram in Fig. 28).

Following this, interaction principles are applied to build the activation diagram. Directly to node +eC2M1, the interaction principle CreationNeg is applicable with regard to each of the constraints forbidding M1 to be assigned to C2 and another Class simultaneously. Figure 30 illustrates the application of the principle for Class C1. Note that it suffices to trigger that interaction principle for the two edges eC1M1 and eC2M1 appearing in the negative constraint—creation of Classes C1 or C2 alone cannot create the forbidden pattern. Similarly, deleting eC1M1 suffices; deleting C1 would additionally trigger to delete eC1M1 anyhow. Analogously, every ElementCreation node corresponding to an edge triggers various applications of the interaction principle CreationNeg with regard to the various constraints forbidding Methods and Attributes to be assigned to two Classes; in our snippet, we just indicate this via the references to the dots. The node OR_6 is a further node from our snippet inserted in that way.

The node -eC1M1 triggers the interaction principles DeletionPos and DeletionImpl. DeletionPos is triggered with regard to the positive constraint requiring M1 to be assigned to a Class and Or_2 and

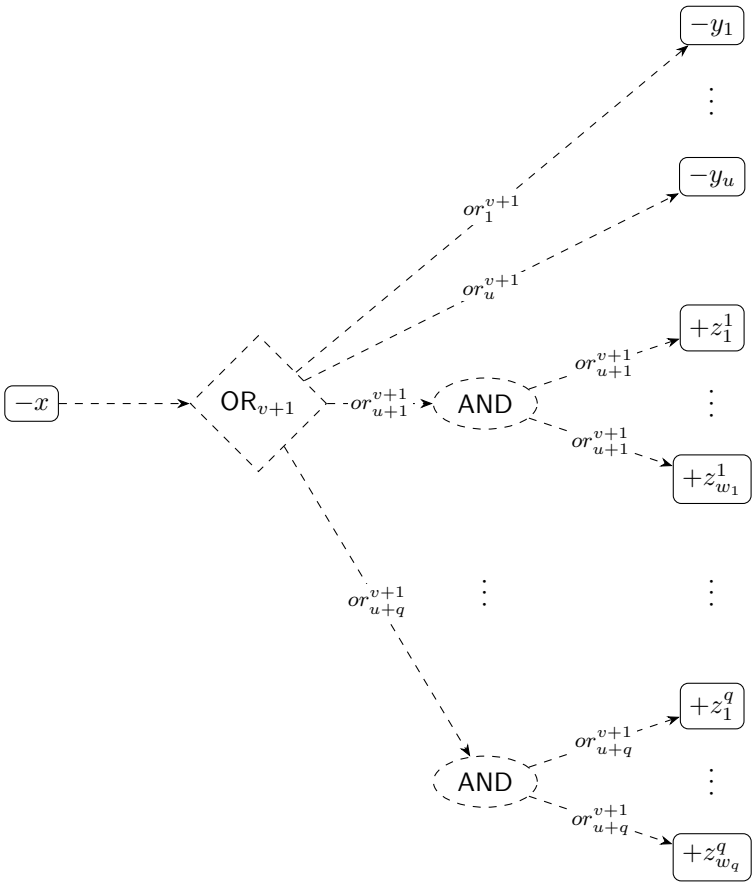


Figure 27: Illustration of the interaction principle DeletionImpl. Dashed elements are newly inserted in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ whereas elements with solid borders (the ElementDecision nodes) already exist. Note that the ElementCreation nodes in the right column do not need to be distinct because the same element can occur in several of the involved graphs.

its outgoing edges are inserted by the application. Figure 31 illustrates the application of that principle, including the And nodes we omit in Fig. 28. Note again that is suffices to apply that principle for the edge eC1M1 and not also for the Class C1: The ElementDeletion node corresponding to C1 references the ElementDeletion node for eC1M1 and we will read off rules in such a way that therefore consequences of the deletion of eC1M1 are also implied by the deletion of C1. Similarly, it suffices to create references to the ElementCreation nodes +eCiM1, where $i = 2, \ldots, 5$, because these reference the ElementCreation nodes corresponding to their incident nodes; in this way, creation of the edge will automatically trigger the creation of the necessary Class. The node OR_4 is a further node in Fig. 28 that is inserted by an analogous application of principle DeletionPos to node -eC1A1.

DeletionImpl is triggered by node -eC1M1 with regard to the constraint requiring A1 to be private and Or_3 and its outgoing edge are inserted by the application. Figure 32 illustrates the snippet of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ resulting from the application. Note that in this specific case, because the consequent of the constraint consists of a single graph, deletion of one of the edges or isolated nodes of the graph of the antecedent that stem from $T \setminus P$ is the only available possibility (and there is only one such element here, namely edge eC1A1).

Finally, the ElementCreation node +eC2A1 triggers the application of the principle CreationImpl with regard to the constraint requiring A1 to be private, leading to the insertion of Or_5 and its outgoing edge. Figure 33 illustrates this application, again including the And node we omit in Fig. 28. ♢

The following proposition shows that, in the worst case, the size of an activation diagram depends linearly

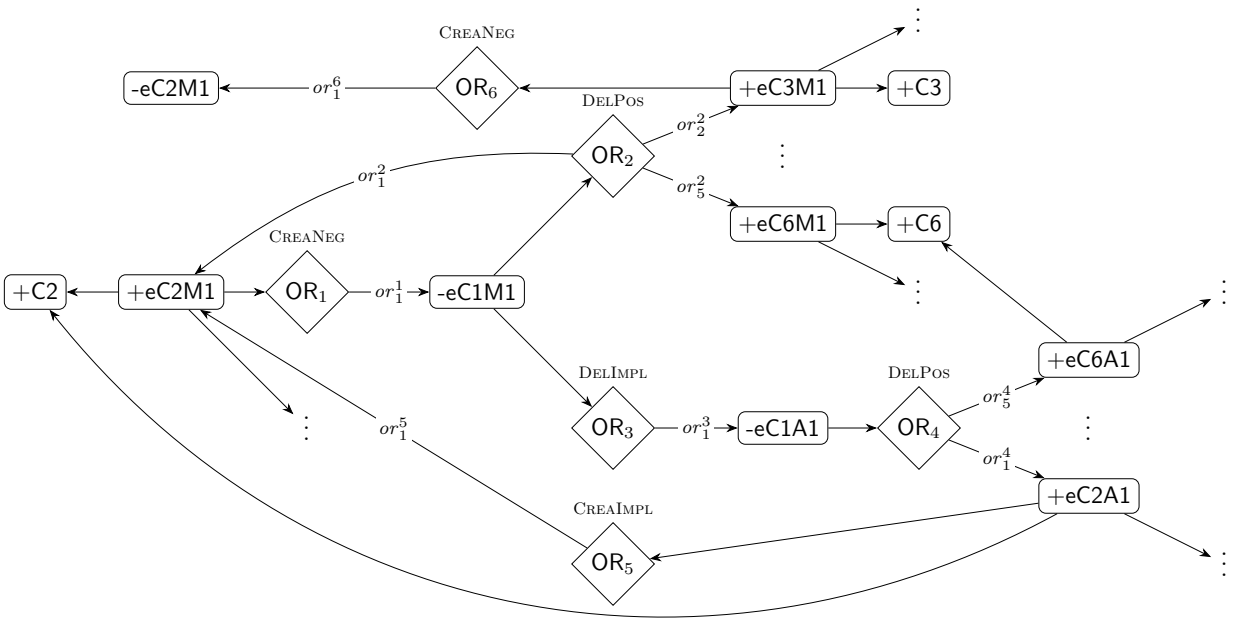


Figure 28: Snippet of the activation diagram for our CRA running example, where labels above the Or nodes indicate which interaction principle has been applied



(a) . . . basic consistency of edge creation

(b) . . . basic consistency of node deletion (not part of snippet of activation diagram)

Figure 29: Illustration of edges inserted into $\mathrm{AD}_{T \supseteq P,\mathcal{C}}$ to ensure. . .

on the number of constraints but quadratically on the size of the largest graph occurring in one of the constraints and on the maximal number of graphs occurring in one of the constraints. In particular, activation diagrams are finite. After that basic clarification, we present a result that shows in which way interaction principles (and, by that, activation diagrams) indeed capture the interaction of graph modifications and the validity of constraints.

Proposition 4 (Size of activation diagram)**.** *Given a type graph TG, a finite graph T typed over it, a problem graph $P \subseteq T$ and a finite set $\mathcal{C}$ of constraints in* $\mathbf{Sub}(T \supseteq P)$ *in normal form according to Assumption 2, for the corresponding activation diagram $\mathrm{AD}_{T \supseteq P,\mathcal{C}}$ it holds that*

$$|V_{\mathrm{AD}_{T \supseteq P,\mathcal{C}}}| \leq 2\left(|T \setminus P| + |\mathcal{C}|(t_{\max}+1)^2|B_{\max} \setminus P|\right), \tag{1}$$

and

$$|E_{\mathrm{AD}_{T \supseteq P,\mathcal{C}}}| \leq 4\left(|E_T \setminus E_P| + |\mathcal{C}|(t_{\max}+1)^2|B_{\max} \setminus P|^2\right) \tag{2}$$

where $|V_{\mathrm{AD}_{T \supseteq P,\mathcal{C}}}|$ and $|E_{\mathrm{AD}_{T \supseteq P,\mathcal{C}}}|$ denote the numbers of nodes and edges of the activation diagram $\mathrm{AD}_{T \supseteq P,\mathcal{C}}$, respectively, $|E_T|$ the number of edges of T, $t_{\max}$ is the maximal number of graphs occurring in any of the

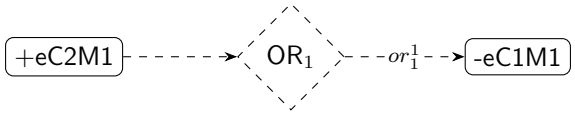


Figure 30: Illustration of principle CreationNeg on the example of edge eC2M1

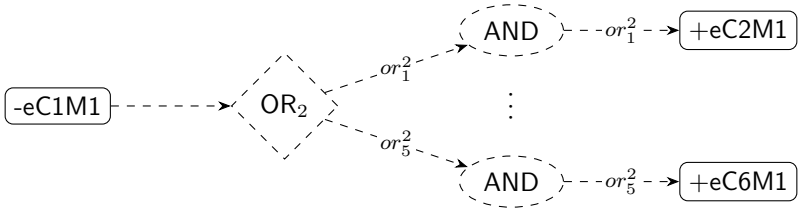


Figure 31: Illustration of principle DeletionPos on the example of edge eC1M1

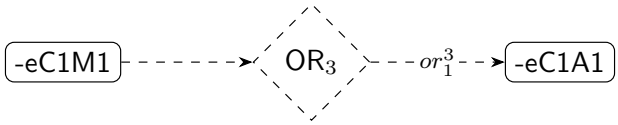


Figure 32: Illustration of principle DeletionImpl on the example of assignment of Method M1 to Class C1

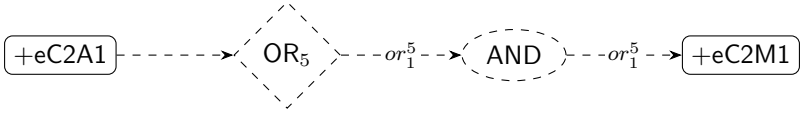


Figure 33: Illustration of principle CreationImpl on the example of data dependency between M1 and A1

constraints $c \in \mathcal{C}$ and $|B_{\max}|$ denotes the size (sum of nodes and edges) of the largest graph occurring in any of the constraints $c \in \mathcal{C}$. ♢

Remark 9. In practical examples, the number $|B_{\max}|$ tends to be rather small. The crucial factors, actually, are the number of constraints $|\mathcal{C}|$ and $t_{\max}$, which tend to be large for constraints in $\mathbf{Sub}(T)$ because these list every possible concrete situation individually. (We make this intuition technically precise in [27], where we translate constraints from $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$ to $\mathbf{Sub}(T)$.)

Moreover, in principle, it is possible to construct activation diagrams lazily. This is (at least partially) done in the implementation of feature activation diagrams that Martinez, Horcas et al. provide [43, 10]: the diagram is constructed lazily as feature-activation decisions are added. Essentially, adding a feature-activation decision "adds" all directly implied feature-activation decisions and activation-diagram nodes and stops where these nodes are already part of the diagram. This constructs a (potentially partial) feature activation diagram that (as we will see later) would allow one to read off the VB rule derived for the feature-activation decision for which the process was started (and also for any of the feature decisions encountered in the process). In practice, the implementation then loops over all real-optional features, but in principle, this could also be done online during the search for an optimal feature configuration. This analogously transfers to the construction of activation diagrams in general.

For these reasons, we are convinced that computing activation diagrams can be performed efficiently. ♢

Example 11. In our running example (as introduced in Example 9), the above proposition guarantees that the resulting activation diagram has at most 47,124 nodes and at most 376,464 edges.

Revisiting Example 10 more closely, the actual size of the resulting activation diagram is 348 nodes (of which 264 are Or nodes) and 744 edges. In the first three steps of the construction, we add 84 ElementDecision nodes and 72 edges to the diagram. The applications of the interaction principles then add the following structure: We apply the principle CreationNeg 180 times, namely 5 times per every encapsulates edge of T (or twice per each of the occurring 90 negative constraints expressing upper bounds), each time introducing one Or node together with an incoming and an outgoing edge. We apply the principle CreationImpl 24 times, namely 6 times per every dataDependency edge of T (or once per each of the occurring 24 implication constraints expressing privateness of attributes), each time introducing one Or node together with an incoming and an outgoing edge. We apply the principle DeletionPos 36 times, namely 6 times per feature (Method or Attribute) of T (or six times per each of the occurring 6 positive constraint expressing lower bounds), each time introducing one Or node together with an incoming and 5 outgoing edges. We apply the principle DeletionImpl 24 times, namely 6 times per every dataDependency edge of T (or once per each of the occurring 24 implication constraints expressing privateness of attributes), each time introducing one Or node together with an incoming and an outgoing edge.

That the actual numbers are much lower than estimated by our proposition is due to a number of factors. The most important one is that the squared maximal length (here: $t_{\max} = 6$) is multiplied with the number of constraints but in our example only 6 constraints are of length 6, whereas 24 are of length 2 and 90 are of length 1. Moreover, the graphs in our considered constraints do not contain any isolated nodes; thus, interaction principles are only triggered for the edges and never for nodes from $T \setminus P$. We expect both these phenomena to occur frequently in applications and therefore the above proposition to regularly grossly overestimate the size of the resulting activation diagram. ♢

For formally stating in which way activation diagrams capture the interaction of graph modifications and the validity of constraints and, in particular, in which sense an activation diagram captures "repair actions", we first introduce two notions of how a graph can satisfy an ElementDecision node: either directly or its consequences (regarding one interaction principle). Our formal result will be that if a graph directly satisfies an ElementDecision node (i.e., contains or not contains the corresponding element) then it satisfies a given constraint if and only if it also satisfies the consequences the corresponding interaction principle adds to the ElementDecision node. In that sense, the consequences of the ElementDecision capture exactly the possible "repair actions". This will be a central ingredient for the correctness of our approach and its ability (under certain circumstances) to simulate transformation sequences of the original rules.

Definition 11 (Satisfaction of node or subgraph of activation diagram)**.** Given a type graph TG, a finite graph T typed over it, a problem graph $P \subseteq T$, a finite set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ in normal form according to Assumption 2, let $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ be the corresponding activation diagram constructed according to Construction 1. Let $G \in \mathbf{Sub}(T \supseteq P)$ be a graph and let n be a node of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$. We say that G *satisfies* n, denoted as $G \models n$, if

1. n is an ElementCreation node, corresponding to element $x \in T \setminus P$, and $x \in G$;
2. n is an ElementDeletion node, corresponding to element $x \in T \setminus P$, and $x \notin G$;
3. n is an And node and for every node $n' \in \mathrm{AD}_{T \supseteq P, \mathcal{C}}$ such that there is an edge from n to n' it holds that $G \models n'$; or
4. n is an Or node and there exists a node $n' \in \mathrm{AD}_{T \supseteq P, \mathcal{C}}$ such that there is an edge from n to n' and it holds that $G \models n'$.

A graph G *satisfies* a subgraph $W \subseteq \mathrm{AD}_{T \supseteq P, \mathcal{C}}$, denoted as $G \models W$, if G satisfies every node of W, i.e., if $G \models x$ for all $x \in V_W$. ♢

Via *consequences of an ElementDecision node n with regard to a constraint c* we refer to that part of an activation diagram we have added to it to deal with the consequences for the validity of c of taking the decision captured by n. As discussed before Definition 10, to make activation diagrams more compact, we only trigger interaction principles for edges or isolated nodes from the graphs of the constraints and, for non-isolated nodes, rely on the interaction principles triggered for their adjacent edge(s) to introduce the necessary consequences. This makes the following definition slightly more technical as we therefore also have to consider the ElementDecision nodes corresponding to these adjacent edges.

Definition 12 (Satisfaction of consequences of ElementDecision node with regard to constraint)**.** Given a type graph TG, a finite graph T typed over it, a problem graph $P \subseteq T$, a finite set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ and the corresponding activation diagram $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$, we say that an ElementDecision node n of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ *interacts* with constraint $c \in \mathcal{C}$ if n corresponds to an element $x \in T \setminus P$ and either (i) n is an ElementCreation node, x appears in a negative literal D of c and x is an edge or an isolated node of D or (ii) n is an ElementDeletion node and x appears in a positive literal of c.

Given an ElementDecision node n and a constraint c with which n interacts, the *consequences of n with regard to c*, denoted as $\mathrm{post}(n, c)$, are the subgraph of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ that is induced by the following nodes:

1. The node n belongs to $V_{\mathrm{post}(n,c)}$.
2. If n is an ElementDeletion node corresponding to a node $x \in T \setminus P$ that does not appear isolated in any positive literal of the constraint c, all ElementDeletion nodes corresponding to edges $e \in T \setminus P$ adjacent to x and appearing in any positive literal of c belong to $V_{\mathrm{post}(n,c)}$. (That is, ElementDeletion nodes that n references via edges inserted in Step 3. of the construction of activation diagrams (Construction 1) belong to $V_{\mathrm{post}(n,c)}$.)
3. Since n interacts with c, constraint c triggers the application of an interaction principle during Step 4. of the construction of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$. The nodes created by that application belong to $V_{\mathrm{post}(n,c)}$.

We say that a graph G *satisfies the consequences of n with regard to c* if G satisfies the subgraph $\mathrm{post}(n, c)$ of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$, i.e., if $G \models \mathrm{post}(n, c)$. ♢

Remark 10. Note that the kind of interaction principle that can be applied is uniquely determined by a given node n of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ and the format of the constraint c. Either this interaction principle is directly applied to n or, if n is an ElementDeletion node whose corresponding node x does not appear isolated in any positive literal of c, the interaction principle is triggered for (at least) one edge adjacent to x. In the latter case, this interaction principle (then either DeletionPos or DeletionImpl) might be applied several times, for different such adjacent edges. However, each such application is structurally identical (i.e., while each such application creates its distinct Or node and maybe And nodes, the structure and, in particular, the referenced ElementDecision nodes are exactly the same). ♢

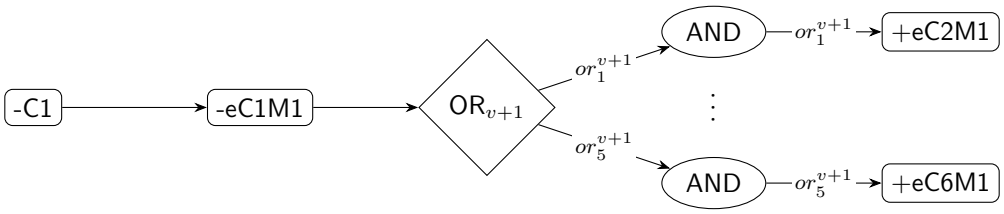


Figure 34: Subgraph post(-C1, c) of $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ for the constraint c requiring Method M1 to be assigned to one Class

Example 12. Figure 34 shows the subgraph post(-C1, c) of $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ for the constraint c requiring Method M1 to be assigned to one Class. A graph $G \in \mathbf{Sub}(T \supseteq P)$ satisfies post(-C1, c) if it does not contain any of the potential edges between Class C1 and a feature but contains one of the edges assigning Method M1 to another Class. ♢

Theorem 5 (Activation diagrams capture interplay of graph modifications and validity of constraints)**.** *Given a type graph TG, a finite graph T typed over it, a problem graph $P \subseteq T$, a finite set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ in normal form according to Assumption 2, let $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ be the corresponding activation diagram constructed according to Construction 1. Let $G \in \mathbf{Sub}(T \supseteq P)$ be arbitrary and either (i) n be an ElementCreation node n of $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$, corresponding to some element $x \in G \setminus P$, and let $c \in \mathcal{C}$ be a constraint such that x appears in a negative literal of c as edge or as isolated node, or (ii) n be an ElementDeletion node n of $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$, corresponding to some element $x \in G \setminus P$, and let $c \in \mathcal{C}$ be a constraint such that x appears in a positive literal of c. Then it holds that if G satisfies n, then G satisfies c if and only if G satisfies the consequences of n; i.e.,*

$$G \models n \implies \big(G \models c \iff G \models \mathrm{post}(n, c)\big).$$ ♢

For our construction of consistency-preserving VB rules in the next section, we need to formalise the notion that deciding for one alternative at an Or node of an activation diagram necessitates further decisions. Crucially, for each potential decision we just encode the *direct* consequences; in this way, when constructing rules, we avoid the need to enumerate all paths through the activation diagram.

