Sustaining and Improving Graduated Graph Consistency: A Static Analysis of Graph Transformations

Jens Kosiol\textsuperscript{a}, Daniel Str"uber\textsuperscript{b}, Gabriele Taentzer\textsuperscript{a}, Steffen Zschaler\textsuperscript{c}

\textsuperscript{a}Philipps-Universit"at Marburg, Marburg, Germany
\textsuperscript{b}Radboud University, Nijmegen, the Netherlands
\textsuperscript{c}King’s College London, London, UK

Abstract
Where graphs are used for modelling and specifying systems, consistency is an important concern. To be a valid model of a system, the graph structure must satisfy a number of constraints. To date, consistency has primarily been viewed as a binary property: a graph either is or is not consistent with respect to a set of graph constraints. This has enabled the definition of notions such as constraint-preserving and constraint-guaranteeing graph transformations. Many practical applications—for example model repair or evolutionary search—implicitly assume a more graduated notion of consistency, but without an explicit formalisation only limited analysis of these applications is possible. In this paper, we introduce an explicit notion of consistency as a graduated property, depending on the number of constraint violations in a graph. We present two new characterisations of transformations (and transformation rules) enabling reasoning about the gradual introduction of consistency: while consistency-sustaining transformations do not decrease the consistency level, consistency-improving transformations strictly reduce the number of constraint violations. We show how these new definitions refine the existing concepts of constraint-preserving and constraint-guaranteeing transformations. To support a static analysis based on our characterisations, we present criteria for deciding which form of consistency-ensuring transformations is induced by the application of a transformation rule. We validate our contributions in the context of an application in search-based model engineering.

Keywords: Graph Consistency, Graph Transformation Systems, Graph Repair, Evolutionary Search

1. Introduction

Graphs and graph transformations \cite{1, 2} are a good means for system modelling and specification. Graph structures naturally relate to the structures typically found in many (computer) systems and graph transformations provide intuitive tools to specify dynamic changes of those structures. Graph transformations may be applied in model-based software and system development as well as model-driven engineering. The model transformation language and tool environment Henshin \cite{3, 4} uses graph transformation concepts for the transformation of models formulated with the Eclipse Modeling Framework (EMF).

In all of these application scenarios, it is important that the graphs used are consistent; that is, that their structures satisfy a set of constraints. Some constraints can be captured by typing graphs over so-called type graphs \cite{1}—these allow capturing basic structural constraints such as which kinds of nodes may be connected to each other. To allow the expression of further constraints, the theory of nested graph constraints has been introduced \cite{5}. A graph is considered consistent if it is correctly typed and satisfies all given constraints. Note that this notion of consistency is binary: a graph either is consistent or it is not consistent. It is impossible to distinguish different degrees of consistency.
In software engineering practice, it is often necessary to live with, and manage, a degree of inconsistency [6]. This requires tools and techniques for identifying, measuring, and correcting inconsistencies. In the field of graph-based specifications, this has led to many practical applications, where a more fine-grained notion of graph consistency is implicitly applied. For example, research in model repair has aimed to automatically produce graph-transformation rules that will gradually improve the consistency of a given graph. Such a rule may not make a graph completely consistent in one transformation step, but performing a sequence of such transformations will eventually produce a consistent graph (e.g., [7, 8, 9, 10]). In the area of search-based model engineering (e.g., [11, 12]), rules are required to be applicable to inconsistent graphs and, at least, not to produce new inconsistencies. In earlier work, we have shown how such rules can be generated at least with regard to multiplicity constraints [11]. However, in all of these works, the notion of “partial” graph consistency remains implicit. Without explicitly formalising this notion, it becomes difficult to reason about the validity of the rules generated or the correctness of the algorithm by which these rules were produced.