Definition 13 (Triggering of direct consequences)**.** Given an activation diagram $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$, we say that an edge or_j^i, starting at Or node Or_i, *directly triggers* Or node Or_l if there is a directed path $e_1; \ldots; e_k$ in $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ such that $e_1 = or_j^i$, the target node of e_k is Or_l and none of the target nodes of $e_2, \ldots, e_{k-1}$ is an Or node. Analogously, we say that an ElementDecision node n *directly triggers* Or node Or_l if there is a directed path $e_1; \ldots; e_k$ in $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ such that n is the source node of e_1, the target node of e_k is Or_l and none of the target nodes of $e_1, \ldots, e_{k-1}$ is an Or node. ♢

5. Deriving Non-Blocking Consistency-Preserving Rules

In this section, we derive *non-blocking consistency-preserving variants* for a given rule using activation diagrams. The obvious idea is, given a rule, to use an activation diagram to transitively read off the consequences of the rule's actions and add these to the actions of the rule. This approach, however, has two problems. First, in general, there are various ways to construct from a given rule a non-blocking consistency-preserving one, as witnessed by the alternative options captured by Or nodes in an activation diagram. Furthermore, deriving a non-blocking consistency-preserving rule from an activation diagram is a complex task as one needs to find a set of non-contradictory choices (not selecting the ElementCreation and the ElementDeletion node for the same element). Because of this, we follow a more sophisticated approach. Instead of constructing an ordinary rule, we first construct a VB rule, referred to as *consistency-preserving variability-based extension*, from the given rule and activation diagram. Those VB rule's cross-tree constraints Q and presence conditions capture, as propositional formulas, the different ways in which the given rule can be amended to become consistency-preserving. Then, off-the-shelf SAT solvers can be used to

compute instantiations of the VB rule, to which we refer as non-blocking consistency-preserving variants. To recapitulate our workflow (compare also Fig. 1):

1. For a given category $\mathbf{Sub}(T \supseteq P)$ and set of constraints $\mathcal{C}$, we compute the activation diagram $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$.
2. Given some ordinary rule $p = L \supseteq K \subseteq R$ in $\mathbf{Sub}(T \supseteq P)$, whose application might introduce violations of some of the constraints from $\mathcal{C}$, we use the activation diagram $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ to read off the consistency-preserving variability-based extension of p. This is a VB rule capturing the different ways in which the actions of the originally given rule p can be amended in such a way that upon application consistency with regard to the constraints of $\mathcal{C}$ is preserved.
3. The resulting consistency-preserving variability-based extension can be instantiated (see Definition 6) to obtain non-blocking consistency-preserving variants of p. These are effect-oriented rules, each of which captures a different way in which consistency-preservation can be achieved. In this way, we cater for the fact that a diverse set of possibilities for consistency-preservation might exist.
4. Upon application of a non-blocking consistency-preserving variant of p, its induced rule is computed according to the graph to which it is applied (see Definition 8). This caters for the fact that for each graph $G \in \mathbf{Sub}(T \supseteq P)$ to which the non-blocking consistency-preserving variant is applied, some of the additionally performed actions (compared to p) might be unnecessary because G already contains certain elements to be created or not contains certain elements to be deleted.

In this section, we detail the construction of consistency-preserving variability-based extensions of rules and their non-blocking consistency-preserving variants, prove the correctness of our approach (i.e., that non-blocking consistency-preserving variants are actually consistency-preserving), and prove certain completeness results (that, for an instantiable combination of rule actions and constraints, we obtain a consistency-preserving variability-based extension that is instantiable and that, for certain kinds of rules, we can simulate application traces of original rules leading to valid instances via our non-blocking consistency-preserving variants).

We start by constructing a consistency-preserving variability-based extension from a given rule and an activation diagram. The elements of the *feature model* of the consistency-preserving variability-based extension correspond to the Or nodes of the activation diagram and their outgoing edges (i.e., the alternative ways they offer to preserve consistency). The *cross-tree constraint* Q of the feature model serves to (i) encode paths, i.e., serves to ensure that when one alternative is chosen for an Or node, also the consequences of that choice are dealt with, and (ii) avoid contradictory choices, i.e., serves to prevent that for the same element the corresponding ElementCreation and ElementDeletion node are instantiated in the same variant. By definition, the *presence condition* of a VB rule declares the conditions under which the available elements are present in a specific instance of the VB rule. In our concrete application, the construction of this presence condition is the crucial but also most complex step because there are different reasons for elements to be present. Most basically, (i) an element can be present in a consistency-preserving variability-based extension because it is present in the originally given rule that we want to amend. Centrally for our approach, (ii) an element can be present in a consistency-preserving variability-based extension to preserve consistency with regard to the given constraints. And, finally, (iii) an element can be present in a consistency-preserving variability-based extension to complete it, i.e., to ensure that the rule actually consists of graphs. Concretely, when a rule prescribes to delete an edge, its incident nodes also have to be present in the LHS (and possibly also the interface) of the rule. To address all three cases, we construct our presence conditions as disjunctions $\varphi_{\mathrm{orig}} \vee \varphi_{\mathrm{pres}} \vee \varphi_{\mathrm{graph}}$, where φ_{orig} encodes the presence of an element because of its presence in the original rule, φ_{pres} encodes the presence of an element to preserve consistency, and φ_{graph} encodes the presence of an element to complete a graph. In the work of Horcas et al. [10], only case (ii) had to be considered because their rules exclusively work on nodes and their given rules just create or delete one node.

Construction 2 (Consistency-preserving variability-based extension of rule)**.** Let a type graph TG, graphs $P \subseteq T$ typed over it, a finite set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ in normal form according to Assumption 2,

and a rule $p = (L \supseteq K \subseteq R)$ in $\mathbf{Sub}(T \supseteq P)$ be given. Let $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ be the activation diagram constructed from T, P and $\mathcal{C}$. The *consistency-preserving variability-based extension of* p, denoted as

$$vp_{\mathcal{C}} = \left(p_{\mathcal{C}} = (L_{\mathcal{C}} \supseteq K_{\mathcal{C}} \subseteq R_{\mathcal{C}}), fm_{\mathcal{C}}, b_{\mathcal{C}}^{L}, b_{\mathcal{C}}^{K}, b_{\mathcal{C}}^{R}\right)$$

is a VB rule that is constructed as follows:

- *Construction of* $p_{\mathcal{C}}$*:* Let $L^- \subseteq T$ be those elements of T whose corresponding ElementDeletion nodes in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ stem from L or are (transitively) reachable in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ from an ElementDeletion node that corresponds to an element from $L \setminus K$ or from an ElementCreation node that corresponds to an element from $R \setminus K$. Let $\overline{L^-}$ be the set that consists of all nodes of $T \setminus L$ that are adjacent to an edge $e \in L^-$. Let $R^+ \subseteq T$ be those elements of T whose corresponding ElementCreation nodes in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ stem from R or are (transitively) reachable in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ from an ElementDeletion node that corresponds to an element from $L \setminus K$ or from an ElementCreation node that corresponds to an element from $R \setminus K$. With this notation, the rule $p_{\mathcal{C}}$ is defined as $L_{\mathcal{C}} \coloneqq L^- \cup \overline{L^-}$, $K_{\mathcal{C}} \coloneqq K$, and $R_{\mathcal{C}} \coloneqq R^+$.

- *Construction of* $fm_{\mathcal{C}}$ *(literals):* The feature model $fm_{\mathcal{C}}$ has as set of literals a root feature, one literal O_i per Or-node of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ and, for each such node, one literal o_j^i per outgoing edge in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$, i.e., a set

  $$\mathcal{F} \coloneqq \{root, O_1, o_1^1, \ldots, o_{w_1}^1, \ldots, O_v, o_1^v, \ldots, o_{w_v}^v\},$$

  where v denotes the number of Or nodes and w_j the respective number of outgoing edges (for $j = 1, \ldots, v$). The parent–child relationship is defined via $root < x$ for every other literal x and $O_i < o_j^i$ for all $i = 1, \ldots, v$ and $j = 1, \ldots, w_v$. All literals O_i that correspond to Or nodes are optional children of *root* and constitute Xor groups in case they have more than one child; otherwise, its single child is mandatory.

  We define the cross-tree constraints Q of $fm_{\mathcal{C}}$ after we have further defined the rule.

- *Construction of presence conditions:* As explained above, we compute the presence conditions of elements in three steps, i.e., for $X \in \{L_{\mathcal{C}}, K_{\mathcal{C}}, R_{\mathcal{C}}\}$, we set

  $$b_{\mathcal{C}}^{X} \coloneqq \varphi_{\mathrm{pres}}^{X} \vee \varphi_{\mathrm{orig}}^{X} \vee \varphi_{\mathrm{graph}}^{X},$$

  where the disjuncts $\varphi_{\mathrm{pres}}^{X}, \varphi_{\mathrm{orig}}^{X}$ and $\varphi_{\mathrm{graph}}^{X}$ are defined separately. The disjuncts φ_{pres} capture that an element has to be deleted resp. created to preserve the validity of constraints when applying the rule. Furthermore, we need to ensure that together with an edge that is to be deleted, its adjacent nodes are at least matched; the disjuncts φ_{graph} serve that purpose. The disjuncts φ_{orig} just encode that an element already belonged to the originally given rule p.

  Definition of $\varphi_{\mathrm{pres}}^{X}$*:* For $K_{\mathcal{C}}$, we set $\varphi_{\mathrm{pres}}^{K_{\mathcal{C}}} \coloneqq \mathsf{false}$ (which is equivalent to $\neg root$). Likewise, for elements $x \in L \setminus K$ (resp. $x \in R$), we set $\varphi_{\mathrm{pres}}^{L_{\mathcal{C}}}(x) \coloneqq \mathsf{false}$ (resp. $\varphi_{\mathrm{pres}}^{R_{\mathcal{C}}}(x) \coloneqq \mathsf{false}$).

  For elements $x \in L_{\mathcal{C}} \setminus (L \setminus K)$ (resp. $x \in R_{\mathcal{C}} \setminus R$), $\varphi_{\mathrm{pres}}^{L_{\mathcal{C}}}$ and $\varphi_{\mathrm{pres}}^{R_{\mathcal{C}}}$ are constructed in the same way, namely as follows: For every such element x, the condition $\varphi_{\mathrm{pres}}^{L_{\mathcal{C}}}(x)$ (resp. $\varphi_{\mathrm{pres}}^{R_{\mathcal{C}}}(x)$) is a disjunction with the disjuncts being the annotations of all incoming edges of its corresponding ElementDeletion node (resp. ElementCreation node). In that, edges that start at And- or Or-nodes are annotated as explained in Definition 10, whereas an edge that starts from an ElementDecision-node is annotated with that node's annotation (i.e., the disjunction of the annotations of that node's incoming edges). (Note that this definition is not cyclic because Definition 10 only introduces incoming edges for ElementDecision nodes that start at Or and And nodes and in Construction 1, 2. and 3., we only introduce edges between ElementDecision nodes of adjacent node and edge and only do so in one direction.)

  Definition of $\varphi_{\mathrm{graph}}^{X}$*:* For $X \in \{K_{\mathcal{C}}, R_{\mathcal{C}}\}$, we set $\varphi_{\mathrm{graph}}^{X}(x) \coloneqq \mathsf{false}$; for $L_{\mathcal{C}}$, for all elements x that do not stem from $\overline{L^-}$ we set $\varphi_{\mathrm{graph}}^{L_{\mathcal{C}}}(x) \coloneqq \mathsf{false}$. For a node $n \in \overline{L^-}$ we set

  $$\varphi_{\mathrm{graph}}^{L_{\mathcal{C}}}(n) \coloneqq \neg \varphi_{\mathrm{pres}}^{L_{\mathcal{C}}}(n) \wedge \bigvee_{\substack{e \in T \\ e \text{ adjacent to } n}} \varphi_{\mathrm{pres}}^{L_{\mathcal{C}}}(e).$$

This condition evaluates to true if a node is not to be deleted to preserve validity but one of its adjacent edges is.

Definition of φ^X_{orig}: We set $\varphi^{X_\mathcal{C}}_{\text{orig}}(x) = \mathsf{true} \iff x \in X$ (which is equivalent to *root*) and false otherwise for $X \in \{L, K, R\}$, i.e., $\varphi^{X_\mathcal{C}}_{\text{orig}}$ just encodes membership in the originally given rule p.

- *Construction of $fm_\mathcal{C}$ (cross-tree constraints):* The cross-tree constraint is a conjunction $\psi_{\text{path}} \wedge \psi_{\text{cons}}$, where ψ_{path} encodes paths in the given activation diagram and ψ_{cons} prevents inconsistent choices of features, i.e., choices that lead to the same element to be simultaneously created and deleted by a rule.

  Concretely, ψ_{path} is a conjunction where for every alternative o^i_j that any Or node in the activation diagram offers, we add as conjunct $o^i_j \implies O_{k_1} \wedge \cdots \wedge O_{k_r}$, where $O_{k_1} \wedge \cdots \wedge O_{k_r}$ are the Or nodes in $\text{AD}_{T \supseteq P,\mathcal{C}}$ that o^i_j directly triggers (see Definition 13). Moreover, we add the same kind of constraint for the element *root*, where *root* spans all actions of the original rule, i.e., a conjunct $root \implies O_{k_1} \wedge \cdots \wedge O_{k_t}$, where $O_{k_1}, \ldots, O_{k_t}$ ranges over the Or nodes that are triggered by edges leaving ElementDecision nodes corresponding to a node x from $L \setminus K$ or $R \setminus K$. In the other direction, for every Or node O_j *that is not directly triggered by some ElementDecision node corresponding to one of the actions of the original rule* we add as a conjunct $O_j \implies o^{i_1}_{j_1} \vee \cdots \vee o^{i_s}_{j_s}$, where $o^{i_1}_{j_1}, \ldots, o^{i_s}_{j_s}$ are those edges that directly trigger O_j.

  Finally, ψ_{cons} is defined via

$$\psi_{\text{cons}} \coloneqq \bigwedge_{x \in (L_\mathcal{C} \cap R_\mathcal{C})} \Big(\big(\varphi^{L_\mathcal{C}}_{\text{pres}}(x) \implies \neg(\varphi^{R_\mathcal{C}}_{\text{orig}}(x) \vee \varphi^{R_\mathcal{C}}_{\text{pres}}(x))\big) \wedge \big(\varphi^{R_\mathcal{C}}_{\text{pres}}(x) \implies \neg((\varphi^{L_\mathcal{C}}_{\text{orig}}(x) \wedge \neg\varphi^{K_\mathcal{C}}_{\text{orig}}(x)) \vee \varphi^{L_\mathcal{C}}_{\text{pres}}(x))\big)\Big); \tag{3}$$

  i.e., we require (i) that an element that needs to be deleted to preserve consistency is not simultaneously created/required by the original rule or created to preserve consistency and (ii) that an element that needs to be created to preserve consistency is not simultaneously deleted by the original rule or deleted to preserve consistency. ♢

Example 13. We illustrate our construction of consistency-preserving variability-based extensions using rule *AddM1ToNewC2* (see Fig. 6a) as our starting point. The snippet of $\text{AD}_{T \supseteq P,\mathcal{C}}$ in Fig. 28 (compare Example 10) shows the relevant consequences for the ElementCreation nodes +C2 and +eC2M1. In the following, we use this to illustrate every step of our construction of the consistency-preserving variability-based extension $vp_\mathcal{C} = \big(p_\mathcal{C} = (L_\mathcal{C} \supseteq K_\mathcal{C} \subseteq R_\mathcal{C}), fm_\mathcal{C}, b^L_\mathcal{C}, b^K_\mathcal{C}, b^R_\mathcal{C}\big)$ of *AddM1ToNewC2*.

Construction of rule $p_\mathcal{C}$: For computation of the rule $p_\mathcal{C}$, we have to explore the ElementCreation and ElementDeletion nodes that are reachable from ElementDecision nodes that correspond to actions of the given rule. Thus, in our case, we have to explore the elements reachable from +C2 and +eC2M1. The reachable ElementDeletion nodes are the nodes -eCjM1 and -eCjA1, where $j = 1, \ldots, 6$, i.e., the nodes representing the unassignments of A1 and M1 for every available Class Cj. Therefore, these edges, together with the node M1 which already belongs to L, constitute the set L^-; the set $\overline{L^-}$ contains all Classes and the node A1. Similarly, the set R^+ contains all edges between any Class Cj and M1 or A1, all Classes Cj and A1. The resulting rule is shown in Fig. 35.

Literals of feature model $fm_\mathcal{C}$: The literals of the feature model $fm_\mathcal{C}$ result from the Or nodes of $\text{AD}_{T \supseteq P,\mathcal{C}}$; note that, because we only display a snippet, not all such nodes are present in Fig. 28. For instance, $fm_\mathcal{C}$ contains a feature O_4, which is a direct child of *root*, and constitutes a Xor group with children $o^4_1, \ldots, o^4_5$. The snippet of the feature model that arises from Fig. 28 is shown in Fig. 36.

Presence conditions: By design of the rule *AddM1ToNewC2*, we have that $\varphi^{L_\mathcal{C}}_{\text{orig}}$ and $\varphi^{K_\mathcal{C}}_{\text{orig}}$ evaluate to true for M1 and false for every other element of T; $\varphi^{R_\mathcal{C}}_{\text{orig}}$ additionally evaluates to true for C2 and the encapsulates-edge between C2 and M1.

More interestingly, $\varphi^{L_\mathcal{C}}_{\text{pres}}$ and $\varphi^{R_\mathcal{C}}_{\text{pres}}$ are constructed from the annotations of incoming edges in $\text{AD}_{T \supseteq P,\mathcal{C}}$. For instance $\varphi^{L_\mathcal{C}}_{\text{pres}}(\mathsf{eC1M1}) = o^1_1 \vee \ldots$, where the dots indicate annotations of incoming edges omitted in our

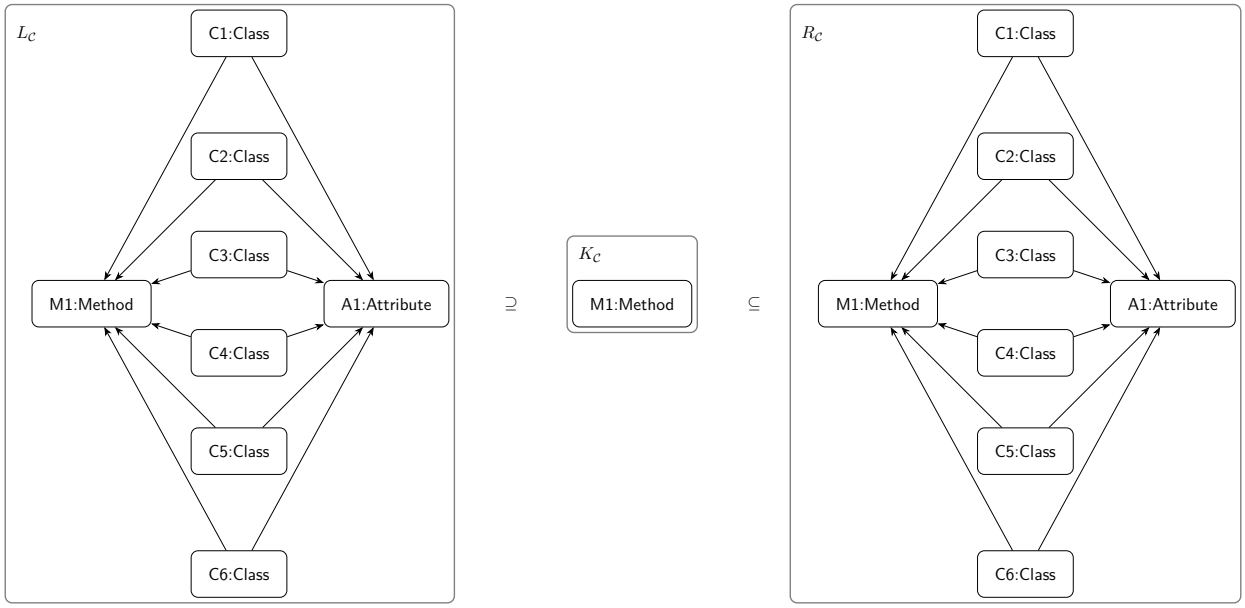


Figure 35: Rule part of the consistency-preserving variability-based extension resulting in our example

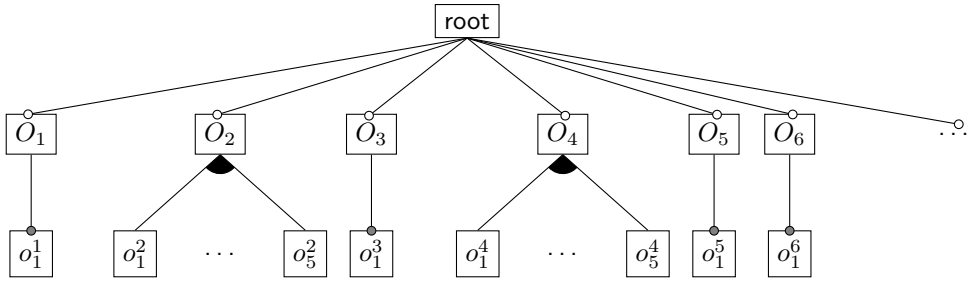


Figure 36: Snippet of the feature model of the consistency-preserving variability-based extension resulting in our example

snippet. (Every ElementCreation node +eCjM1, where $j \neq 1$, has an Or node $\mathsf{OR_k}$ with a single outgoing edge or_1^k referencing -eC1M1.) Similarly, $\varphi^{R_\mathcal{C}}_{\text{pres}}(eC6M1) = o_5^2 \vee \dots$ and $\varphi^{R_\mathcal{C}}_{\text{pres}}(C6) = o_5^2 \vee o_5^4 \vee \dots$. Note that $\varphi^{R_\mathcal{C}}_{\text{pres}}(eC2M1) = \varphi^{R_\mathcal{C}}_{\text{pres}}(C2) = \mathsf{false}$ because these elements already belong to the RHS R of our original rule *AddM1ToNewC2*.

Finally, by definition, φ^X_{graph} evaluates to false for $X = K_\mathcal{C}, R_\mathcal{C}$, and also for $X = L_\mathcal{C}$, except for nodes from $\overline{L^-}$. As seen above, here the set $\overline{L^-}$ contains all Classes and the node A1. One can also check that $\varphi^{L_\mathcal{C}}_{\text{pres}}(n) = \mathsf{false}$ for every $n \in \overline{L^-}$. Thus, in this example we have $\varphi^{L_\mathcal{C}}_{\text{graph}}(n) = \mathsf{true} \wedge \bigvee_{e \text{ admissible}} \varphi^{L_\mathcal{C}}_{\text{pres}}(e)$, where the admissible edges are all edges from T that are adjacent to n. For instance, we get $\varphi^{L_\mathcal{C}}_{\text{graph}}(\mathsf{C1}) = \varphi^{L_\mathcal{C}}_{\text{pres}}(\mathsf{eC1M1}) \vee \varphi^{L_\mathcal{C}}_{\text{pres}}(\mathsf{eC1A1}) \vee \dots = o_1^1 \vee o_1^3 \vee \dots$.

Cross-tree constraints: From the snippet of $\text{AD}_{T \supseteq P,\mathcal{C}}$ displayed in Fig. 28 we obtain, for instance, $o_1^1 \implies O_2 \wedge O_3$, *root* $\implies O_1 \wedge \dots$ and $O_4 \implies o_1^3$ as conjuncts of ψ_{path}.

Regarding ψ_{cons}, the edge assigning Method M1 to Class C2 is an example for an element appearing in $L_\mathcal{C} \cap R_\mathcal{C}$ (and where both corresponding ElementDecision nodes are displayed in our snippet of $\text{AD}_{T \supseteq P,\mathcal{C}}$). We have $\varphi^{L_\mathcal{C}}_{\text{pres}}(\mathsf{eC2M1}) = o_1^6 \vee \dots$, $\varphi^{R_\mathcal{C}}_{\text{orig}}(\mathsf{eC2M1}) = \mathsf{true}$ and $\varphi^{R_\mathcal{C}}_{\text{pres}}(\mathsf{eC2M1}) = \varphi^{L_\mathcal{C}}_{\text{orig}}(\mathsf{eC2M1}) = \varphi^{K_\mathcal{C}}_{\text{orig}}(\mathsf{eC2M1}) = \mathsf{false}$. Thus, instantiating Eq. 3 with these values results in the conjunct

$$(o_1^6 \vee \dots \implies \mathsf{false}) \wedge (\mathsf{false} \implies \neg o_1^6 \vee \dots).$$

This effectively excludes o_1^6 of ever being selected, which, in turn (via the path constraints), excludes o_2^2 of ever being selected (and so forth). ♢

From the consistency-preserving variability-based extensions of rules that we just constructed, we can instantiate variants as introduced in Definition 6. We interpret such a variant as an effect-oriented rule by letting the originally given rule be the base rule, an induced variant be the maximal rule, and the context extension be given by those nodes that were just added to the LHS of the induced variant such that it constitutes a graph.

Definition 14 (Non-blocking consistency-preserving variant)**.** Let a type graph TG, graphs $P \subseteq T$ typed over it, a finite set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ in normal form according to Assumption 2, and a rule $p = (L \supseteq K \subseteq R)$ in $\mathbf{Sub}(T \supseteq P)$ be given. Let $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ be the activation diagram constructed from T and $\mathcal{C}$ and let $vp_{\mathcal{C}}$ be the consistency-preserving variability-based extension of p. Then, a *non-blocking consistency-preserving variant of* p is an effect-oriented rule, denoted as

$$vp^F_{\mathcal{C},\mathrm{e}} \coloneqq \left(p, p^F_{\mathcal{C},\mathrm{m}} = (L^F_{\mathcal{C},\mathrm{m}} \supseteq K^F_{\mathcal{C},\mathrm{m}} \subseteq R^F_{\mathcal{C},\mathrm{m}}), k^F_{\mathrm{ext}}\right)$$

that has been obtained as follows: The rule p is the originally given rule, the set $F \subseteq \mathcal{F}$ is a valid configuration of $fm_{\mathcal{C}}$ and $p^F_{\mathcal{C},\mathrm{m}}$ its induced variant, and $k^F_{\mathrm{ext}} \colon K^F_{\mathcal{C},\mathrm{m}} \to K^F_{\mathrm{m}}$, where K^F_{m} is the union of $K^F_{\mathcal{C},\mathrm{m}}$ with those elements of $\overline{L^-}$ for which $\varphi^{L_{\mathcal{C}}}_{\mathrm{graph}}$ evaluates to true but which are not created by the original rule p, i.e., those elements $n \in \overline{L^-}$ for which

$$\chi_F \models \varphi^{L_{\mathcal{C}}}_{\mathrm{graph}}(n) \wedge \neg \varphi^{R_{\mathcal{C}}}_{\mathrm{orig}}. \tag{4}$$

♢

Example 14. In our running example of deriving a consistency-preserving variant for *AddM1ToNewC2*, there is only one valid configuration of the feature model of the consistency-preserving variability-based extension of that rule: First, we have to select *root* by definition of configurations. The path constraints then force us to select O_1, O_2, O_3, O_4 (and the features corresponding to the Or nodes omitted in Fig. 28 that trigger the deletion of the edges assigning M1 to other Classes than C2). Regarding O_1 and O_3, we have to select their only children o_1^1 and o_1^3, respectively. Regarding O_2, the cross-tree constraints ensure that o_1^2 is the only child that can be selected (because every other child would trigger the deletion of the edge eC2M1). Similarly, o_1^4 is the only selectable child of O_4 because every other choice would first trigger the assignment of M1 to another Class besides C2 and consequently, by path constraints, the deletion of eC2M1, which is excluded by ψ_{cons}. Moreover, the choice of o_1^4 necessitates choices that trigger the deletion of all edges assigning A1 to other Classes than C2 (omitted in Fig. 28).

The overall resulting rule is shown in Fig. 37. The rule is applicable if Class C2 does not exist; in this case, it creates this Class and assigns Method M1 to it. Moreover, it also assigns Attribute A1 to C2 and removes M1 and A1 from every other Class they might be assigned to. For this, these Classes are part of the context extension of the rule, ensuring that such a Class is only part of the induced rule if an outgoing edge exists that has to be deleted. In this way, it is ensured that the rule remains applicable to graphs in which some of these Classes do not exist. ♢

After defining consistency-preserving variability-based extension of rules and their non-blocking consistency-preserving variants formally, the rest of this section is concerned with proving the properties of these kinds of rules. In particular, we prove that (i) non-blocking consistency-preserving variants indeed constitute rules; (ii) non-blocking consistency-preserving variants extend the originally given rule in the sense that they are always applicable when the original rule is and perform at least the actions performed by the original rule; (iii) non-blocking consistency-preserving variants indeed preserve consistency; (iv) the consistency-preserving variability-based extension of a rule is complete in the sense that whenever this rule specifies a coherent set of actions, its consistency-preserving variability-based extension has a non-empty set of instances; and (v) under appropriate circumstances (namely given atomic rules that only create or delete a single element), the consistency-preserving variability-based extensions of a given set of rules are even complete in the sense that their non-blocking consistency-preserving variants can simulate all transformation sequences via the originally given rules that end in valid graphs.

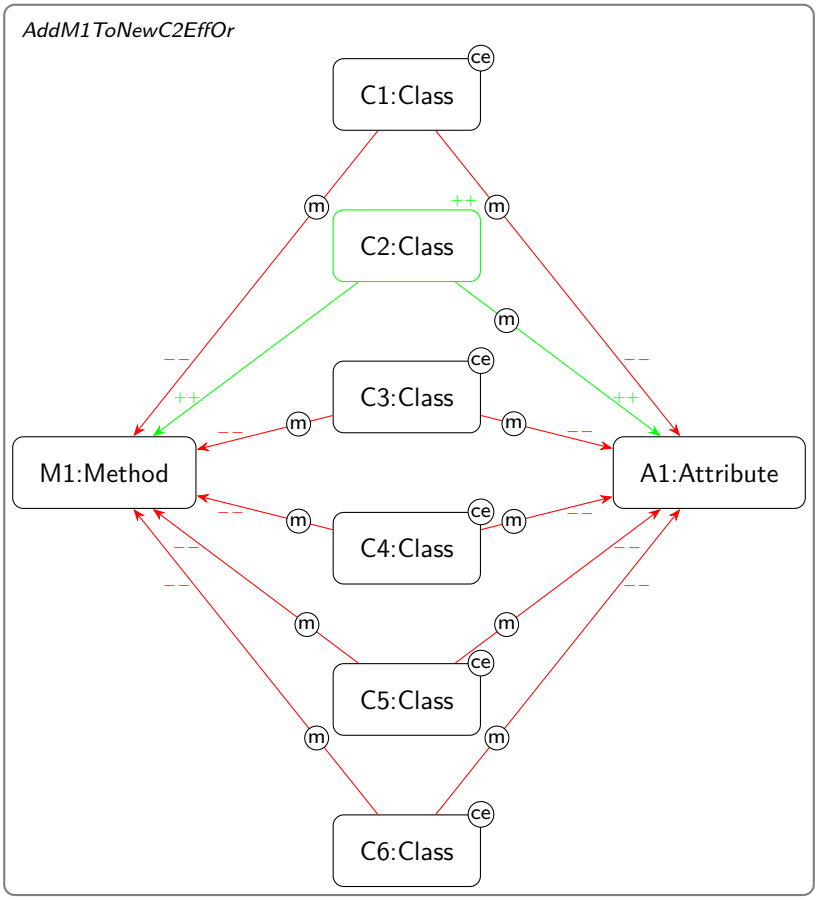


Figure 37: Non-blocking consistency-preserving variant for *AddM1ToNewC2*

Proposition 6 (Consistency-preserving rules are effect-oriented rules)**.** *Let a type graph TG, graphs $P \subseteq T$ typed over it, a finite set $\mathcal{C}$ of constraints in* $\mathbf{Sub}(T \supseteq P)$ *in normal form according to Assumption 2, and a rule $p = (L \supseteq K \subseteq R)$ in* $\mathbf{Sub}(T \supseteq P)$ *be given. Let $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ be the activation diagram constructed from T and $\mathcal{C}$, let $vp_{\mathcal{C}}$ be the consistency-preserving variability-based extension of p and $vp^F_{\mathcal{C},\mathrm{e}}$ a consistency-preserving variant of p, induced by some valid configuration F. Then, $vp^F_{\mathcal{C},\mathrm{e}}$ is an effect-oriented rule in the sense of Definition 7.* ♢

Having ascertained that our construction actually produces effect-oriented rules, we next clarify that the induced consistency-preserving variants of a rule still have the same matches as their original rule, i.e., they are indeed *non-blocking*. Moreover, their applications do not alter the problem graph P but only affect elements from $T \setminus P$.

Lemma 7 (Applicability of consistency-preserving rules)**.** *Let a type graph TG, graphs $P \subseteq T$ typed over it, a finite set $\mathcal{C}$ of constraints in* $\mathbf{Sub}(T \supseteq P)$ *in normal form according to Assumption 2, a rule $p = (L \supseteq K \subseteq R)$ in* $\mathbf{Sub}(T \supseteq P)$*, and some match $m\colon L \subseteq G$ for it be given. Then, any consistency-preserving variant $vp^F_{\mathcal{C},\mathrm{e}}$ of p is applicable to G, i.e., there is an effect-oriented transformation (in the sense of Definition 8) $t_{\mathrm{ind}}\colon G \Longrightarrow_{vp^F_{\mathcal{C},\mathrm{e}}} H$ via the induced rule $p^{F,G}_{\mathcal{C},\mathrm{ind}} = \left(L^{F,G}_{\mathcal{C},\mathrm{ind}} \supseteq K^{F,G}_{\mathcal{C},\mathrm{ind}} \subseteq R^{F,G}_{\mathcal{C},\mathrm{ind}}\right)$.*

Moreover, it holds that $L^{F,G}_{\mathcal{C},\mathrm{ind}} \setminus K^{F,G}_{\mathcal{C},\mathrm{ind}} \subseteq (T \setminus P)$ and $R^{F,G}_{\mathcal{C},\mathrm{ind}} \setminus K^{F,G}_{\mathcal{C},\mathrm{ind}} \subseteq (T \setminus P)$, i.e., the transformation t_{ind} only creates and deletes elements from $T \setminus P$, respectively (compare Assumption 2). Furthermore, for any $x \in L^{F,G}_{\mathcal{C},\mathrm{ind}}$ for which $\chi_F \models \varphi^{Lc}_{\mathrm{graph}}(x)$ (i.e., for any node that is just part of the LHS to complete it to

constitute a graph), it holds that the induced rule does not delete that element, i.e., $x \in K^{F,G}_{\mathcal{C},\text{ind}}$. ♢

The following theorem is our central result, showing that applications of our constructed consistency-preserving variants of a rule p transform consistent graphs (satisfying all given constraints) into consistent graphs; i.e., these rules are indeed *consistency-preserving*.

Theorem 8 (Correctness of consistency-preserving rules)**.** *Let a type graph TG, a graph T typed over it, a finite set $\mathcal{C}$ of constraints in* $\mathbf{Sub}(T \supseteq P)$ *in normal form according to Assumption 2, and a rule* $p = (L \supseteq K \subseteq R)$ *in* $\mathbf{Sub}(T \supseteq P)$ *be given. Let* $\text{AD}_{T \supseteq P,\mathcal{C}}$ *be the activation diagram constructed from T and $\mathcal{C}$, let $vp_{\mathcal{C}}$ be the consistency-preserving variability-based extension of p and let $vp^F_{\mathcal{C},\text{e}}$ be any consistency-preserving variant of p, induced by some valid configuration F of $fm_{\mathcal{C}}$.*

Then, for every graph $G \in \mathbf{Sub}(T \supseteq P)$ *such that* $G \models \mathcal{C}$*, if there is an effect-oriented transformation* $t\colon G \Longrightarrow_{vp^F_{\mathcal{C},\text{e}}} H$*, it also holds that* $H \models \mathcal{C}$. ♢

Remark 11. The consistency-preserving rules that we construct even enjoy a somewhat stronger property: They actually guarantee consistency with regard to every constraint with which they interact upon application, i.e., with every constraint that positively quantifies an element that the rule deletes or that negatively quantifies an element that the rule creates. For such actions, potential remedies are captured by the activation diagram and thus performed by our constructed rules. Thus, applications of consistency-preserving rules may also repair formerly violated constraints.

Conversely, this is why we need to assume a consistent graph G, i.e., a graph G such that $G \models \mathcal{C}$, in the statement of the above theorem. Given a graph G, a constraint c such that $G \not\models c$ and a rule p that principally cannot render c invalid (because it neither creates elements forbidden by c nor deletes elements required by c), then for consistency-preserving variants $vp^F_{\mathcal{C},\text{e}}$ of p applicable to G there is no guarantee that they transform G into a graph satisfying c. ♢

The above theorem clarifies that every application of a consistency-preserving rule is indeed consistency-preserving; however, existence of such rules remains unclear. By definition, given a rule p, the number of consistency-preserving variants we can derive for p corresponds to the number of valid configurations F of the feature model $fm_{\mathcal{C}}$ of the consistency-preserving variability-based extension $\big(p_{\mathcal{C}} = (L_{\mathcal{C}} \supseteq K_{\mathcal{C}} \subseteq R_{\mathcal{C}}), fm_{\mathcal{C}}, b^L_{\mathcal{C}}, b^K_{\mathcal{C}}, b^R_{\mathcal{C}}\big)$ of p. While we cannot exclude that different valid configurations induce the same consistency-preserving rule, existence of a valid configuration guarantees existence of a consistency-preserving rule.

It is easy to see that such valid configurations do not need to exist; however, this only happens if the given constraints are conflicting or if the actions of the given rule are such that a certain constraint cannot be valid after these have been performed. For instance, given a positive constraint $c = \exists\, C_1$ and a rule p that deletes an element x of C_1 it is clear after every transformation $G \Longrightarrow_p H$ that $H \not\models c$. Thus, to render p validity-preserving, we would have to revert the deletion of x. We consciously exclude this possibility by our cross-tree constraints to ensure that the action of the given rule remains preserved.

In the next proposition, we give a sufficient criterion for the existence of a validity-preserving rules: The given set of constraints is satisfiable and compatible with the actions of the given rule. To express this formally, we first express the actions of a rule as constraints (that express that the elements that the rule deletes do not exist but that the ones it creates exist).

Definition 15 (Rule constraint)**.** Given a rule $p = L \supseteq K \subseteq R$ in $\mathbf{Sub}(T \supseteq P)$, its *rule constraint* c_p is the constraint

$$\exists\, R \wedge c_L, \tag{5}$$

where

$$c_L = \neg \left(\bigvee_{x \in L \setminus K} \exists\, \hat{x} \right), \tag{6}$$

where, in case x is a node, $\hat{x} = x$, and, in case x is an edge, $\hat{x}$ is the graph containing also x's adjacent edges. ♢

Theorem 9 (Existence of consistency-preserving rules)**.** *Let a type graph TG, graphs $P \subseteq T$ typed over it, a finite set $\mathcal{C}$ of constraints in* $\mathbf{Sub}(T \supseteq P)$ *in normal form according to Assumption 2, and a rule $p = (L \supseteq K \subseteq R)$ in* $\mathbf{Sub}(T \supseteq P)$ *be given. Let $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ be the activation diagram constructed from T and $\mathcal{C}$ and let $vp_{\mathcal{C}}$ be the consistency-preserving variability-based extension of p.*

If there exists some graph $E \in \mathbf{Sub}(T \supseteq P)$ such that $E \models \mathcal{C} \wedge c_p$, where c_p is the rule constraint of p, then there exists some valid configuration F of $fm_{\mathcal{C}}$; in particular, there exists a consistency-preserving variant $vp^F_{\mathcal{C},\mathrm{e}}$ of p. Moreover, this variant can be constructed in such a way that $E \models c_{vp^F_{\mathcal{C},\mathrm{e}}}$, where $c_{vp^F_{\mathcal{C},\mathrm{e}}}$ is the rule constraint of $vp^F_{\mathcal{C},\mathrm{e}}$. ♢

Remark 12. While the above result provides guarantees under which a consistency-preserving variability-based extension of some rule can be instantiated, it does not provide any bound on the number of existing variants of such a rule (comparable to Proposition 4 where we provide such bounds for the size of activation diagrams). Trivially, the number of rules is in $\mathcal{O}(2^{|T \setminus P|})$, because the graphs constituting the rules are determined by subsets of $T \setminus P$.

In Example 14, quite favourably, there was a single variant of our constructed consistency-preserving variability-based extension of *AddM1ToNewC2*. This, however, is due to the fact that the constraints of the CRA problem heavily constrain the structure of valid solutions (every Feature is assigned to exactly one Class and Attributes are assigned together with Methods depending on them).

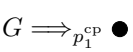
Generally, the problem for counting the number of variants more precisely and, in particular, guaranteeing a low upper bound, is that this number primarily rests on the number of satisfiable assignments of the presence conditions of the consistency-preserving variability-based extension. Counting the number of satisfying assignments to a formula from propositional logic is a well-known $\#P$ hard problem [44], and we did not observe any structure that would make the problem tractable in our special case.

While this means that the number of instantiable rules might grow exponentially, this does not need to pose a problem in practice. First of all, note that we derive exactly one VB rule per originally given rule; typically, the number of given rules can be restricted to $2|T \setminus P|$ (having for each element one rule creating and one rule deleting it). *This means, the exponential growth, if present, is encoded into the VB rules, of which there is only a number linear in $|T \setminus P|$.* Then, it might not be necessary to compute all possible instantiations of each VB rules. In their application, for instance, Horcas et al. [10] just computed at most one consistency-preserving configuration operator per feature decision (and not all possible ones) and still obtained excellent results when solving feature configuration problems with these rule sets. We expect that computing a small number of rules, maybe recomputing additional ones one the fly when the need becomes somehow apparent, will suffice for many applications, at least for metaheuristic approaches to solving optimisation problems. ♢

Finally, we are interested in investigating in how far our consistency-preserving rules are able to *simulate* transformation sequences of their underlying original rules. That is, given a set of constraints $\mathcal{C}$ and a set of rules $\mathcal{R}$ in $\mathbf{Sub}(T \supseteq P)$ and a transformation sequence $G \Longrightarrow^*_{\mathcal{R}} H$ such that $H \models \mathcal{C}$, does there also exist a transformation sequence $G \Longrightarrow^*_{\mathcal{R}_{\mathrm{cp}}} H$, where $\mathcal{R}_{\mathrm{cp}}$ denotes the set of consistency-preserving rules derived from $\mathcal{R}$?

In general, unfortunately, this is not the case. The underlying cause for this is our design decision to construct non-blocking consistency-preserving rules in such a way that the actions of the underlying rule are preserved, i.e., it remains a subrule of the constructed rule (by combination of Propositions 6 and 2). To illustrate this phenomenon in a minimal setting, we do not use the CRA problem in the following example but use untyped graphs. In the example, nodes of graphs are drawn as black dots, indices of these dots are used as identifiers (also implicitly defining the inclusions) and we do not need to make any use of edges.

Example 15. Let $G = \bullet_1$ be a graph, $\mathcal{C} = \{\exists \bullet_1 \vee \exists \bullet_3\}$ a set consisting of a single constraint and $\mathcal{R} = \{p_1, p_2\}$ a set of two rules with $p_1 = (\bullet_1 \supseteq \emptyset \subseteq \bullet_2)$ and $p_2 = (\bullet_2 \supseteq \bullet_2 \subseteq \bullet_2 \bullet_3 \bullet_4)$, where T contains $\bullet_1 \bullet_2 \bullet_3 \bullet_4$ and $P = \emptyset$. Then, there is a transformation $G = \bullet_1 \Longrightarrow_{p_1} \bullet_2 \Longrightarrow_{p_2} \bullet_2 \bullet_3 \bullet_4 = H$ and $H \models \mathcal{C}$. Computing our consistency-preserving rules in this situation will result in $p_1^{\mathrm{cp}} = \bullet_1 \supseteq \emptyset \subseteq \bullet_2 \bullet_3$

and $p_2^{\text{cp}} = p_2$. Now, $G \Longrightarrow_{p_1^{\text{cp}}}$  $= H' \neq H$ but p_2^{cp} is not applicable to H' (because  already exists) and one also cannot derive H from G by switching the transformation order. ♢

The problem in the above example is that, to derive our desired result, we need to apply a rule whose actions have been partially pre-empted by an action that was already performed to preserve consistency. While a fine-grained analysis of this situation remains future work, it is easy to see that it is unproblematic as long as the actions performed by the originally given rules are fine-grained enough in the sense that they only create or delete a single element. We call such rules *atomic*.