In this paper, we introduce a new notion of graph consistency as a graduated property. A graph can be consistent to a degree, depending on the number of constraint violations that occur in the graph. This conceptualisation allows us to introduce two new characterisations of graph transformations: a consistency-sustaining transformation does not decrease the overall consistency level, while a consistency-improving transformation strictly decreases the number of violations in a graph. We lift these characterisations to the level of graph-transformation rules, allowing rules to be characterised as consistency-sustaining and consistency-improving, respectively. We show how these definitions fit with the already established terminology of constraint-preserving and constraint-guaranteeing transformations/rules. Finally, we introduce formal criteria that allow checking whether a given graph-transformation rule is consistency-sustaining or consistency-improving w.r.t. constraints in specific forms.

Thus, the contributions of our paper are:

1. We present the first formalisation of graph consistency as a graduated property of graphs.
2. We present two novel characterisations of graph transformations and graph-transformation rules with regard to this new definition of graph consistency and show how these refine the existing terminology.
3. We present static analysis techniques for checking whether a graph-transformation rule is consistency-sustaining or consistency-improving.

This paper is an extended version of a previous conference paper [13]. In this paper, we extend the work in [13] by

- generalising our static analysis techniques to constraints of arbitrary nesting level,
- proposing a first technique to make transformation rules consistency-sustaining by computing suitable negative application conditions, and
- providing a practical validation (in addition to our running example), in which we study the degree of over- and under-approximation of our static analysis.

The remainder of this paper is structured as follows: We introduce a running example in Sect. 2 before outlining some fundamental terminology in Sect. 3. Section 4 introduces our new concepts and Sect. 5 discusses how graph-transformation rules can be statically analysed for these properties. In Sect. 6, we validate this analysis technique. A discussion of related work in Sect. 7 concludes the paper. The proofs of all results in this paper can be found in Appendix A.

2. Example

Consider class responsibility assignment (CRA, [14]), a standard problem in object-oriented software analysis. Given a set of features (methods, fields) with dependencies between them, the goal is to create
a set of classes and assign the features to classes so that a certain fitness function is maximised. The fitness function rewards the assignment of dependent features to the same class (cohesion), while punishing dependencies that run between classes (coupling) and solutions with too few classes. Solutions can be expressed as instances of the type graph shown in the left of Fig. 1. For realistic problem instances, an exhaustive enumeration of all solutions to find the optimal one is not feasible.

Recently, a number of works have addressed the CRA problem via a combination of graph transformation and meta-heuristic search techniques, specifically evolutionary algorithms [15, 16, 11]. An evolutionary algorithm uses genetic operators such as crossover and mutation to find optimal solution candidates in an efficient way. In this paper, we focus on mutation operators, which have been specified using graph-transformation rules in these works.

Figure 1 depicts four mutation rules for the CRA problem, taken from the available MDEOptimiser solution [17]. The rules are specified as graph-transformation rules [1] in the Henshin notation [3, 4]: Rule elements are tagged as delete, create, preserve or forbid, which denotes them as being included in the LHS, the RHS, in both rule sides, or a NAC. Rule assignFeature assigns a randomly selected as-yet-unassigned Feature to a Class. Rule createClass creates a Class and assigns an as-yet-unassigned Feature to it. Rule moveFeature moves a Feature between two Classes. Rule deleteEmptyClass deletes a Class to which no Feature is assigned.

Solutions in an optimisation problem such as the given one usually need to be consistent with regard to the constraints given by the problem domain. We consider three constraints for the CRA case:

\(c_1\) Every Feature is contained in at most one Class.
\(c_2\) Every Class contains at least one Feature.
\(c_3\) If a Feature \(f_1\) has a dependency to another Feature \(f_2\), and \(f_2\) is contained in a different class than \(f_1\), then \(f_1\) must have a dependency to a Feature \(f_3\) in the same class.

Constraints \(c_1\) and \(c_2\) come from Fleck et al.’s formulation of the CRA problem [15]. Constraint \(c_3\) can be considered a helper constraint (compare helper objectives [18]) that aims to enhance the efficiency of the search by formulating a constraint with a positive impact to the fitness function: Assigning dependent Features to the same Class is likely to improve coherence.

Given an arbitrary solution model (valid or invalid), mutations may introduce new violations. For example, applying moveFeature can leave behind an empty Class, thus violating \(c_2\). While constraint violations can potentially be removed using repair techniques [9, 7, 10], these can be computationally expensive and may involve strategies that lead to certain regions of the search space being preferred, threatening the efficiency of the search. Instead, it would be desirable to design mutation operators that impact consistency in a positive or at least neutral way. Each application of a mutation rule should contribute to some particular...
violations being removed, or at least ensure that the degree of consistency does not decrease. Currently, there exists no formal framework for identifying such rules. The established notions of constraint-preserving and constraint-guaranteeing rules [5] assume an already valid model or a transformation that removes all violations at once; both are infeasible in our scenario.

3. Preliminaries

Our new contributions are based on typed graph transformation systems following the double-pushout approach [1]. In the following, we recall the notions of typed graph, graph constraint, and graph transformation, in particular c-guaranteeing and c-preserving transformations. To be able to present criteria for consistency-sustainment and -improvement, we also recall the notions of conflicts and dependencies between rules.

3.1. Graphs and graph morphisms

We assume that all graphs, also the ones occurring in rules and constraints, are typed over a common type graph $TG$; that is, there is a class $Graph_{TG}$ of graphs typed over $TG$.

**Definition 1 (Graph).** A graph $G = (G_V, G_E, src_G, tgt_G)$ consists of a set $G_V$ of vertices (or nodes), a set $G_E$ of edges, and two maps $src_G, tgt_G : G_E \rightarrow G_V$ assigning the source and target to each edge, respectively. $e : x \rightarrow y$ denotes an edge $e$ with $src_G(e) = x$ and $tgt_G(e) = y$.

**Definition 2 (Graph morphism).** A graph morphism $f : G \rightarrow H$ consists of a pair of maps $f_V : G_V \rightarrow H_V$, $f_E : G_E \rightarrow H_E$ preserving the graph structure: For each edge $e : x \rightarrow y$ in $G$ we have an edge $f_E(e) : f_V(x) \rightarrow f_V(y)$ in $H$, i.e., we have $f_V \circ src_G = src_H \circ f_E$ and $f_V \circ tgt_G = tgt_H \circ f_E$. Morphism $f$ is injective (surjective) if $f_V$ and $f_E$ are injective (surjective). If $f$ is injective, we denote it with $f : G \Rightarrow H$. A pair of morphisms $f_1 : G_1 \rightarrow H$, $f_2 : G_2 \rightarrow H$ is jointly surjective, if every element of $H$ (node or edge) has a preimage under (at least) one of the morphisms.

A typed graph is a graph that is mapped to a given type graph. A mapping between two typed graphs over one and the same type graph has to be type-conformant.

**Definition 3 (Type graph, typed graph and typed morphism).** A type graph is a distinguished graph $TG = (TG_V, TG_E, src_{TG}, tgt_{TG})$. A typed graph $G = (G', t_G : G' \rightarrow TG)$ which is typed by $TG$ is a graph $G'$ together with a graph morphism $t_G$ from $G'$ to $TG$. A typed graph $G$ is also called instance graph of graph $TG$ and the morphism $t_G$ is called typing morphism.

Given a typed graph $TG$, a typed graph morphism $f : G \rightarrow H$ between typed graphs $G = (G', t_G)$ and $H = (H', t_H)$ is a graph morphism $f' : G' \rightarrow H'$ such that $t_G = t_H \circ f'$.

In the following, unless stated otherwise, we assume that a fixed type graph $TG$ is given and all graphs and graph morphisms are typed over $TG$. If we do not emphasise the typing of typed graphs and graph morphisms, we just call them graphs and graph morphisms.