Definition 16 (Atomic rule)**.** A transformation rule p in $\mathbf{Sub}(T \supseteq P)$ is *atomic* if it only creates or deletes a single element and has minimal context for that purpose. That is, an atomic rule that deletes a node $x \in T$ has x as LHS and empty RHS and interface; an atomic rule that deletes an edge $x \in T$ has x and its adjacent nodes as LHS and the adjacent nodes as RHS and interface. For creation, the situation is analogous. For $x \in T$, we denote its atomic creation rule as at_x^+ and its atomic deletion rule as at_x^-. ♢

Theorem 10 (Simulating transformation sequences)**.** *Let a type graph TG, a graph T typed over it, a finite set $\mathcal{C}$ of constraints in $\mathbf{Sub}(T \supseteq P)$ in normal form according to Assumption 2, and a set of rules $\mathcal{R}$ in $\mathbf{Sub}(T \supseteq P)$ be given. Let $\text{AD}_{T \supseteq P, \mathcal{C}}$ be the activation diagram constructed from T and $\mathcal{C}$ and let $\mathcal{R}_{\text{cp}}$ be the set of consistency-preserving variability-based extensions of rules $p \in \mathcal{R}$.*

Given a transformation sequence $G \Longrightarrow_{\mathcal{R}}^* H$ in $\mathbf{Sub}(T \supseteq P)$ such that $H \models \mathcal{C}$, if $\text{at}_x^- \in \mathcal{R}$ for all $x \in T \setminus H$ and $\text{at}_x^+ \in \mathcal{R}$ for all $x \in H \setminus G$, then there also exists a transformation sequence $G \Longrightarrow_{\mathcal{R}_{\text{cp}}}^* H$. ♢

Remark 13. While the restriction to atomic rules is a serious restriction in general, it is not for our desired application in model-driven optimisation, computing solutions to selection problems. In that setting, we can easily start with atomic rules, namely the set of rules that for each element of $T \setminus P$ contains a rule creating that element and one deleting it (in fact, this is also the set of rules that Horcas et al. [10] use as basis to derive their consistency-preserving configuration operators for efficient product line configuration). Then, the above theorem is a kind of *completeness result*, guaranteeing that from every feasible solution all other feasible solutions can be reached via our consistency-preserving variants. Here, the real restriction is rather the number of rules: The theorem is not guaranteed to hold if not all consistency-preserving variants are computed, which might not be possible for practical reasons (as already discussed above in Remark 12). ♢

6. Related Work

We begin our discussion of related work by considering some related notions of graph consistency, before discussing existing approaches for generating consistency-preserving (or even -improving) graph-transformation rules. Finally, we briefly touch on some related concepts from the SAT-solving literature.

6.1. Graph consistency

Our paper is concerned with producing graph-transformation rules that preserve the validity of a graph with respect to a set of constraints. We, therefore, briefly discuss related work discussing what it means for a graph to be consistent and how this affects desirable properties of graph transformations and transformation rules.

A large part of the literature considers graph consistency as a binary property: a graph G is either consistent with graph constraint c (there are no violations) or it is not. This leads to notions of c-preserving and c-guaranteeing transformations [11], which focus on the full validity of the result graphs.

However, in many practical applications we require a more graduated notion of consistency, where the number of locations in a graph where a constraint is violated is used to differentiate degrees of consistency. Following this perspective, we have previously [9] introduced the concepts of consistency-sustaining and improving transformations and rules, which are conceptually located properly in between c-preserving and -guaranteeing transformations. These new forms of transformations make the gradual improvements in consistency explicit. Work on partial graphs as in, e.g. [45], investigates the validity of constraints in families of graphs which is not our focus here and therefore, not further considered.

Important related work on graph repair (when consistency is viewed as a binary property) has been undertaken by Schneider et al. [46], who develop a correct and complete repair algorithm. In contrast to our work here, they repair individual graphs whereas we derive non-blocking consistency-preserving rules (which might be used for graph repair); moreover, they work in the category of attributed typed graphs whereas we work in the finite setting of a category of subgraphs $\mathbf{Sub}(T \supseteq P)$. However, (the structural part of) their repair approach is based on a tableau calculus, where branches represent different options to fulfil a constraint and, by that, capture information that is akin to the information we capture in our activation diagrams. A more detailed future comparison thus might help in extending our approach to the more general setting of a category of typed graphs.

6.2. Consistency-preserving rules

Different strategies have been explored for generating consistency-preserving rules given graph constraints. One of the easiest ways of ensuring that all applications of a graph transformation rule will be consistency-preserving is to incorporate them as suitable application conditions into the transformation rules [11]. The rules are only applicable if the result of the transformation satisfies the constraint. This approach starts with [8] for graphs and a restricted kind of constraints and has been lifted to general nested constraints (which are equivalent to a first-order logic in the case of graphs) in arbitrary weak adhesive categories [11]. The process can be fully automated and is implemented, integrating nested graph constraints, or constraints defined in OCL, as application condition into Henshin [37] transformation rules [16, 6, 7]. However, there are two weaknesses: First, rule applicability can be severely restricted by the constructed application conditions, potentially even making the rule entirely inapplicable in the worst case. Secondly, rules that preserve validity throughout can lack flexibility in some applications.

Another approach is to design the actions of rules in such a way that they cannot introduce violations of constraints. In [5] the authors provide a framework to check if a transformation rule can introduce a violation of a given constraint. If this is the case, the user is provided with an example for such a violation to adapt the rule accordingly. Going one step further towards automation, Burdusel et al. [13] present techniques for generating multiplicity-preserving rules from a given type graph; however, they do not provide any proofs of completeness or correctness. Similarly, in [4], the authors derive editing rules in such a way that they cannot introduce violations of multiplicity constraints. In [3], Kosiol et al. construct so-called interaction schemes from rules and constraints, such that the resulting rules are consistency-preserving or even -improving.

Our approach goes beyond the existing literature in that we can handle more general constraints (in particular, we are able to handle multiplicity constraints), though we are unique in the formalisation based on subgraph transformation systems. Kosiol et al. [3] perhaps comes closest to our work. Their work handles constraints of the form $\forall\,(P, \exists\, C)$, where $P \subseteq C$. The approach is not able to handle transitive effects (if creating C creates a new instance of P) but otherwise, P can be arbitrary. Our activation diagrams naturally address transitive effects, however our approach is restricted to cases where the first graph in an implication consists of a single node or edge (with its adjacent nodes). Hence, it may be possible to construct examples that [3] can handle and we cannot.

Somewhat orthogonally to these works, Ficsor et al. recently introduced an approach to "differentiate" graph predicates to derive propagation rules that can be used to refine partial models [47]. In their setting, they use a 4-valued logic to express that the existence of certain model elements and relations is as of yet unknown. From well-formedness constraints they derive propagation rules that reduce the uncertainty of partial models by fixing the (non-)existence of such elements or relations. Similar to our approach here, these rules are automatically derived from the constraints in such a way that (non-)existence of an element is ensured if otherwise the constraint would be violated. In contrast, they are not computed to augment actions of a given rule to render it consistency-preserving but rather to monitor a partial model and reduce uncertainty as soon as it is clear that certain decisions are unavoidable to achieve consistency with regard to a given constraint. Ficsor et al. also prove soundness and completeness of their approach. However, completeness in their setting does not mean that all well-formed models can be reached via their rules but rather that after an application of one of their rules, the resulting partial model can still be refined to any complete well-formed model to which the original partial model could have been refined (where refinement generally refers to the resolving of uncertainty).

In contrast to approaches that try to preserve consistency throughout transformation processes, other approaches allow for intermediate violations of constraints and restore consistency at some point—often referred to as 'model repair' in the literature. For example, in [2], rules that allow to repair arbitrary violations of multiplicity constraints are derived from a given meta-model. Habel and Sandmann [1] derive rules from nested graph constraints that allow to repair occurring violations of the constraints. Our approach targets consistency-preserving transformation rules. Rules that improve consistency are out of scope of our work at the moment.

6.3. Implication graphs in SAT solving

SAT solving is the problem of allocating truth values to a set of Boolean variables such that a given formula using these variables evaluates to true (or identifying that the given formula is not satisfiable; that is, that no such assignment exists).

While the overall SAT-solving problem is NP-complete [48], the so-called 2-SAT problem (where the disjunctions in a CNF-formulation of the problem are allowed only a maximum of two literals) can be solved in polynomial time. One early strategy here was to translate the problem into an implication graph [49], where nodes represent negative and positive assignments to each variable, and edges represent direct consequences. This is structurally similar to our activation diagrams, which also use nodes to encode positive and negative choices about the presence of individual elements of a graph. However, our activation diagrams provide a richer encoding as we also allow 'or'- and 'and'-nodes to encode more complex dependencies between individual choices. Critically, like Aspvall et al's implication graphs [49], we do not encode consequences of groups of choices; that is each edge only starts at a single literal.

General SAT solvers use implication graphs as part of a strategy called 'clause learning' [50]. These implication graphs, however, differ significantly from our activation diagrams [50, Definition 4]: while nodes still represent individual literals, edges are annotated with sets of clauses that link these literals. The implication graphs are used to identify the source of a conflict found during resolution and to calculate an efficient way in which to reset the currently maintained valuation of variables to a point that avoids the conflict.

In both forms of implication graphs, the primary interest is in finding *a* solution, not in identifying the transformations that lead from one solution to another. There are results showing how the set of all solutions to a 2-SAT problem can be captured by a so-called median graph, where nodes are solutions and edges possible transformations from one solution to another [51, Proposition 2.5]. It would be interesting to explore how this encoding relates to our encoding using VB rules and if this can be used to make our approach more efficient.

7. Conclusion

Building on [10], we have presented a generalised formal framework for generating graph-transformation rules that apply a particular change to a graph and ensure no additional constraint-violations are created as a result—the rules preserve consistency (Theorem 8). Importantly, our rules remain applicable wherever the original rule was applicable—they are non-blocking rules (Lemma 7)—different from related work that generates (negative) application conditions to avoid rule applications that would introduce new constraint violations.

Our formalisation is based on subgraph transformation systems [25, 26], which are convenient for deriving the activation diagrams that are central to our approach. We have introduced the notion of selection problems, which match this condition, and have shown that this encompasses numerous problems of real-world significance.

Deriving rules via activation diagrams and encoding them as variability-based rules (VB rules) correctly ensures that the rules are consistency-preserving (Theorem 8) and ensures that, at least when the given rules are *atomic*, every possible consistent graph can be reached from a consistent starting point (Theorem 10). The compact encoding also means that these rules can be computed tractably (Proposition 4).

Future work. The trick to achieve tractable computability is a degree of over-estimation of the repairs needed. As a result, the VB rules we derive may cover multiple identical instance rules (that are simply reached by different configurations of the feature model in the VB rule). In [10], we addressed this challenge through several additional optimisations of the VB rule encoding, avoiding self-activating repair cycles in the rule and ensuring consistent choices in different parts of the activation diagram. We leave the generalisation of such symmetry-breaking [52] optimisations for future work.

Another optimisation concerns the form of the rules we generate. Currently, our rules explicitly enumerate all possible ways in which a constraint-violation can be repaired. For example, for a minimum-multiplicity constraint, the deletion of an edge requires adding an edge to another node (possibly creating that node in the process). Our VB rules would explicitly enumerate all possible edges that could be created in this case. We need to create different instantiations of the VB rule to be able to explore different choices of edge; each of the instantiated rules would match exactly one edge. For future work, we are interested to study whether it is possible to shift this from the instantiation of the VB rule to the time when matches are computed for the rule. This way, there would be only *one* instantiated rule, which can match *all* possible edges to create (similar to the bespoke rules constructed in [13]). To do this, we would have to identify patterns in the activation diagram so these can be more compactly encoded in the resulting rules. It is an open research question whether this can be done efficiently.

This is related to another limitation of our current work: the specificity vs generality of the rules we generate. Both in [10] and the present paper, the rules we generate are specific to T; if a new graph is given—even if everything else stays the same—we need to regenerate the rules. Other works (e.g., [13]) produce rules that can be used for arbitrary instances of the same problem. We should investigate whether (under what circumstances) it is possible to extend our approach to generate such more abstract rules.

Solving these two challenges should also pave the way for dealing with search spaces of infinite size instead of subgraphs of a container graph T. Naturally, our approach cannot be arbitrarily generalised to the category of typed graphs—after all, satisfiability of nested constraints is well-known to be undecidable in that category [11]. Still, in our running example for instance, the obtained activation diagram is completely symmetrical for the different elements of the same type. We plan to investigate in how far we can exploit such symmetries to obtain a more general approach and reduce the overall size of the activation diagram.

Acknowledgements

This work was partially funded by the German Research Foundation (DFG), project "Model-Driven Optimization in Software Engineering" (TA 294/19-1) and by a Short-Term Scientific Mission "From Models to the ROAR-NET API and Back" in the context of the *COST Action Randomised Optimisation Algorithms Research Network* (ROAR-NET, https://www.roar-net.eu), CA22137, supported by COST (European Cooperation in Science and Technology).

We thank the anonymous reviewers for their valuable feedback, which helped us improve the presentation of this paper.

References

[1] A. Habel, C. Sandmann, Graph Repair by Graph Programs, in: M. Mazzara, I. Ober, G. Salaün (Eds.), Software Technologies: Applications and Foundations – STAF 2018 Collocated Workshops, Toulouse, France, June 25–29, 2018, Revised Selected Papers, Vol. 11176 of Lecture Notes in Computer Science, Springer, 2018, pp. 431–446. doi:10.1007/978-3-030-04771-9_31.
URL https://doi.org/10.1007/978-3-030-04771-9_31

[2] N. Nassar, H. Radke, T. Arendt, Rule-Based Repair of EMF Models: An Automated Interactive Approach, in: E. Guerra, M. van den Brand (Eds.), Theory and Practice of Model Transformation – 10th International Conference, ICMT@STAF 2017, Marburg, Germany, July 17–18, 2017, Proceedings, Vol. 10374 of Lecture Notes in Computer Science, Springer, 2017, pp. 171–181. doi:10.1007/978-3-319-61473-1_12.
URL https://doi.org/10.1007/978-3-319-61473-1_12

[3] J. Kosiol, L. Fritsche, N. Nassar, A. Schürr, G. Taentzer, Constructing Constraint-Preserving Interaction Schemes in Adhesive Categories, in: J. L. Fiadeiro, I. Tutu (Eds.), Recent Trends in Algebraic Development Techniques – 24th IFIP WG 1.3 International Workshop, WADT 2018, Egham, UK, July 2–5, 2018, Revised Selected Papers, Vol. 11563 of Lecture Notes in Computer Science, Springer, 2019, pp. 139–153. doi:10.1007/978-3-030-23220-7_8.
URL https://doi.org/10.1007/978-3-030-23220-7_8

[4] T. Kehrer, G. Taentzer, M. Rindt, U. Kelter, Automatically Deriving the Specification of Model Editing Operations from Meta-Models, in: P. V. Gorp, G. Engels (Eds.), Theory and Practice of Model Transformations – 9th International Conference, ICMT@STAF 2016, Vienna, Austria, July 4–5, 2016, Proceedings, Vol. 9765 of Lecture Notes in Computer Science, Springer, 2016, pp. 173–188. doi:10.1007/978-3-319-42064-6_12.
URL https://doi.org/10.1007/978-3-319-42064-6_12

[5] B. Becker, L. Lambers, J. Dyck, S. Birth, H. Giese, Iterative Development of Consistency-Preserving Rule-Based Refactorings, in: J. Cabot, E. Visser (Eds.), Theory and Practice of Model Transformations – 4th International Conference, ICMT@TOOLS 2011, Zurich, Switzerland, June 27–28, 2011. Proceedings, Vol. 6707 of Lecture Notes in Computer Science, Springer, 2011, pp. 123–137. doi:10.1007/978-3-642-21732-6_9.
URL https://doi.org/10.1007/978-3-642-21732-6_9

[6] N. Nassar, J. Kosiol, T. Arendt, G. Taentzer, OCL2AC: Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules, in: L. Lambers, J. H. Weber (Eds.), Graph Transformation – 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25–26, 2018, Proceedings, Vol. 10887 of Lecture Notes in Computer Science, Springer, 2018, pp. 171–177. doi:10.1007/978-3-319-92991-0_11.
URL https://doi.org/10.1007/978-3-319-92991-0_11

[7] N. Nassar, J. Kosiol, T. Arendt, G. Taentzer, Constructing Optimized Validity-Preserving Application Conditions for Graph Transformation Rules, in: E. Guerra, F. Orejas (Eds.), Graph Transformation – 12th International Conference, ICGT 2019, Held as Part of STAF 2019, Eindhoven, The Netherlands, July 15–16, 2019, Proceedings, Vol. 11629 of Lecture Notes in Computer Science, Springer, 2019, pp. 177–194. doi:10.1007/978-3-030-23611-3_11.
URL https://doi.org/10.1007/978-3-030-23611-3_11

[8] R. Heckel, A. Wagner, Ensuring consistency of conditional graph rewriting – a constructive approach, in: A. Corradini, U. Montanari (Eds.), Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation, SEGRAGRA 1995, Volterra, Italy, August 28 – September 1, 1995, Vol. 2 of Electronic Notes in Theoretical Computer Science, Elsevier, 1995, pp. 118–126. doi:10.1016/S1571-0661(05)80188-4.
URL https://doi.org/10.1016/S1571-0661(05)80188-4

[9] J. Kosiol, D. Strüber, G. Taentzer, S. Zschaler, Sustaining and improving graduated graph consistency: A static analysis of graph transformations, Sci. Comput. Program. 214 (2022) 102729. doi:10.1016/J.SCICO.2021.102729.
URL https://doi.org/10.1016/j.scico.2021.102729

[10] J. M. Horcas, D. Strüber, A. Burdusel, J. Martinez, S. Zschaler, We're Not Gonna Break It! Consistency-Preserving Operators for Efficient Product Line Configuration, IEEE Trans. Software Eng. 49 (3) (2023) 1102–1117. doi:10.1109/TSE.2022.3171404.
URL https://doi.org/10.1109/TSE.2022.3171404

[11] A. Habel, K. Pennemann, Correctness of high-level transformation systems relative to nested conditions, Math. Struct. Comput. Sci. 19 (2) (2009) 245–296. doi:10.1017/S0960129508007202.
URL https://doi.org/10.1017/S0960129508007202

[12] E. Biermann, C. Ermel, G. Taentzer, Formal Foundation of Consistent EMF Model Transformations by Algebraic Graph Transformation, Softw. Syst. Model. 11 (2) (2012) 227–250. doi:10.1007/s10270-011-0199-7.
URL https://doi.org/10.1007/s10270-011-0199-7

[13] A. Burdusel, S. Zschaler, S. John, Automatic generation of atomic multiplicity-preserving search operators for search-based model engineering, Softw. Syst. Model. 20 (6) (2021) 1857–1887. doi:10.1007/s10270-021-00914-w.
URL https://doi.org/10.1007/s10270-021-00914-w

[14] A. Habel, K. Pennemann, Nested Constraints and Application Conditions for High-Level Structures, in: H. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, G. Taentzer (Eds.), Formal Methods in Software and Systems Modeling, Essays Dedicated to Hartmut Ehrig, on the Occasion of His 60th Birthday, Vol. 3393 of Lecture Notes in Computer Science, Springer, 2005, pp. 293–308. doi:10.1007/978-3-540-31847-7_17.
URL https://doi.org/10.1007/978-3-540-31847-7_17

[15] K.-H. Pennemann, Development of correct graph transformation systems, Ph.D. thesis, University of Oldenburg, Germany (2009).
URL https://nbn-resolving.org/urn:nbn:de:gbv:715-oops-9483

[16] H. Radke, T. Arendt, J. S. Becker, A. Habel, G. Taentzer, Translating essential OCL invariants to nested graph constraints for generating instances of meta-models, Sci. Comput. Program. 152 (2018) 38–62. doi:10.1016/J.SCICO.2017.08.006.
URL https://doi.org/10.1016/j.scico.2017.08.006

[17] D. Strüber, J. Rubin, T. Arendt, M. Chechik, G. Taentzer, J. Plöger, Variability-based model transformation: formal foundation and application, Formal Aspects Comput. 30 (1) (2018) 133–162. doi:10.1007/s00165-017-0441-3.
URL https://doi.org/10.1007/s00165-017-0441-3

[18] S. John, J. Kosiol, L. Lambers, G. Taentzer, A graph-based framework for model-driven optimization facilitating impact analysis of mutation operator properties, Softw. Syst. Model. 22 (4) (2023) 1281–1318. doi:10.1007/S10270-022-01078-X.
URL https://doi.org/10.1007/s10270-022-01078-x

[19] M. Fleck, J. Troya, M. Wimmer, Marrying search-based optimization and model transformation technology, in: Proc. 1st North American Search Based Software Engineering Symposium (NasBASE'15), 2015, preprint available at http://martin-fleck.github.io/momot/downloads/NasBASE_MOMoT.pdf.

[20] H. Abdeen, D. Varró, H. A. Sahraoui, A. S. Nagy, C. Debreceni, Á. Hegedüs, Á. Horváth, Multi-objective optimization in rule-based design space exploration, in: I. Crnkovic, M. Chechik, P. Grünbacher (Eds.), ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden – September 15 – 19, 2014, ACM, 2014, pp. 289–300. doi:10.1145/2642937.2643005.
URL https://doi.org/10.1145/2642937.2643005

[21] S. Ehmes, M. Kratz, A. Schürr, Graph-Based Specification and Automated Construction of ILP Problems, in: R. Heckel, C. M. Poskitt (Eds.), Proceedings of the Thirteenth International Workshop on Graph Computation Models, GCM@STAF 2022, Nantes, France, 6th July 2022, Vol. 374 of EPTCS, 2022, pp. 3–22. doi:10.4204/EPTCS.374.3.
URL https://doi.org/10.4204/EPTCS.374.3

[22] J. Murphy, A. Burdusel, M. Luck, S. Zschaler, E. Black, Deriving Persuasion Strategies Using Search-Based Model Engineering, in: S. Modgil, K. Budzynska, J. Lawrence (Eds.), Computational Models of Argument – Proceedings of COMMA 2018, Warsaw, Poland, 12–14 September 2018, Vol. 305 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2018, pp. 221–232. doi:10.3233/

978-1-61499-906-5-221.
URL https://doi.org/10.3233/978-1-61499-906-5-221

[23] N. Macedo, J. Tiago, A. Cunha, A Feature-Based Classification of Model Repair Approaches, IEEE Trans. Software Eng. 43 (7) (2017) 615–640. doi:10.1109/TSE.2016.2620145.
URL https://doi.org/10.1109/TSE.2016.2620145

[24] A. Lauer, J. Kosiol, G. Taentzer, Empowering model repair: a rule-based approach to graph repair without side effects – extended version, Innov. Syst. Softw. Eng. 20 (4) (2024) 597–618. doi:10.1007/S11334-024-00587-W.
URL https://doi.org/10.1007/s11334-024-00587-w

[25] A. Corradini, F. Hermann, P. Sobocinski, Subobject Transformation Systems, Appl. Categorical Struct. 16 (3) (2008) 389–419. doi:10.1007/s10485-008-9127-6.
URL https://doi.org/10.1007/s10485-008-9127-6

[26] F. Hermann, A. Corradini, H. Ehrig, Analysis of permutation equivalence in $\mathcal{M}$-adhesive transformation systems with negative application conditions, Math. Struct. Comput. Sci. 24 (4) (2014). doi:10.1017/S0960129512000382.
URL https://doi.org/10.1017/S0960129512000382

[27] J. Kosiol, S. Zschaler, A note on nested conditions for finite categories of subgraphs, arXiv preprint (2026). doi:10.48550/arXiv.2601.18376.
URL https://arxiv.org/abs/2601.18376

[28] J. Kosiol, D. Strüber, G. Taentzer, S. Zschaler, Finding the Right Way to Rome: Effect-Oriented Graph Transformation, in: M. Fernández, C. M. Poskitt (Eds.), Graph Transformation – 16th International Conference, ICGT 2023, Held as Part of STAF 2023, Leicester, UK, July 19–20, 2023, Proceedings, Vol. 13961 of Lecture Notes in Computer Science, Springer, 2023, pp. 43–63. doi:10.1007/978-3-031-36709-0_3.
URL https://doi.org/10.1007/978-3-031-36709-0_3

[29] F. Gemeinhardt, M. Eisenberg, S. Klikovits, M. Wimmer, Model-Driven Optimization for Quantum Program Synthesis with MOMoT, in: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2023 Companion, Västerås, Sweden, October 1–6, 2023, IEEE, 2023, pp. 614–621. doi:10.1109/MODELS-C59198.2023.00100.
URL https://doi.org/10.1109/MODELS-C59198.2023.00100

[30] B. Cheang, H. Li, A. Lim, B. Rodrigues, Nurse rostering problems – a bibliographic survey, Eur. J. Oper. Res. 151 (3) (2003) 447–460. doi:10.1016/S0377-2217(03)00021-3.
URL https://doi.org/10.1016/S0377-2217(03)00021-3

[31] I. Rahimi, A. H. Gandomi, A Comprehensive Review and Analysis of Operating Room and Surgery Scheduling, Arch. Computat. Methods Eng. 28 (3) (2021) 1667–1688. doi:10.1007/s11831-020-09432-2.
URL https://doi.org/10.1007/s11831-020-09432-2

[32] G. Caselli, M. Delorme, M. Iori, Integer Linear Programming for the Tutor Allocation Problem: A practical case in a British University, Expert Syst. Appl. 187 (2022) 115967. doi:10.1016/J.ESWA.2021.115967.
URL https://doi.org/10.1016/j.eswa.2021.115967

[33] M. Fleck, J. Troya, M. Wimmer, The Class Responsibility Assignment Case, in: A. García-Domínguez, F. Krikava, L. M. Rose (Eds.), Proceedings of the 9th Transformation Tool Contest, co-located with the 2016 Software Technologies: Applications and Foundations (STAF 2016), Vienna, Austria, July 8, 2016, Vol. 1758 of CEUR Workshop Proceedings, CEUR-WS.org, 2016, pp. 1–8.
URL https://ceur-ws.org/Vol-1758/paper1.pdf

[34] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of Algebraic Graph Transformation, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2006. doi:10.1007/3-540-31188-2.

[35] S. Lack, P. Sobociński, Adhesive and quasiadhesive categories, RAIRO Theor. Informatics Appl. 39 (3) (2005) 511–545. doi:10.1051/ITA:2005028.
URL https://doi.org/10.1051/ita:2005028

[36] A. Rensink, The GROOVE Simulator: A Tool for State Space Generation, in: J. L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, Charlottesville, VA, USA, September 27 – October 1, 2003, Revised Selected and Invited Papers, Vol. 3062 of Lecture Notes in Computer Science, Springer, 2003, pp. 479–485. doi:10.1007/978-3-540-25959-6_40.
URL https://doi.org/10.1007/978-3-540-25959-6_40

[37] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations, in: D. C. Petriu, N. Rouquette, Ø. Haugen (Eds.), Model Driven Engineering Languages and Systems – 13th International Conference, MODELS 2010, Oslo, Norway, October 3–8, 2010, Proceedings, Part I, Vol. 6394 of Lecture Notes in Computer Science, Springer, 2010, pp. 121–135. doi:10.1007/978-3-642-16145-2_9.
URL https://doi.org/10.1007/978-3-642-16145-2_9

[38] A. Rensink, Representing first-order logic using graphs, in: H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Eds.), Graph Transformations, Second International Conference, ICGT 2004, Rome, Italy, September 28 – October 2, 2004, Proceedings, Vol. 3256 of Lecture Notes in Computer Science, Springer, 2004, pp. 319–335. doi:10.1007/978-3-540-30203-2_23.
URL https://doi.org/10.1007/978-3-540-30203-2_23

[39] H. Ehrig, U. Golas, A. Habel, L. Lambers, F. Orejas, $\mathcal{M}$-adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation, Math. Struct. Comput. Sci. 24 (4) (2014). doi:10.1017/S0960129512000357.
URL https://doi.org/10.1017/S0960129512000357

[40] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. Rep. CMU/SEI-90-TR-021, Carnegie-Mellon University Software Engineering Institute (1990).
URL https://www.sei.cmu.edu/library/feature-oriented-domain-analysis-foda-feasibility-study/

[41] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. C. Ferrari, S. S. Khan, F. C. Filho, F. Dantas, Evolving software product lines with aspects: an empirical study on design stability, in: W. Schäfer, M. B. Dwyer, V. Gruhn (Eds.), 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10–18, 2008, ACM, 2008, pp. 261–270. doi:10.1145/1368088.1368124.
URL https://doi.org/10.1145/1368088.1368124

[42] U. Golas, A. Habel, H. Ehrig, Multi-amalgamation of rules with application conditions in $\mathcal{M}$-adhesive categories, Math. Struct. Comput. Sci. 24 (4) (2014). doi:10.1017/S0960129512000345.
URL https://doi.org/10.1017/S0960129512000345

[43] J. Martinez, D. Strüber, J. M. Horcas, A. Burdusel, S. Zschaler, Acapulco: an extensible tool for identifying optimal and consistent feature model configurations, in: A. Felfernig, L. Fuentes, J. Cleland-Huang, W. K. G. Assunção, C. Quinton, J. Guo, K. Schmid, M. Huchard, I. Ayala, J. M. Rojas, V. Le, J. M. Horcas (Eds.), SPLC '22: 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 – 16, 2022, Volume B, ACM, 2022, pp. 50–53. doi:10.1145/3503229.3547067.
URL https://doi.org/10.1145/3503229.3547067

[44] L. G. Valiant, The complexity of computing the permanent, Theor. Comput. Sci. 8 (1979) 189–201. doi:10.1016/0304-3975(79)90044-6.
URL https://doi.org/10.1016/0304-3975(79)90044-6

[45] O. Semeráth, D. Varró, Graph Constraint Evaluation over Partial Models by Constraint Rewriting, in: E. Guerra, M. van den Brand (Eds.), Theory and Practice of Model Transformation – 10th International Conference, ICMT@STAF 2017, Marburg, Germany, July 17–18, 2017, Proceedings, Vol. 10374 of Lecture Notes in Computer Science, Springer, 2017, pp. 138–154. doi:10.1007/978-3-319-61473-1_10.
URL https://doi.org/10.1007/978-3-319-61473-1_10

[46] S. Schneider, L. Lambers, F. Orejas, A logic-based incremental approach to graph repair featuring delta preservation, Int. J. Softw. Tools Technol. Transf. 23 (3) (2021) 369–410. doi:10.1007/S10009-020-00584-X.
URL https://doi.org/10.1007/s10009-020-00584-x

[47] A. Ficsor, I. A. Papp, K. Marussy, O. Semeráth, From constraints to commands: Graph predicate differentiation in 4-valued first-order logic, J. Object Technol. 25 (3) (2026) 3:43–56. doi:doi:10.5381/jot.2026.25.3.a4.

[48] R. M. Karp, Reducibility Among Combinatorial Problems, in: R. E. Miller, J. W. Thatcher (Eds.), Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103. doi:10.1007/978-1-4684-2001-2_9.
URL https://doi.org/10.1007/978-1-4684-2001-2_9

[49] B. Aspvall, M. F. Plass, R. E. Tarjan, A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas, Inf. Process. Lett. 8 (3) (1979) 121–123. doi:10.1016/0020-0190(79)90002-4.
URL https://doi.org/10.1016/0020-0190(79)90002-4

[50] P. Beame, H. A. Kautz, A. Sabharwal, Understanding the Power of Clause Learning, in: G. Gottlob, T. Walsh (Eds.), IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9–15, 2003, Morgan Kaufmann, 2003, pp. 1194–1201.
URL http://ijcai.org/Proceedings/03/Papers/171.pdf

[51] H.-J. Bandelt, V. Chepoi, Metric graph theory and geometry: a survey, in: Surveys on Discrete and Computational Geometry, Vol. 453 of Contemporary Mathematics, American Mathematical Society, 2008, pp. 49–86. doi:10.1090/conm/453/08795.

[52] T. Walsh, General Symmetry Breaking Constraints, in: F. Benhamou (Ed.), Principles and Practice of Constraint Programming – CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25–29, 2006, Proceedings, Vol. 4204 of Lecture Notes in Computer Science, Springer, 2006, pp. 650–664. doi:10.1007/11889205_46.
URL https://doi.org/10.1007/11889205_46

[53] A. Knüppel, T. Thüm, S. Mennicke, J. Meinicke, I. Schaefer, Is there a mismatch between real-world feature models and product-line research?, in: E. Bodden, W. Schäfer, A. van Deursen, A. Zisman (Eds.), Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4–8, 2017, ACM, 2017, pp. 291–302. doi:10.1145/3106237.3106252.
URL https://doi.org/10.1145/3106237.3106252

[54] D. S. Batory, Feature models, grammars, and propositional formulas, in: J. H. Obbink, K. Pohl (Eds.), Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26–29, 2005, Proceedings, Vol. 3714 of Lecture Notes in Computer Science, Springer, 2005, pp. 7–20. doi:10.1007/11554844_3.
URL https://doi.org/10.1007/11554844_3

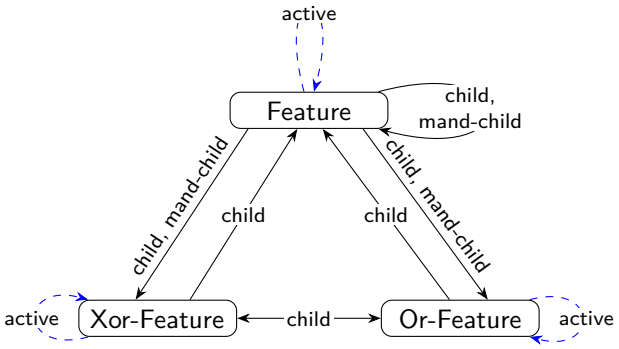


Figure A.38: A type graph for feature models

Appendix A. Comparison to consistency-preserving configuration operators

In this section, we compare our *interaction principles* (Definition 10) to the *activation* and *deactivation principles* of Horcas et al. [10]. Horcas et al. exclusively work on feature models, which we introduced in Sect. 3.4 (Definition 5). We start by presenting how the problem of feature selection can be cast in our framework, i.e., be presented as a category $\mathbf{Sub}(T \supseteq P)$ (Appendix A.1), and subsequently perform the actual comparison (Appendix A.2).

Appendix A.1. Formalising feature selection via subgraph transformation systems

For casting the problem of feature selection in the setting of this paper, i.e., as a category $\mathbf{Sub}(T \supseteq P)$, we first have to provide a suitable type graph TG. Figure A.38 presents our solution, where we make the following design decisions and use the following conventions: We use node types to distinguish between the different kinds of features, namely plain Features, Xor-Features and Or-Features. Xor-Features and Or-Features indicate that their children constitute a xor-group or an or-group, respectively. To model the parent–child relationships, we use graph edges; a dedicated edge type mand-child can mark mandatory children. To not clutter the figure, we sometimes draw two edges as a single one, separating the two types via a comma; similarly, the bidirectional child-edge between Xor-Feature and Or-Feature represents two edges—one in every direction. The edges of type active serve to enable expressing that a feature has been selected as part of a configuration. Analogous to our visual syntax for the CRA problem, drawing these edges dashed and blue indicates that edges of this type are the variable parts of models (enabling the computation of configurations), whereas all other types, depicted solid and in black, serve to model "problems", i.e., plain feature diagrams.

Figure A.39 shows the Mobile Media feature diagram from above (Fig. 13) as a graph typed over the type graph TG from Fig. A.38. We omit the cross-tree constraint in that figure and (to avoid cluttering the diagram) use the visual notation from Fig. 13 to distinguish between mandatory and other children. Moreover, we abbreviate some of the names. Because we have included all possible edges of type active, the graph from Fig. A.39 can serve as container graph T when expressing the feature selection problem for the Mobile Media example in our formal framework. The problem graph P is the subgraph consisting of all solid, black elements, i.e., consisting of all elements except for the edges of type active. In this way, configurations F of the Mobile Media feature model correspond to graphs $G \in \mathbf{Sub}(T \supseteq P)$, mediated by the presence or absence of the active-edge.

In the following, we show how the six properties of valid configurations (see Definition 5) can be expressed as constraints in our framework. For most cases, we first show how the property can be expressed as nested constraint over the type graph from Fig. A.38 (i.e., as a constraint on objects in $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$) and then exemplify how this is instantiated to the Mobile Media case, i.e., to $\mathbf{Sub}(T \supseteq P)$ as just introduced.

CRoot. The property CRoot, requiring the root feature to be part of every valid configuration F, can be generically expressed as nested constraint over TG that requires exactly one feature without incoming

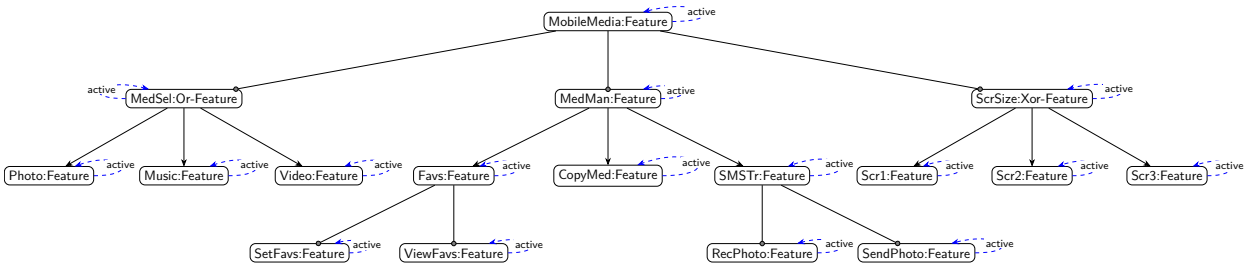


Figure A.39: Feature diagram of Mobile Media, displayed as graph typed over the type graph from Fig. A.38 (omitting the cross-tree constraint)



Figure A.40: Property CRoot instantiated for the Mobile Media example

child-edge to be active. In this case, however, because the root node of a given feature diagram is known, directly defining this constraint in $\mathbf{Sub}(T \supseteq P)$ is easy. In the Mobile Media example, one just requires the node MobileMedia to be active. This constraint is shown in Fig. A.40.

CPar. In $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$, the property CPar, requiring that together with a child its parent feature is activated, can be expressed as indicated in Fig. A.41. To completely express CPar, we have to build a conjunction of such constraints that consider every allowed combination of kinds of features (Feature, Or-Feature, Xor-Feature) and edge kind (child, mand-child).

When instantiating and normalising these constraints for the Mobile Media example according to the procedures developed in [27] (such that we gain constraints meeting Assumption 1), for every child and every mand-child edge, we obtain a constraint like the one displayed in Fig. A.42 for the case of the mand-child edge between Favs and SetFavs. (Strictly speaking, the constraint shown in Fig. A.42 is not an instantiation of the one shown in Fig. A.41 but of one that is analogous to this but concerns edges of type mand-child.) Note that the structure of the result of this instantiation is not specific to the Mobile Media example: For every feature model, for every edge of type mand-child or child we will obtain a constraint in implication format completely analogous to the one in Fig. A.42, just that the source and target nodes of the mand-child- or child-edge might also be of types Or-Feature or Xor-Feature.

CMand. In $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$, the property CMand, requiring that together with a parent feature its mandatory children are activated, can be expressed as indicated in Fig. A.43. To completely express CMand, we have to build a conjunction of such constraints that consider every allowed kind of target node for the edge kind mand-child, i.e., we add analogous constraints with Or-Feature and Xor-Feature as target node as conjuncts.

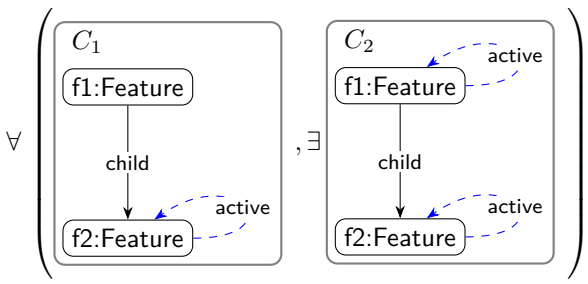


Figure A.41: Property CPar (for ordinary Features and ordinary child-edges) as constraint in $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$

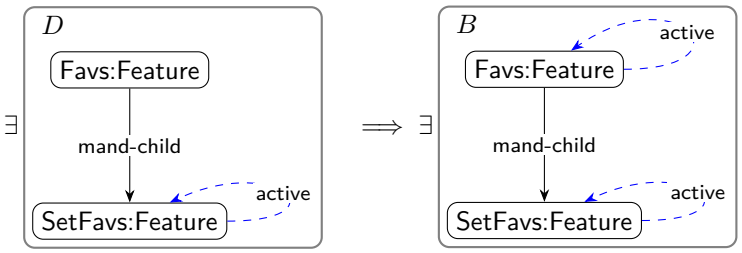


Figure A.42: Instantiation of CPar for Feature SetFavs and its incoming edge from the Mobile Media example

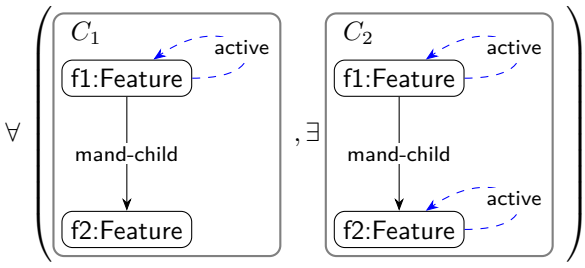


Figure A.43: Property CMand (for ordinary Features as target nodes) as constraint in $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$

When instantiating and normalising these constraints for the Mobile Media example according to the procedures developed in [27] (such that we gain constraints meeting Assumption 1), for every mand-child edge, we obtain a constraint like the one displayed in Fig. A.44 for the case of the mand-child edge between Favs and SetFavs. Again, the structure of the result of this instantiation is not specific to the Mobile Media example: For every feature model, for every edge of type mand-child we will obtain a constraint in implication format completely analogous to the one in Fig. A.44, just that the source and target nodes of the mand-child-edge might also be of types Or-Feature or Xor-Feature.

COr. In $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$, the property COr, requiring that together with an Or-Feature one of its child features is activated, can be expressed as displayed in Fig. A.45; note that the child feature to be activated can be of types Feature, Xor-Feature or Or-Feature.

When instantiating and normalising this constraint for the Mobile Media example according to the procedures developed in [27] (such that we gain constraints meeting Assumption 1), we obtain the constraint displayed in Fig. A.46, because MediaSelection is the only Or-Feature. Again, the structure of the result of this instantiation is not specific to the Mobile Media example: For every feature model, for every node of type Or-Feature we will obtain a constraint in implication format completely analogous to the one in

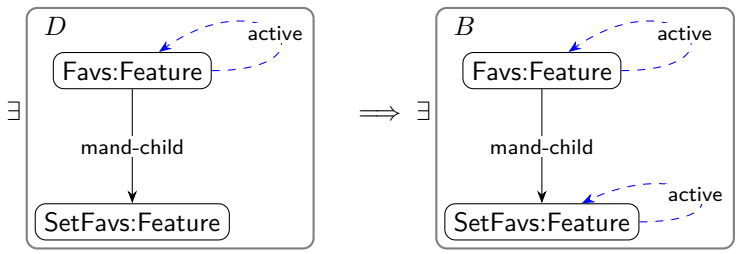


Figure A.44: Instantiation of CMand for the mand-child edge between Favs and SetFavs

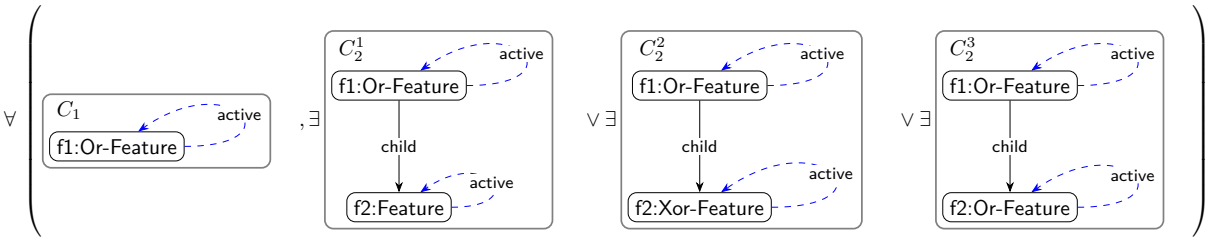


Figure A.45: Property COr

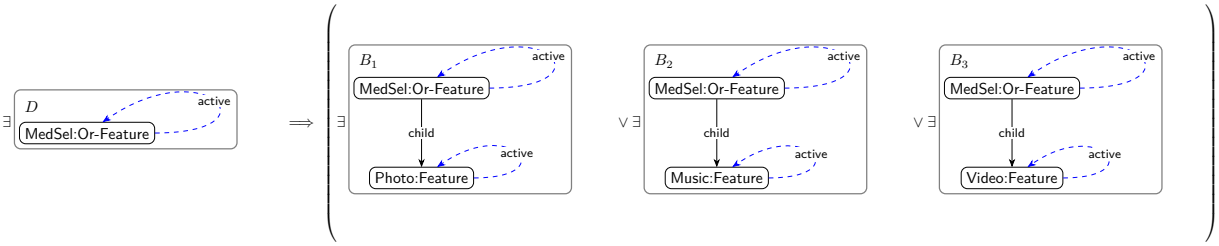


Figure A.46: Instantiation of property COr for the Mobile Media example

Fig. A.46, just that the number of available alternatives corresponds to the number of potential children of the instantiated Or-Feature and these children might also be of types Or-Feature or Xor-Feature.

CXor. In $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$, the property CXor, requiring that together with a Xor-Feature *exactly* one of its child features is activated, can be expressed as indicated in Fig. A.47. The first line of the constraint is analogous to the one for COr, requiring the activation of *at least one* child feature of one of the types Feature, Xor-Feature or Or-Feature. The second line forbids more than one child feature to be activated, where we have to consider all six possible combinations of types of two child features.

When instantiating and normalising this constraint for the Mobile Media example according to the procedures developed in [27] (such that we gain constraints meeting Assumption 1), we obtain the constraint displayed in Fig. A.48, because ScreenSize is the only Xor-Feature. According to our assumptions on constraints, we would consider the four conjuncts of that constraint as four individual constraints, the first being in implication format, the other three being purely negative. Again, the structure of the result of this instantiation is not specific to the Mobile Media example: For every feature model, for every node of type

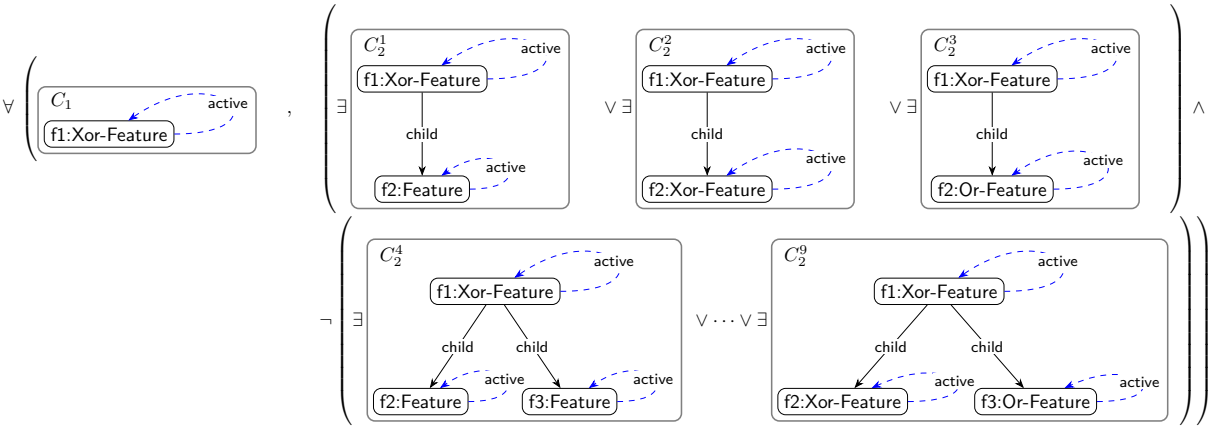


Figure A.47: Property CXor

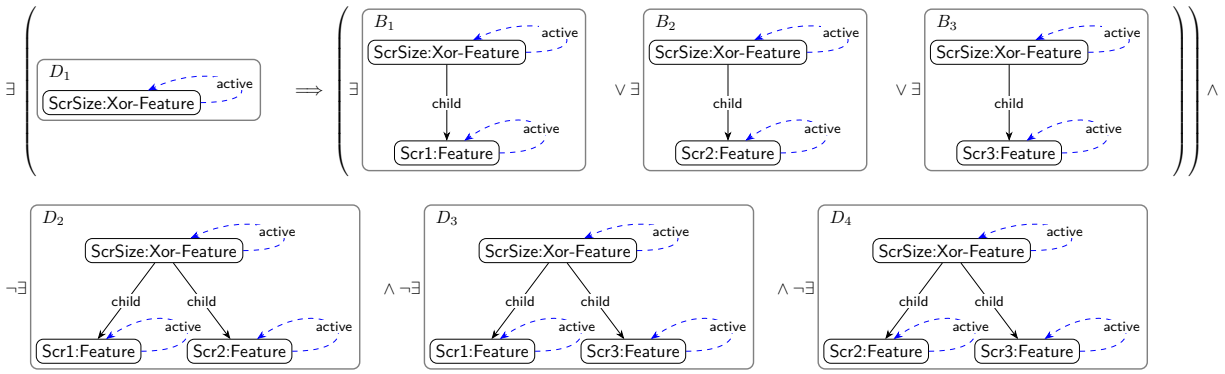


Figure A.48: Instantiation of property CXor for the Mobile Media example

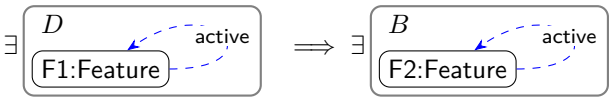


Figure A.49: Schema for property CReq

Xor-Feature we will obtain an analogous conjunction of a first constraint in implication format, followed by a series of negative constraints (as in Fig. A.48). Again, what varies is that the number of available alternatives in the implication corresponds to the number of potential children of the instantiated Xor-Feature and these children might also be of types Or-Feature or Xor-Feature. Moreover, the number of negative constraints is $\binom{t}{2}$, with t being the number of potential children of the Xor-Feature, i.e., one negative constraint for every possible pair of two children.

CRel. The relational, i.e., cross-tree constraints supported by the approach of Horcas et al. are simple requirement (CReq) or exclusion (CExcl) relationships between individual features. As such, they are specific to a concrete feature model and can then directly be formalised in $\mathbf{Sub}(T \supseteq P)$; their structures, however, are always the same. Figure A.49 shows the schema for requirement constraints, where the activation of one feature (called F1 here) requires the activation of another one (here called F2); in principle, the features can also be of types Or-Feature and Xor-Feature (with any combination of the three types being possible). The exclusion constraints, in contrast, forbid two specific features being activated at the same time, as schematically depicted in Fig. A.50, where the two concerned features are also called F1 and F2. Again, these features might also be of types Or-Feature and Xor-Feature. In the Mobile Media example, one requirement constraint is present, demanding together with the Feature SMSTransfer also the feature CopyMedia is activated. We depict this concrete constraint in Fig. A.51.

Appendix A.2. Retrieving interaction principles for feature selection

Based on our above formalisation of the feature selection problem via subgraph transformation systems, we compare our interaction principles to the activation and deactivation principles of Horcas et al. [10]. For this, we go over the just presented constraints for the feature selection problem and determine which interaction principles are triggered. Recall that in our formalisation, the set $T \setminus P$ consists of active-edges;

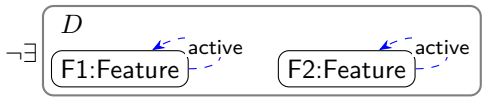


Figure A.50: Schema for property CExcl

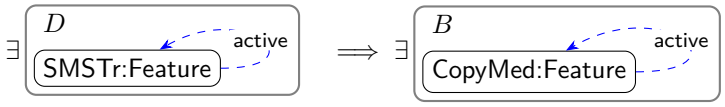


Figure A.51: Concrete instantiation of property CReq in case of the Mobile Media example

that is, ElementDecision nodes in an activation diagram correspond to the creation or deletion of such an edge, which, intuitively, corresponds to adding or removing a feature to or from a configuration. We discuss the constraint and the interaction principles they trigger in the order in which we presented them above.

Interaction principles triggered by CRoot. As Horcas et al. [10] analogously do, we can ensure CRoot (which just requires the root feature to be part of every valid configuration) by adding its active-edge to the problem definition P (making it part of every configuration) and removing/adapting all rules that somehow (de)activate the root feature. Therefore, we do not need any interaction principles for this constraint; exactly as is the case in [10].

Interaction principles triggered by CPar. Instantiated CPar constraints (requiring parent features to be activated together with their children) are in implication format (compare Fig. A.42). Recalling that exactly the active-edges belong to $T \setminus P$ and comparing the graphs D and B in this kind of constraint shows that in our approach two interaction principles apply. First, to the ElementCreation node corresponding to the active-edge of the child feature (SetFavs in our example), the interaction principle CreationImpl applies, inserting an Or node with a single outgoing edge to the ElementCreation node corresponding to the active-edge of the parent feature. This corresponds to the activation principle ActPar of Horcas et al. [10].[4]

Secondly, to the ElementDeletion node corresponding to the active-edge of the parent feature (Favs in our example), the interaction principle DeletionImpl applies, inserting an Or node with a single outgoing edge to the ElementDeletion node corresponding to the active-edge of the child feature. Since we apply this principle for every child feature, this corresponds to the deactivation principle DeChild of Horcas et al. [10].

Interaction principles triggered by CMand. Instantiated CMand constraints (requiring mandatory children to be activated together with their parents) are in implication format (compare Fig. A.44). Recalling that exactly the active-edges belong to $T \setminus P$ and comparing the graphs D and B in this kind of constraint shows that in our approach two interaction principles apply. First, to the ElementCreation node corresponding to the active-edge of the parent feature (Favs in our example), the interaction principle CreationImpl applies, inserting an Or node with a single outgoing edge to the ElementCreation node corresponding to the active-edge of the child feature. Because such a constraint exists for every mand-child-edge, this corresponds to the activation principle ActMand of Horcas et al. [10].

Secondly, to the ElementDeletion node corresponding to the active-edge of the child feature (SetFavs in our example), the interaction principle DeletionImpl applies, inserting an Or node with a single outgoing edge to the ElementDeletion node corresponding to the active-edge of the parent feature. This corresponds to the deactivation principle DeParent of Horcas et al. [10].

Interaction principles triggered by COr. Instantiated COr constraints (requiring at least one of the potential children of an Or group to be activated with the parent) are also in implication format (compare Fig. A.46). Recalling that exactly the active-edges belong to $T \setminus P$ and comparing the graphs D and $B_1, \ldots, B_t$ in this kind of constraint shows that in our approach again two interaction principles apply. First, to the

[4]There are two small differences that are, however, not fundamental. First, Horcas et al. only activate non-core features, i.e., they have a pre-processing step we ignore here but which can easily incorporated into our approach as well. Moreover, to not discuss corner cases separately, we insert an Or node even though it later turns out to only have a single outgoing edge, whereas Horcas et al. directly insert an edge in their activation diagram. The behaviours, however, are identical and we could simply integrate a post-processing step after computing our activation diagrams that replaces superflous And and Or nodes by edges. This slight difference in syntax of activation diagrams appears regularly and we do not mention it further in the following.

ElementCreation node corresponding to the active-edge of the parent feature (MedSel in our example), the interaction principle CreationImpl applies, inserting an Or node with t outgoing edges to the ElementCreation nodes corresponding to the active-edges to the t available child features of the Or group. This corresponds to the part of the activation principle ActGroup of Horcas et al. [10] that concerns Or groups.

Secondly, to each ElementDeletion node corresponding to one of the active-edges of the children of the Or group, the interaction principle DeletionImpl applies, inserting an Or node with outgoing edges to (i) the ElementDeletion node corresponding to the active-edge of the parent node of the Or group and (ii) the $t-1$ ElementCreation nodes corresponding to the active-edges of the siblings. This corresponds to the deactivation principle DeOr of Horcas et al. [10] (except for the fact that, again, they take into account that the parent feature might be an indelible "core" feature).

Interaction principles triggered by CXor. The first conjunct of an instantiated CXor constraint, i.e., the conjunct in implication format, is structurally equivalent to instantiated COr constraints. Therefore, the same interaction principles apply as above; this corresponds to the part of the activation principle ActGroup of Horcas et al. [10] that concerns Xor groups and their deactivation principle DeXor (again, up to our missing optimisation with regard to core features).

Regarding the conjuncts consisting of negative constraints, we observe the following. To every such constraint, the interaction principle CreationNeg applies thrice, namely once per active-edge. In each case, starting from the ElementCreation node corresponding to the respective active-edge, an Or node is inserted into the activation diagram with two outgoing edges to the ElementDeletion nodes corresponding to the other two active-edges from the graph of the negative constraint. Because we have such a negative constraint for every pair of children of a Xor-Feature, in our activation diagrams the following structures arise:

1. From every ElementCreation node that corresponds to the active-edge of a Xor-Feature, there are $\binom{t}{2}$ outgoing edges to Or nodes, each of which has two outgoing edges directed at two ElementDeletion nodes corresponding to the active-edges of two of the Xor-Feature's children (which t being the number of children).

2. For every ElementCreation node that corresponds to an active-edge of one of the t children of that Xor-Feature, we have $t-1$ outgoing edges to Or nodes, each of which has two outgoing edges, one directed to the ElementDeletion node corresponding to the active-edge of the Xor-Feature and the other one directed to an ElementDeletion node corresponding to the active-edge of a sibling.

Intuitively, considered in isolation these structures in the activation diagram ensure consistency of the second part of the constraint CXor (i.e., the upper bound expressing that a Xor-Feature cannot simultaneously be active together with two of its children) by either allowing for a Xor-Feature to be active but only one of its children or preventing the Xor-Feature to be active (but arbitrarily many of its children). However, when later using the activation diagram to derive rules, because of the structures added by other constraints, we have the following overall effect: The structures inserted because of constraint CPar ensure that together with the child of a Xor-Feature this parent feature is also activated. Thus, the alternative to deactivate the Xor-Feature is a "dead" alternative and together with the child of a Xor-Feature being activated always all of its siblings are deactivated. This effect exactly corresponds to the activation principle ActXor of Horcas et al. [10]. However, they encode this far more elegantly in their feature activation diagrams. Beyond not encoding the "dead" alternative of deactivating a Xor-Feature to ensure that is not simultaneously active together with two of its children, they also encode the necessary deactivation of siblings only once, namely when activating the child of a Xor-Feature. Our approach results in encoding this twice, namely as consequence of activating the Xor-Feature and as consequence of activating one of its children. Because, by the first part of constraint CXor, activating a Xor-Feature triggers the activation of (at least) one of its children (and, hence, also the consequences thereof), it suffices to rely on this transitive effect and encode the deactivation of siblings only as consequences of the activation of children.

Interaction principles triggered by CReq. Instantiated CReq constraints (expressing the dependence of one specific feature on another) are in implication format (compare Fig. A.49). Recalling that exactly the

active-edges belong to $T \setminus P$ and comparing the graphs D and B in this kind of constraint shows that in our approach two interaction principles apply. First, to the ElementCreation node corresponding to the active-edge of the node of the antecedent D of the implication (F1 in Fig. A.49 and SMSTr in the concrete example in Fig. A.51), the interaction principle CreationImpl applies, inserting an Or node with a single outgoing edge to the ElementCreation node corresponding to the active-edge of the feature in the consequent B of the implication (F2 in Fig. A.49 and CopyMed in the concrete example in Fig. A.51). This corresponds to the activation principle ActReq of Horcas et al. [10] (again, up to the fact that they consider core features).

Secondly, to the ElementDeletion node corresponding to the active-edge of the feature of the consequent B (CopyMed in our example), the interaction principle DeletionImpl applies, inserting an Or node with a single outgoing edge to the ElementDeletion node corresponding to the active-edge of the feature of the antecedent D (SMSTr in our example). This corresponds to the deactivation principle DeReq of Horcas et al. [10].

Interaction principles triggered by CExcl. Instantiated CExcl constraints (forbidding two concrete features to be activated simultaneously) are negative constraints; their principal structure is shown in Fig. A.50. Recalling that exactly the active-edges belong to $T \setminus P$, in our approach the interaction principle CreationNeg applies to both active-edges. That is, in the activation diagram at both ElementCreation nodes corresponding to these two active-edges we insert an outgoing edge to an Or node that, in turn, has a single outgoing edge to the ElementDeletion node corresponding to the respective other active-edge. This corresponds to the activation principle ActExc of Horcas et al. [10].

Discussion. Comparing our interaction principles with the activation and deactivation principles for the concrete case of feature selection in [10] shows two things: First, by and large, our four interaction principles offer a systematic way to derive the 11 activation and deactivation principles in the case of feature selection in [10]. However, Horcas et al.'s bespoke derivation of these 11 principles includes several optimisations that our general approach does not automatically achieve:

1. In the use case of feature selection, every application of an interaction principle in our approach extends the activation diagram in a way that is equivalent to how the feature-activation diagrams in [10] would be extended by one of their (de)activation principles in the same situation. Conversely, the effect of every (de)activation principles from [10] is also achieved by application of one of our interaction principles in the according situation. *Therefore, the approach of [10] and ours are equivalent for the use case of feature selection in the sense that the resulting (feature) activation diagrams allow deriving the same rules.* Thus, we indeed generalise the approach of Horcas et al.

   This also means that we provide a far more thorough formal foundation for that approach. In [10], correctness of their approach is rather sketched and no equivalent to our simulation result (Theorem 10) is proven so that with our approach we also obtain new results for the use case of feature allocation. Moreover, our general approach can serve as a basis to extend the approach of Horcas et al. [10] to also support more complex cross-tree constraints. As follows from [27], we support first-order logic on graphs (and, consequently, complex constraints among features), whereas Horcas et al. restrict themselves to simple *requires* or *excludes* constraints (and preprocess complex constraints using the approach by Knüppel et al. [53]). Considering more complex relations among features [54] is of high practical importance [53].

2. Horcas et al. [10] use domain knowledge and the specific structure of their considered constraints to arrive at more streamlined feature activation diagrams. Minor differences are the following: Instead of inserting Or nodes with a single outgoing edge, they just insert edges. To avoid individual discussions of such corner cases, we do not consider this in our presentation; however, a simple post-processing step on our activation diagrams could replace such Or nodes by an edge without any further changes being necessary to our approach. Moreover, Horcas et al. consider "core" features in their approach, i.e., features that have to be active in any valid configuration (because of being mandatory children of core features, because of cross-tree constraints, etc.). Our approach can simply be extended to this by

treating core features like we treat the root feature, i.e., by making the active-edges of such features part of the problem definition P, which leads to the desired result.

The difference discussed regarding the constraint CXor is more distinct, where, in our approach, we add alternatives to our feature diagrams that a domain analysis (i.e., an analysis of the interactions between constraints) proves to never be consistently realisable. Theorem 5 shows that, at least in general, we cannot weaken our interaction principles without losing other results like our simulation theorem. However, for specific domains, like feature selection, domain-specific improvements are clearly available, as shown by [10].

3. Beyond their general construction of feature-activation diagrams, Horcas et al. [10] also present a post-processing step to optimise these diagrams and the resulting rules. These involve removing so-called Or-Overlaps—redundant Or branches that can be combined to make consistent choices across a diagram—and preventing self-activating cycles in the diagrams. They also present optimisations for the resulting rules, not least by removing dead features and dead elements from the generated VB-rules. These optimisations are orthogonal to our interaction principles and could, in principle, also be applied to the activation diagrams and rules generated by our approach. We leave generalising these optimisations to future work.

Appendix B. Proofs

In this section, we present all proofs for the results in our paper.

Appendix B.1. Proofs from Section 3

Proof of Proposition 2. We first prove that the base rule p_b is a subrule of the induced rule p^G_ind and then that p^G_ind is pure. For the proof, compare Fig. B.52, where, for reference, we just provide names for the squares (compared to Fig. 20).

To show the subrule relationship, we have to show that squares (1) + (4) and (2) + (5) are pullbacks (in $\mathbf{Graph}^{\mathbf{fin}}_{\mathbf{TG}}$); by pullback composition, for this it suffices to show that the individual squares (1), (2), (4), (5) are pullbacks. Square (1) is a pullback by applying pullback decomposition to the left part of Fig. 19, where the solid square $K_\mathrm{b} \to L_\mathrm{b} \to L_\mathrm{m} \leftarrow K_\mathrm{m} \leftarrow K_\mathrm{b}$, which is decomposed, is a pullback by definition of the context extension of an effect-oriented rule. That square (2) is a pullback follows similarly, using the fact that $G \cap R_\mathrm{b} = K_\mathrm{b}$ (which holds by assumption).

The arguments for squares (4) and (5) being pullbacks are analogous; we provide the one for (4): Let $x \in L^{G'}_\mathrm{ind} \cap K^G_\mathrm{ind}$; we have to show that $x \in K^{G,L}_\mathrm{ind}$. Assume that this is not the case. By construction of square (3) this means that $x \in K^{G,R}_\mathrm{ind} \setminus K_\mathrm{b}$; in particular, $x \in R_\mathrm{m} \cap G$. However, $x \in L^{G'}_\mathrm{ind}$ also means $x \in L_\mathrm{m} \cap G$. Together this implies $x \in R_\mathrm{m} \cap L_\mathrm{m} \subseteq K_\mathrm{m}$ (which holds by assumption), which implies $x \in L^{G'}_\mathrm{ind} \cap K_\mathrm{m} = K^{G,L}_\mathrm{ind}$—a contradiction.

To show that p^G_ind is a pure rule, let $x \in L^G_\mathrm{ind} \cap R^G_\mathrm{ind}$; we have to show that $x \in K^G_\mathrm{ind}$. Assume $x \notin K^G_\mathrm{ind}$. Then $x \in L^{G'}_\mathrm{ind}$ and $x \in R_\mathrm{m}$; the first implies $x \in G$, which, together with $x \in R_\mathrm{m}$, implies $x \in K^{G,R}_\mathrm{ind}$. This, in turn, implies $x \in K^G_\mathrm{ind}$ (by construction)—a contradiction. □

Proof of Proposition 3. First, let $x \in L_\mathrm{m} \setminus K_\mathrm{m}$. In any case, $L_\mathrm{m} \cap R_\mathrm{m} \subseteq K_\mathrm{m}$ implies that $x \notin R_\mathrm{m}$. This means that $x \notin G$ implies $x \notin H$. If $x \in G$, then $x \in L_\mathrm{m}$ implies $x \in L^{G'}_\mathrm{ind}$ which, in turn, implies $x \in L^G_\mathrm{ind}$. Furthermore, $x \notin K_\mathrm{m}$ implies $x \notin K^{G,L}_\mathrm{ind}$; and $x \notin R_\mathrm{m}$ implies $x \notin K^{G,R}_\mathrm{ind}$. Together, these imply $x \notin K^G_\mathrm{ind}$, which means $x \notin H$.

More obviously, $x \in R_\mathrm{m}$ immediately implies $x \in R^G_\mathrm{ind}$ and therefore $x \in H$ for every $x \in R_\mathrm{m} \setminus K_\mathrm{b}$. □

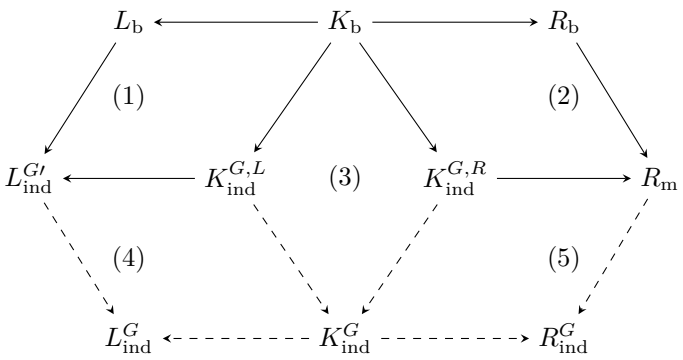


Figure B.52: Second part of construction of the induced rule – squares renamed

Appendix B.2. Proofs from Section 4

Proof of Proposition 4. For proving this statement, we just count the number of elements that are added in the individual steps of Construction 1.

The first step adds $2\times$ $|T \setminus P|$ nodes to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$. Per edge $e \in E_T \setminus E_P$, Steps 2. and 3. add maximally two edges to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$; thus, we maximally add $4|E_T \setminus E_P|$ edges to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$ there.

For Step 4., we have to consider the four different interaction principles from Definition 10. First, in the case of CreationNeg, given a constraint $\neg\exists\, B$, in a single application we add one node and $|B \setminus P|$ edges to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$. We have to apply that interaction principle for every purely negative constraint of $\mathcal{C}$—a set we denote as $\mathcal{C}_\mathrm{neg}$—and $|B \setminus P|$-times per such constraint (once for every element). This leads to adding maximally

$$|\mathcal{C}_\mathrm{neg}||B_\mathrm{max} \setminus P| \tag{CreationNegNodes}$$

nodes and maximally

$$|\mathcal{C}_\mathrm{neg}||B_\mathrm{max} \setminus P|^2 \tag{CreationNegEdges}$$

edges to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$, where B_max is the largest graph occurring in any of the constraints in $\mathcal{C}$.

Second, in the case of CreationImpl, given a constraint $\exists\, D \implies \vee_{i=1,\dots,t}\exists B_i$, in a single application we add $t+1$ nodes and $t+1+\sum_{i=1}^{t}|B_i \setminus P|$ edges to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$. We have to apply that interaction principle for every implication constraint of $\mathcal{C}$—a set we denote as $\mathcal{C}_\mathrm{impl}$—and $|D \setminus P|$-times per such constraint. This leads to adding maximally

$$(t_\mathrm{max}+1)|\mathcal{C}_\mathrm{impl}||B_\mathrm{max} \setminus P| \tag{CreationImplNodes}$$

nodes and maximally

$$(t_\mathrm{max}+1+t_\mathrm{max}\cdot|B_\mathrm{max} \setminus P|)|\mathcal{C}_\mathrm{impl}||B_\mathrm{max} \setminus P| \tag{CreationImplEdges}$$

edges to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$, where, again, B_max is the largest graph occurring in any of the constraints in $\mathcal{C}$ and t_max is the maximal length of any disjunction appearing in such a constraint.

Third, in the case of DeletionPos, given a constraint $\vee_{i=1,\dots,t}\exists\, B_i$, in a single application we add maximally t nodes and $t+\sum_{i=1}^{t}|B_i \setminus P|$ edges to $\mathrm{AD}_{T\supseteq P,\mathcal{C}}$. We have to apply that interaction principle for every positive constraint of $\mathcal{C}$—a set we denote as $\mathcal{C}_\mathrm{pos}$— for every element outside of P of every graph occurring in it. This leads to maximally adding

$$t^2_\mathrm{max}|\mathcal{C}_\mathrm{pos}||B_\mathrm{max} \setminus P| \tag{DeletionPosNodes}$$

nodes and maximally

$$(t_\mathrm{max}+t_\mathrm{max}|B_\mathrm{max} \setminus P|)t_\mathrm{max}|\mathcal{C}_\mathrm{pos}||B_\mathrm{max} \setminus P| \tag{DeletionPosEdges}$$

edges to $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$.

Fourth, in the case of DeletionImpl, given a constraint $\exists\, D \implies \vee_{i=1,\dots,t} \exists B_i$, in a single application we add maximally t nodes and $t + 1 + \sum_{i=1}^{t} |B_i \setminus P|$ edges to $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$. We have to apply that interaction principle for every implication constraint of $\mathcal{C}$, for every element outside of P appearing in one of the graphs of the consequent of the constraint (i.e., in one of the graphs that form the disjunction). This leads to maximally adding

$$t_{\max}^2 |\mathcal{C}_{\mathrm{Impl}}| |B_{\max} \setminus P| \tag{DeletionImplNodes}$$

nodes and maximally

$$(t_{\max} + 1 + t_{\max} |B_{\max} \setminus P|) t_{\max} |\mathcal{C}_{\mathrm{Impl}}| |B_{\max} \setminus P| \tag{DeletionImplEdges}$$

edges to $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$.

Summing up generously proves the overall claim. □

Proof of Theorem 5. We prove the theorem by distinguishing the type of n and the applicable interaction principles.

Case 1: ElementCreation. First, let n be an ElementCreation node and let c be such that x appears negatively bound in c, either as an isolated node or as an edge. For this, $c = \neg\exists\,(D, \mathsf{true})$ is either purely negative, x appears in its literal and CreationNeg is the applicable interaction principle (Case 1a)), or $c = \exists\,(D, \mathsf{true}) \implies \vee_{i=1,\dots,t} \exists\,(B_i, \mathsf{true})$ is in implication format, x appears in the antecedent D of c and CreationImpl is the applicable interaction principle (Case 1b)). For the following, let $G \in \mathbf{Sub}(T \supseteq P)$ be such that $G \models n$, i.e., $x \in G$.

In Case 1a), $c = \neg\exists\,(D, \mathsf{true})$ and $G \models c$ if and only if there exists some element $x \neq y \in D$ such that $y \notin G$ if and only if there exists some element $x \neq y \in D \cap (T \setminus P)$ such that $y \notin G$. By definition of CreationNeg and x being an edge or a node isolated in D, the principle CreationNeg inserts as consequences to n in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ an Or node with outgoing edges to all ElementDeletion nodes that represent edges or isolated nodes from $(D \cap (T \setminus P)) \setminus \{x\}$. Also by definition, $\mathrm{post}(\mathsf{n}, c)$ consists of this part of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$. Thus, G satisfies $\mathrm{post}(\mathsf{n}, c)$ if and only if it does not contain one of the edges or isolated nodes from $(D \cap (T \setminus P)) \setminus \{x\}$. Because G is a graph, i.e., contains adjacent nodes for all its edges, both statements are equivalent (in particular, if G misses a non-isolated node $(D \cap (T \setminus P)) \setminus \{x\}$, it also misses an edge from that set).

In Case 1b), $c = \exists\,(D, \mathsf{true}) \implies \vee_{i=1,\dots,t} \exists\,(B_i, \mathsf{true})$ and $G \models c$ if and only if there exists some element $x \neq y \in D \cap (T \setminus P)$ such that $y \notin G$ or $B_{i_0} \subseteq G$ for some $i_0 \in \{1, \dots, t\}$. By definition of CreationImpl and x being an edge or a node isolated in D, the principle CreationImpl inserts as consequences to n in $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ an Or node with (i) outgoing edges to all ElementDeletion nodes that represent edges or isolated nodes from $(D \cap (T \setminus P)) \setminus \{x\}$ and (ii) edges to And nodes that each, for one graph B_i that does not contain x, reference all ElementCreation nodes that correspond to edges and nodes isolated in $B_i \cap (T \setminus P)$. Also by definition, $\mathrm{post}(\mathsf{n}, c)$ consists of this part of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$. Thus, G satisfies $\mathrm{post}(\mathsf{n}, c)$ if and only if it (i) does not contain one of the edges or isolated nodes from $(D \cap (T \setminus P)) \setminus \{x\}$ or (ii) contains all of the edges and isolated nodes of $B_{i_0} \cap (T \setminus P)$ for one $i_0 \in \{1, \dots, t\}$. Because G is a graph, i.e., contains adjacent nodes for all its edges, and $P \subseteq G$ both statements are equivalent (in particular, if G misses a non-isolated node $(D \cap (T \setminus P)) \setminus \{x\}$, it also misses an edge from that set, and if G contains all edges and isolated nodes from $B_{i_0} \cap (T \setminus P)$, it contains B_{i_0}).

Case 2: ElementDeletion. Now, let n be an ElementDeletion node and let c be such that x appears positively bound in c, (in this case, as an arbitrary node or edge). For this, $c = \vee_{i=1,\dots,t} \exists\,(B_i, \mathsf{true})$ is either purely positive, x appears in (at least) one of its literals and DeletionPos is the applicable interaction principle (Case 2a)), or $c = \exists\,(D, \mathsf{true}) \implies \vee_{i=1,\dots,t} \exists\,(B_i, \mathsf{true})$ is in implication format, x appears in (at least) one of its positive literals and DeletionImpl is the applicable interaction principle (Case 2b)). For the following, let $G \in \mathbf{Sub}(T \supseteq P)$ be such that $G \models n$, i.e., $x \notin G$.

In Case 2a), $c = \vee_{i=1,\dots,t} \exists\,(B_i, \mathsf{true})$ and $x \in B_{i_0}$ for (at least) one $i_0 \in \{1,\dots,t\}$. Therefore, $G \models c$ if and only if $B_j \subseteq G$, where $j \in \{1,\dots q\}$ and this set indexes those graphs $B_i, i = 1,\dots,t$, in which x does not occur. By definition of DeletionPos and x being an element of one of the graphs B_i, this interaction principle is triggered—either directly for n (in case n corresponds to an edge or a node isolated in one of the graphs B_i) or for at least one ElementDeletion node n' of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ such that n' corresponds to an edge e that is adjacent to x and that occurs in one of the graphs B_i. In both cases, $\mathrm{post}(\mathsf{n}, c)$ first contains n and, whenever x is a node, references to the ElementDeletion nodes representing the edges adjacent to x; if x is a node not isolated in B_{i_0}, a node n' representing an edge also appearing in B_{i_0} is among these nodes. The application of principle DeletionPos inserts an Or node and references to q And nodes that each reference w_j ElementCreation nodes representing all isolated nodes and edges appearing in $B_j \cap (T \subseteq P)$, where $j \in \{1,\dots,q\}$ and this set indexes those graphs from $B_1,\dots,B_t$ that do not contain x. By definition, this structure is part of $\mathrm{post}(\mathsf{n}, c)$—either n directly references the starting Or node or (at least) one ElementDeletion node n', representing an edge adjacent to x and being itself referenced by n. In either case, since $x \notin G$ implies $e \notin G$ for every edge e adjacent to x, $G \models \mathrm{post}(\mathsf{n}, c)$ if and only if G contains all isolated nodes and edges from $B_j \cap (T \setminus P)$ for one $j \in \{1,\dots,q\}$ as introduced above. Again, by G being a graph and containing P, this is equivalent to $B_j \subseteq G$ for one such graph B_j.

In Case 2b), $c = \exists\,(D, \mathsf{true}) \implies \vee_{i=1,\dots,t} \exists\,(B_i, \mathsf{true})$ and $x \in B_{i_0}$ for (at least) one $i_0 \in \{1,\dots,t\}$. Therefore, $G \models c$ if and only if $D \not\subseteq G$ or $B_j \subseteq G$, where $j \in \{1,\dots q\}$ and this set indexes those graphs $B_i, i = 1,\dots,t$, in which x does not occur. By definition of DeletionImpl and x being an element of one of the graphs B_i, this interaction principle is triggered—either directly for n (in case n corresponds to an edge or a node isolated in one of the graphs B_i) or for at least one ElementDeletion node n' of $\mathrm{AD}_{T \supseteq P, \mathcal{C}}$ such that n' corresponds to an edge e that is adjacent to x and that occurs in one of the graphs B_i. In both cases, $\mathrm{post}(\mathsf{n}, c)$ first contains n and, whenever x is a node, references to the ElementDeletion nodes representing the edges adjacent to x; if x is a node not isolated in B_{i_0}, a node n' representing an edge also appearing in B_{i_0} is among these nodes. The application of principle DeletionImpl inserts an Or node that (i) references each ElementDeletion node that corresponds to an isolated node or edge from $D \cap (T \setminus P)$ and (ii) references to q And nodes that each reference w_j ElementCreation nodes representing all isolated nodes and edges appearing in $B_j \cap (T \subseteq P)$, where $j \in \{1,\dots,q\}$ and this set indexes those graphs from $B_1,\dots,B_t$ that do not contain x. By definition, this structure is part of $\mathrm{post}(\mathsf{n}, c)$—either n directly references the starting Or node or (at least) one ElementDeletion node n', representing an edge adjacent to x and being itself referenced by n. In either case, since $x \notin G$ implies $e \notin G$ for every edge e adjacent to x, $G \models \mathrm{post}(\mathsf{n}, c)$ if and only if G does not contain one of the isolated nodes or one of the edges from D or if G contains all isolated nodes and edges from $B_j \cap (T \setminus P)$ for one $j \in \{1,\dots,q\}$ as introduced above. Again, by G being a graph and containing P, this is equivalent to $D \not\subseteq G$ or $B_j \subseteq G$ for one such graph B_j. □

Appendix B.3. Proofs from Section 5

Proof of Proposition 6. We have to show that (i) the objects occurring in $p^F_{\mathcal{C},\mathrm{m}}$ are actually graphs (i.e., there are no dangling edges); (ii) $p^F_{\mathcal{C},\mathrm{m}}$ is a rule with p as a subrule with coinciding interface; and (iii) K^F_{m} satisfies the conditions on context extensions, i.e., $K^F_{\mathrm{m}} \subseteq L^F_{\mathcal{C},\mathrm{m}}$, $K^F_{\mathrm{m}} \cap L = K$ and $K^F_{\mathrm{m}} \cap R = K$, and $L^F_{\mathcal{C},\mathrm{m}} \cap R^F_{\mathcal{C},\mathrm{m}} \subseteq K^F_{\mathrm{m}}$. Throughout the proof, let χ_F denote the characteristic function of the configuration $F \subseteq \mathcal{F}$.

For (i) and (ii), let e be an edge in $X^F_{\mathcal{C},\mathrm{m}}$, where $X \in \{L, K, R\}$. Then, by definition, $\chi_F \models b^{X_\mathcal{C}}_{\mathcal{C}}(e)$. We show that for every node n that is adjacent to e (in T) it holds that $b^{X_\mathcal{C}}_{\mathcal{C}}(e) \implies b^{X_\mathcal{C}}_{\mathcal{C}}(n)$; in the following, let n denote such a node. For the proof, recall that $b^{X_\mathcal{C}}_{\mathcal{C}}$ is defined as a disjunction $\varphi^{X_\mathcal{C}}_{\mathrm{pres}} \vee \varphi^{X_\mathcal{C}}_{\mathrm{graph}} \vee \varphi^{X_\mathcal{C}}_{\mathrm{orig}}$. Regarding $\varphi^{X_\mathcal{C}}_{\mathrm{orig}}$, it is clear that $\varphi^{X_\mathcal{C}}_{\mathrm{orig}}(e) \implies \varphi^{X_\mathcal{C}}_{\mathrm{orig}}(n)$ because the originally given rule p consists of graphs and $\varphi^{X_\mathcal{C}}_{\mathrm{orig}}$ just encodes membership in that rule. For $\varphi^{X_\mathcal{C}}_{\mathrm{pres}}$, we have to consider the cases $X = L$ and $X = R$ (because $\varphi^{K_\mathcal{C}}_{\mathrm{pres}} \equiv \mathsf{false}$). Regarding L, $\varphi^{L_\mathcal{C}}_{\mathrm{pres}}(e) \implies \varphi^{L_\mathcal{C}}_{\mathrm{pres}}(n) \vee \varphi^{L_\mathcal{C}}_{\mathrm{graph}}(n)$ by definition of $\varphi^{L_\mathcal{C}}_{\mathrm{graph}}$. Regarding R, point 2. in the construction of activation diagrams (see Construction 1) ensures the existence of an edge from the ElementCreation node that corresponds to e to the ElementCreation node that corresponds to n. By definition of $\varphi^{R_\mathcal{C}}_{\mathrm{pres}}$, this ensures that $\varphi^{R_\mathcal{C}}_{\mathrm{pres}}(e)$ is a disjunct of the disjunction $\varphi^{R_\mathcal{C}}_{\mathrm{pres}}(n)$; in particular, $\varphi^{R_\mathcal{C}}_{\mathrm{pres}}(e) \implies \varphi^{R_\mathcal{C}}_{\mathrm{pres}}(n)$. Finally, $\varphi^{X_\mathcal{C}}_{\mathrm{graph}}$ evaluates to false for all edges e. Thus, we have shown that $X^F_{\mathcal{C},\mathrm{m}}$

is a graph for every $X \in \{L, K, R\}$. Moreover, by construction, it also easily follows that $b_{\mathcal{C}}^{K_c} \implies b_{\mathcal{C}}^{L_c}$ and $b_{\mathcal{C}}^{K_c} \implies b_{\mathcal{C}}^{R_c}$; i.e., $K_{\mathcal{C},\mathrm{m}}^F \subseteq L_{\mathcal{C},\mathrm{m}}^F$ and $K_{\mathcal{C},\mathrm{m}}^F \subseteq R_{\mathcal{C},\mathrm{m}}^F$. Since $\varphi_{\mathrm{graph}}^{K_c} = \varphi_{\mathrm{pres}}^{K_c} = \mathsf{false}$ and $\varphi_{\mathrm{orig}}^{K_c}$ encodes membership in the interface K of the original rule, it also holds that $K_{\mathcal{C},\mathrm{m}}^F = K$. Summarizing, $p_{\mathcal{C},\mathrm{m}}^F$ is a rule with interface K. To see that $p \subseteq p_{\mathcal{C},\mathrm{m}}^F$, the condition $\varphi_{\mathrm{orig}}^{X_c}$ ensures that $X \subseteq X_{\mathcal{C},\mathrm{m}}^F$ for every $X \in \{L, K, R\}$. Thus, we only have to show that $K_{\mathcal{C},\mathrm{m}}^F \cap L = K = K_{\mathcal{C},\mathrm{m}}^F \cap R$, which trivially holds by $K = K_{\mathcal{C},\mathrm{m}}^F$.

For (iii), by construction it is immediate that $K \subseteq K_{\mathrm{m}}^F \subseteq L_{\mathcal{C},\mathrm{m}}^F$. Regarding the requirements on intersections, $K_{\mathrm{m}}^F \cap L = K$ follows from $\overline{L^-} \cap L = \emptyset$, whereas $K_{\mathrm{m}}^F \cap R = K$ is ensured by Eq. 4. Finally, we have to show that $L_{\mathrm{m}} \cap R_{\mathrm{m}} \subseteq K_{\mathrm{m}}^F$. We show this by case analysis; thus, let $x \in L_{\mathrm{m}} \cap R_{\mathrm{m}}$. First, *if* $x \in L \cap R$, also $x \in K \subseteq K_{\mathrm{m}}^F$. Second, *if* $x \in R_{\mathrm{m}} \setminus R$, it must hold that $\chi_F \models \varphi_{\mathrm{pres}}^{R_c}(x)$. By definition of ψ_{cons} (Eq. 3), this means that $\chi_F \not\models \varphi_{\mathrm{pres}}^{L_c}(x)$. Thus, either $\chi_F \models \varphi_{\mathrm{orig}}^{L_c}(x)$ or $\chi_F \models \varphi_{\mathrm{graph}}^{L_c}(x)$. If $\chi_F \models \varphi_{\mathrm{orig}}^{L_c}(x)$, Eq. 3 also ensures that $\chi_F \models \varphi_{\mathrm{orig}}^{K_c}(x)$, which means $x \in K_{\mathrm{m}}^F$. If $\chi_F \models \varphi_{\mathrm{graph}}^{L_c}(x)$, Eq. 4 implies $x \in K_{\mathrm{m}}^F$ (by $\chi_F \not\models \varphi_{\mathrm{orig}}^{R_c}(x)$). Third, *if* $x \in L_{\mathrm{m}} \setminus L$, Eq. 3 ensures that it cannot happen that $\chi_F \models \varphi_{\mathrm{pres}}^{L_c}(x)$ and $\chi_F \models \varphi_{\mathrm{orig}}^{R_c}(x)$ or $\chi_F \models \varphi_{\mathrm{pres}}^{R_c}(x)$. Thus, the only possibility for an element $x \in (L_{\mathrm{m}} \setminus L) \cap R_{\mathrm{m}}$ is that $\chi_F \models \varphi_{\mathrm{graph}}^{L_c}(x)$, a case we already considered above. □

Proof of Lemma 7. In the situation of the lemma, let $vp_{\mathcal{C},\mathrm{e}}^F$ be a variant of $vp_{\mathcal{C}}$, induced by some valid configuration F. By Proposition 6, $vp_{\mathcal{C}}$ is an effect-oriented rule with p as base rule and $p_{\mathcal{C},\mathrm{m}}^F$ as maximal rule. Therefore, by Proposition 2, the instantiation $p_{\mathcal{C},\mathrm{ind}}^{F,G} = \left(L_{\mathcal{C},\mathrm{ind}}^{F,G} \supseteq K_{\mathcal{C},\mathrm{ind}}^{F,G} \subseteq R_{\mathcal{C},\mathrm{ind}}^{F,G}\right)$ exists. By construction, it is ensured that $L_{\mathcal{C},\mathrm{ind}}^{F,G} \subseteq G$ and $R_{\mathcal{C},\mathrm{ind}}^{F,G} \cap G \subseteq K_{\mathcal{C},\mathrm{ind}}^{F,G}$. Therefore, by [25, Proposition 6], a transformation $G \Longrightarrow_{p_{\mathcal{C},\mathrm{ind}}^{F,G}} H$, to some graph H, in $\mathbf{Sub}(T \supseteq P)$ exists if and only if such a transformation exists in $\mathbf{Graph}_{\mathbf{TG}}^{\mathbf{fin}}$. Because $L_{\mathcal{C},\mathrm{ind}}^{F,G}$ is a subgraph of G (i.e., the match is monic), it is well known that this is the case if and only if the dangling edge condition is satisfied [34, Fact 3.11].

Thus, let $n \in G$ be a node that $p_{\mathcal{C},\mathrm{ind}}^{F,G}$ declares to delete, i.e., $n \in L_{\mathcal{C},\mathrm{ind}}^{F,G} \setminus K_{\mathcal{C},\mathrm{ind}}^{F,G}$. If $x \in L$ (i.e., x stems from the LHS of the original rule), Proposition 6 and Proposition 2 together imply that $x \notin K$, i.e., x is already deleted by the original rule. Then x can be deleted (i.e., it has no adjacent edges in G) because we assume $L \subseteq G$ to be a match for the original rule. If $x \notin L$, either $\chi_F \models \varphi_{\mathrm{pres}}^{L_c}(x)$ or $\chi_F \models \varphi_{\mathrm{graph}}^{L_c}$. If the former is the case, the construction of activation diagrams ensures that $\chi_F \models \varphi_{\mathrm{pres}}^{L_c}(e)$ for every edge e adjacent to x (in T): $\varphi_{\mathrm{pres}}^{L_c}(x)$ is a disjunct of every such disjunction $\varphi_{\mathrm{pres}}^{L_c}(e)$. If the latter were the case, i.e., $\chi_F \models \varphi_{\mathrm{graph}}^{L_c}(x)$, $x \in G$ and $x \notin K_{\mathcal{C},\mathrm{ind}}^{F,G}$, Eq. 4 implies that $\chi_F \models \varphi_{\mathrm{orig}}^{R_c}$, i.e., $x \in R \setminus K$. But this would mean that $x \in R \cap G \neq K$, which contradicts our assumption that $L \subseteq G$ is a match for the original rule. Hence, this last case cannot happen. This completes the proof of the applicability of $p_{\mathcal{C},\mathrm{ind}}^{F,G}$ and shows the final statement of the lemma about the non-deletion of nodes $x \in L_{\mathcal{C},\mathrm{ind}}^{F,G}$ for which $\chi_F \models \varphi_{\mathrm{graph}}^{L_c}(x)$.

It remains to show the statement about transformation t_{ind} just affecting elements from $T \setminus P$. For this, if $x \in L_{\mathcal{C},\mathrm{ind}}^{F,G}$, in particular we have $\chi_F(x) \models \varphi_{\mathrm{orig}}^{L_c}(x) \vee \varphi_{\mathrm{pres}}^{L_c}(x) \vee \varphi_{\mathrm{graph}}^{L_c}(x)$. If $\chi_F(x) \models \varphi_{\mathrm{orig}}^{L_c}(x)$, we have $x \in T \setminus P$ or $\chi_F(x) \models \varphi_{\mathrm{orig}}^{K_c}(x)$ by assumption on the original rule p; that is, either $x \in T \setminus P$ or $x \in K_{\mathcal{C},\mathrm{m}}^F$ (and, by $x \in G$, also $x \in K_{\mathcal{C},\mathrm{ind}}^{F,G}$), i.e., x is not deleted by $p_{\mathcal{C},\mathrm{ind}}^{F,G}$. If $\chi_F(x) \models \varphi_{\mathrm{pres}}^{L_c}(x)$, we have $x \in L^- \setminus L$, which implies $x \in T \setminus P$ by the construction of activation diagrams (where only $\mathsf{ElementDecision}$ nodes are added for elements from $T \setminus P$). If $\chi_F(x) \models \varphi_{\mathrm{graph}}^{L_c}(x)$, we also have $x \in K_{\mathcal{C},\mathrm{ind}}^{F,G}$ by the considerations above. The arguments for the right-hand side are completely analogous (there, we just have to consider $\chi_F(x) \models \varphi_{\mathrm{orig}}^{R_c}(x)$ and $\chi_F(x) \models \varphi_{\mathrm{pres}}^{R_c}(x)$). □

Proof of Theorem 8. We prove the statement by case analysis along the different forms of constraints as introduced in Assumptions 1 and 2.

Purely positive case. Let $c = \exists\,(B_1, \mathsf{true}) \vee \cdots \vee \exists\,(B_t, \mathsf{true})$ and $G \models c$. The only way in which an application of $vp_{\mathcal{C},\mathrm{e}}^F$ could introduce a violation of c is by deleting some element $x \in \bigcup_{i=1\ldots t} B_i \cap G$. For this, by definition of transformations and consistency-preserving variants, $\chi_F \models b_{\mathcal{C}}^L(x)$ but $x \notin K_{\mathrm{m}}^F$. In particular,

$\chi_F \models \varphi^{Lc}_{\text{pres}}(x)$ or $\chi_F \models \varphi^{Lc}_{\text{orig}}(x)$. (We do not need to consider the case $\chi_F \models \varphi^{Lc}_{\text{graph}}(x)$ by Lemma 7.) Moreover, also by Lemma 7, such an element x must stem from $T \setminus P$.

By construction of the activation diagram $\text{AD}_{T \supseteq P, \mathcal{C}}$, the interaction principle DeletionPos has been applied to the ElementDeletion node -x, corresponding to x. That is, in $\text{AD}_{T \supseteq P, \mathcal{C}}$ there is an outgoing edge from -x to an Or node that has outgoing edges to w And-nodes where, in turn, each And node has outgoing edges to a set of ElementCreation nodes that correspond to the relevant elements of one of the graphs B_i that does not contain x (compare Fig. 26). Let the Or node be annotated as OR_q and its outgoing edges (and, consequently, the ones of the And nodes) as $or^q_1, \ldots, or^q_w$.

If $\chi_F \models \varphi^{Lc}_{\text{pres}}(x)$, there is some $o^i_j \in F$ that appears as a disjunct in $\varphi^{Lc}_{\text{pres}}$ (to render that evaluation true). Either or^i_j directly labels an incoming edge of $-x$ or x is an edge and or^i_j labels an incoming edge of one of its adjacent nodes. In either case that means that or^i_j directly triggers OR_q (compare Definition 13), which results in $o^i_j \implies O_q$ being one conjunct of the cross-tree constraint ψ_{path}. If $\chi_F \models \varphi^{Lc}_{\text{orig}}(x)$, $\mathit{root} \implies O_q$ is one conjunct of the cross-tree constraint ψ_{path}. Thus, whether $\chi_F \models \varphi^{Lc}_{\text{pres}}(x)$ or $\chi_F \models \varphi^{Lc}_{\text{orig}}(x)$, $\mathit{root} \in F$ or $o^i_j \in F$ and validity of F with regard to ψ_{path} imply $O_q \in F$. This, again by validity of F, implies $o^q_l \in F$ for (exactly) one $l \in \{1, \ldots, w\}$. With this, we have that either (i) $\chi_F \models \varphi^{Rc}_{\text{pres}}(y)$ for all ElementCreation nodes with incoming edge or^q_l or (ii) that $\varphi^{Rc}_{\text{pres}}(y) = \mathsf{false}$ for such an element y but $\varphi^{Rc}_{\text{orig}}(y) = \mathsf{true}$. In any case $\chi_F \models b^{Rc}_{\mathcal{C}}(y)$; that is, every such element y is part of the RHS $R^F_{\mathcal{C},\text{m}}$. Intuitively, this ensures that $vp^F_{\mathcal{C},\text{e}}$ creates one of the graphs $B_1, \ldots, B_t$. More formally, for the graph H, resulting from the transformation t, by Proposition 3 this means that $x \notin H$ but $R^F_{\mathcal{C},\text{m}} \subseteq H$, which means $H \models \text{post}(-x, c)$. Theorem 5 then guarantees that $H \models c$.

Purely negative case. Let $c = \neg\exists(D, \mathsf{true})$ and $G \models c$. The only way in which an application of $vp^F_{\mathcal{C},\text{e}}$ could introduce a violation of c is by creating some element $x \in D \setminus G$. For this, by definition of transformations and consistency-preserving variants, $\chi_F \models b^R_{\mathcal{C}}(x)$, in particular, $\chi_F \models \varphi^{Rc}_{\text{pres}}(x)$ or $\chi_F \models \varphi^{Rc}_{\text{orig}}(x)$, but $x \notin K^F_{\text{m}}$. Moreover, by Lemma 7, such an element x must stem from $T \setminus P$.

By construction of the activation diagram $\text{AD}_{T \supseteq P, \mathcal{C}}$, the interaction principle CreationNeg has been applied to the ElementCreation node +x, corresponding to x. That is, in $\text{AD}_{T \supseteq P, \mathcal{C}}$ there is an outgoing edge from +x to an Or node that has outgoing edges to u ElementDeletion nodes that correspond to the relevant elements of $D \cap (T \setminus P)$ (compare Fig. 24). Let the Or node be annotated as OR_q and its outgoing edges as $or^q_1, \ldots, or^q_u$.

If $\chi_F \models \varphi^{Rc}_{\text{pres}}(x)$, there is some $o^i_j \in F$ that appears as a disjunct in $\varphi^{Rc}_{\text{pres}}$ (to render that evaluation true). Either or^i_j directly labels an incoming edge of $+x$ or x is a node and or^i_j labels the ElementCreation node of one of its adjacent edges. In either case that means that or^i_j directly triggers OR_q (compare Definition 13), which results in $o^i_j \implies O_q$ being one conjunct of the cross-tree constraint ψ_{path}. If $\chi_F \models \varphi^{Rc}_{\text{orig}}(x)$, $\mathit{root} \implies O_q$ is one conjunct of the cross-tree constraint ψ_{path}. Thus, whether $\chi_F \models \varphi^{Rc}_{\text{pres}}(x)$ or $\chi_F \models \varphi^{Rc}_{\text{orig}}(x)$, $\mathit{root} \in F$ or $o^i_j \in F$ and validity of F with regard to ψ_{path} imply $O_q \in F$. This, again by validity of F, implies $o^q_l \in F$ for (exactly) one $l \in \{1, \ldots, u\}$. Let y be the element corresponding to the ElementDeletion node -y to which the edge o^q_l points. By definition, either o^q_l is one of the disjuncts of $\varphi^{Lc}_{\text{pres}}(y)$ or $y \in L \setminus K$, i.e., y is deleted by the originally given rule and $\varphi^{Lc}_{\text{orig}}(y) = \mathsf{true}$ and $\varphi^{Kc}_{\text{orig}}(y) = \mathsf{false}$. Moreover, if $\varphi^{Lc}_{\text{orig}}(y) = \mathsf{true}$, the cross-tree constraint ψ_{cons} ensures that $\chi_F \not\models \varphi^{Rc}_{\text{orig}}(y)$ and, with that, $\chi_F \not\models \varphi^{Kc}_{\text{orig}}(y)$. Thus, in any case, $\chi_F \models b^{Lc}_{\mathcal{C}}(y)$, i.e., $y \in L^F_{\mathcal{C},\text{m}}$ and $y \notin K$. Finally, either of $y \in L$ and $\chi_F \models \varphi^{Lc}_{\text{pres}}(y)$ implies that $y \notin K^F_{\text{m}}$. Proposition 3 then ensures that $y \notin H$.

Summarizing, intuitively, this ensures that $vp^F_{\mathcal{C},\text{e}}$ deletes one of the elements of D. More formally, for the graph H, resulting from the transformation t, by Proposition 3 this means that $x \in H$ and $H \models \text{post}(+x, c)$. Theorem 5 then guarantees that $H \models c$.

Implication case. Let $c = \exists(D, \mathsf{true}) \implies \exists(B_1, \mathsf{true}) \vee \cdots \vee \exists(B_t, \mathsf{true})$. Given that $G \models c$, the only way in which the graph H resulting from transformation t could violate c is that the transformation either creates an element of D or deletes an element of one of the graphs $B_1, \ldots, B_t$. The proof then proceeds completely analogous to the proofs for the purely positive and the purely negative case (combining both in each of the two possible situations), because the interaction principles for the implication case combine the

interaction principles for the positive and the negative case. □

Proof of Theorem 9. Given the situation of the theorem, we traverse the activation diagram $\mathrm{AD}_{C,T}$ to construct the configuration F; in that, we use the graph E to guide our decisions at points of potential variance. Throughout the proof, by F_{temp} we denote the current state of the configuration we build.

First, we set $F_{\text{temp}} \coloneqq \{root\}$. We have to arrive at a valid configuration F, i.e., at a set $F \subseteq \mathcal{F}$ such that that choice fulfils the properties required by the parent–child relationships (e.g., select a child of a selected Or-node) and such that the cross-tree constraints are satisfied, i.e., such that $F \models Q = \psi_{\text{cons}} \wedge \psi_{\text{path}}$.

For this, we iterate over the set of path constraints not satisfied by F_{temp}, i.e., the conjuncts q of ψ_{path} such that $\chi_{F_{\text{temp}}} \not\models q$, and the set of Or and Xor-nodes in F_{temp} for which no child has been selected yet. During that iteration, whenever there is a path constraint $q = o^i_j \implies O_{k_1} \wedge \cdots \wedge O_{k_r}$ or $q = root \implies O_{k_1} \wedge \cdots \wedge O_{k_r}$ such that $\chi_{F_{\text{temp}}} \not\models q$, we set

$$F_{\text{temp}} \coloneqq F_{\text{temp}} \cup \{O_{k_1}, \ldots, O_{k_r}\}.$$

After that, $F_{\text{temp}} \models q$. Moreover, this kind of insertion can never introduce violations of conjuncts z of ψ_{path} that have the format $z = O_j \implies o^{i_1}_{j_1} \vee \cdots \vee o^{i_s}_{j_s}$, because, by definition, in q the edge o^i_j directly triggers $O_{k_1}, \ldots, O_{k_r}$. That is, for every feature O_{k_l} newly added to F_{temp} the corresponding conjunct of the path constraint is satisfied.

In a state during the iteration where all path constraints are satisfied, i.e., where $\chi_{F_{\text{temp}}} \models \psi_{\text{path}}$, we check if there is any element $O_k \in F_{\text{temp}}$ such that no child o^k_j has been selected for O_k. By construction of activation diagrams, the Or node $\mathsf{OR_k}$ corresponding to O_k has a unique incoming edge starting at some ElementDecision node n. Importantly, we can assume that $E \models \mathsf{n}$ (in the sense of Definition 11), i.e., the element corresponding to n belongs to E in case n is an ElementCreation node and and it is not part of E in case n is an ElementDeletion node. At the beginning of our construction of F this is clear because then every $O_k \in F_{\text{temp}}$ has been added because of a conjunct of ψ_{path} of the format $root \implies O_k$ (because, at the very start, *root* is the only element of F_{temp}, i.e., the only element that can trigger the inclusion of further elements into F_{temp}). This means that the incoming edge of $\mathsf{OR_k}$ starts at an ElementDecision node n that corresponds to an action (creation or deletion) of the originally given rule p; and $E \models c_p$ by assumption.

Generally, whenever we have $E \models \mathsf{n}$ for some ElementDecision node n, the fact that $E \models \mathcal{C}$ implies that E also satisfies the consequences of n, i.e., $E \models \mathrm{post}(\mathsf{n}, c)$, where c refers to whatever constraint caused the insertion of the Or node under consideration (this is the one direction of the equivalence proven in Theorem 5). This means that, for every Or node $\mathsf{OR_k}$ such that E satisfies its originating ElementDecision node n, we can select a consequence o^k_j (putting it as child of O_k into F_{temp}) that witnesses that $E \models \mathrm{post}(\mathsf{n}, c)$ (if several such consequences exist, we randomly select one of them). (Intuitively, we select o^k_l according to the way in which the graph E satisfies the constraint c that triggered the application of the interaction principle *IP*.) In particular this means that, for every Or node $\mathsf{OR_q}$ such that the selected consequence o^k_l directly triggers $\mathsf{OR_q}$, we know that $E \models \mathsf{n'}$, where $\mathsf{n'}$ is the ElementDecision node at which $\mathsf{OR_q}$ originates. Summarizing, we (i) iteratively add features of the format O_k into F_{temp} in such a way that, upon termination, the cross-tree constraint ψ_{path} is satisfied; and (ii) iteratively choose children of the format o^k_j for these in such a way for every newly triggered conjunct $o^k_j \implies O_q$ it holds that $E \models \mathsf{n}$, where n is the ElementDecision node at which $\mathsf{OR_q}$ originates.

Since both the number of Or-nodes and the number of path constraints are finite, the overall iteration stops. The resulting configuration $F \subseteq \mathcal{F}$, by construction, satisfies all path constraints and the general constraints on feature configurations (children have parents, Or-nodes have children, ...). Regarding ψ_{cons}, through guiding our selection of elements $o^i_j \in F$ by (non-)existence in E we have ensured that $\chi_F \models \varphi^{L_\mathcal{C}}_{\text{pres}}(x) \implies x \notin E$ and $\chi_F \models \varphi^{R_\mathcal{C}}_{\text{pres}}(x) \implies x \in E$. Moreover $E \models c_p$ also means that $\chi_F \models \varphi^{X}_{\text{orig}}(x) \implies x \in E$ for $X \in \{K_\mathcal{C}, R_\mathcal{C}\}$ and $\chi_F \models \varphi^{L_\mathcal{C}}_{\text{orig}}(x) \implies x \notin E$. Together with the fact that E is a graph (and thus either contains an element x or not) this implies $\chi_F \models \psi_{\text{cons}}$. Summarizing, F is a valid configuration and, by construction, $E \models c_{vp^F_{\mathcal{C},\mathrm{e}}}$. □

Proof of Theorem 10. Given the situation of the above theorem, the existence of suitable atomic rules in $\mathcal{R}$ allows us to iteratively create resp. delete elements from $G \triangle H$ (where $\triangle$ denotes the componentwise

symmetric difference).

In more details, let $G_{\text{temp}} \coloneqq G$; as soon as $G_{\text{temp}} = H$ we stop. If $G_{\text{temp}} \neq H$, there exists $x \in G_{\text{temp}} \triangle H$ and $\text{at}_x^+ \in \mathcal{R}$ if $x \in H \setminus G_{\text{temp}}$ and $\text{at}_x^- \in \mathcal{R}$ if $x \in G_{\text{temp}} \setminus H$. Moreover, it holds that $H \models \mathcal{C} \wedge c_{\text{at}_x^+}$ (resp. $H \models \mathcal{C} \wedge c_{\text{at}_x^-}$). By Theorem 9, at_x^+ (resp. at_x^-) can be instantiated in such a way that $H \models \text{at}_x^{\text{cp}+}$ (resp. $H \models \text{at}_x^{\text{cp}-}$). Thus, we set G_{temp} to the result of applying that rule to G_{temp}. The fact that $H \models \text{at}_x^{\text{cp}+}$ (resp. $H \models \text{at}_x^{\text{cp}-}$) ensures that the size of $G_{\text{temp}} \triangle H$ is decreased by that application. Hence, the process terminates, resulting in H. □