3.2. Graph conditions and constraints

Nested graph constraints can be introduced as trees of injective graph morphisms; they offer a logic that is expressively equivalent to the classic first-order logic on graphs [5]. The more general notion of nested graph conditions allows for a recursive definition (and inductive proofs); conditions express properties of graph morphisms (instead of graphs). In our work, we address a relevant part of this logic, namely graph conditions in so-called alternating quantifier normal form (ANF) [10]. As the name implies, these are conditions where existential and universal quantifiers alternate. Every linear condition, i.e., every condition without a conjunction or disjunction, can be transformed into an equivalent condition in ANF [10, Fact 2]. As linear conditions are the only kind of conditions we address, we directly define conditions in ANF.
Definition 4 (Graph conditions and constraints (in ANF)). Given a graph $C_0$, existential and universal graph conditions in alternating quantifier normal form (ANF) over $C_0$ are defined recursively as follows:

1. true is a universal graph condition over $C_0$ and false is an existential one.

2. Let $a_1 : C_0 \hookrightarrow C_1$ be an injective morphism, which is not an isomorphism.
   
   (a) If $d$ is a universal graph condition over $C_1$, $\exists (a_1 : C_0 \hookrightarrow C_1, d)$ is an existential graph condition over $C_0$.
   
   (b) If $d$ is an existential graph condition over $C_1$, $\forall (a_1 : C_0 \hookrightarrow C_1, d)$ is a universal graph condition over $C_0$.

3. The set of all graph conditions in ANF over $C_0$ is the union of existential and universal ones over $C_0$.

   For a subcondition $Q (a_1 : C_{i-1} \hookrightarrow C_i, d)$, where $i \geq 1$, in such a condition, we say that the graph $C_i$ is bound by a universal resp. existential quantifier dependent on the kind of quantifier $Q$ is. An existential or universal graph constraint is an existential or universal graph condition over the empty graph $\emptyset$.

   The nesting level $nl$ of a graph condition in ANF $c$ is recursively defined by setting

   $$nl(true) = nl(false) := 0,$$

   $$nl(Q (a_1 : C_0 \hookrightarrow C_1, d)) := nl(d) + 1,$$

   where $Q \in \{\exists, \forall\}$.

   Given a graph condition $c$ in ANF over $C_0$, an injective morphism $p_0 : C_0 \hookrightarrow G$ satisfies $c$, written $p_0 \models c$, if the following applies:

   1. Every injective morphism satisfies true and none satisfies false.

   2. Morphism $p_0$ satisfies an existential graph condition in ANF $c = \exists (a_1 : C_0 \hookrightarrow C_1, d)$ if there exists an injective morphism $p_1 : C_1 \hookrightarrow G$ such that $p_0 = p_1 \circ a_1$ and $p_1$ satisfies $d$. It satisfies a universal graph condition in ANF $c = \forall (a_1 : C_0 \hookrightarrow C_1, d)$ if for every injective morphism $p_1 : C_1 \hookrightarrow G$ such that $p_0 = p_1 \circ a_1$, it holds that $p_1$ satisfies $d$.

   Given two conditions $c_1$ and $c_2$ in ANF over graph $C_0$, condition $c_1$ implies condition $c_2$, denoted as $c_1 \Rightarrow c_2$, if $p_0 \models c_1 \implies p_0 \models c_2$ for all injective graph morphisms $p_0 : C_0 \hookrightarrow G$. Two conditions $c_1$ and $c_2$ are equivalent, denoted as $c_1 \equiv c_2$, if $c_1 \Rightarrow c_2$ and $c_2 \Rightarrow c_1$. A graph $G$ satisfies a graph constraint in ANF $c$, denoted as $G \models c$, if the empty morphism $\emptyset \models G \models c$. Implication and equivalence of constraints are likewise reduced to the case of conditions.

   As graph conditions in ANF are the only ones discussed in this paper, we simply refer to them as (graph) conditions. Thus, unless stated otherwise, in the remainder of this paper, a (graph) condition is in alternating quantifier normal form. In the notation of graph conditions, we drop the domains of the involved morphisms and occurrences of true whenever they can unambiguously be inferred. For example, we write $\forall (C_1, \exists C_2)$ instead of $\forall (a_1 : \emptyset \hookrightarrow C_1, \exists (a_2 : C_1 \hookrightarrow C_2, true))$. When denoting a quantifier in an abstractly given condition, we will use $Q$ and $\bar{Q}$ to denote a quantifier and its dual. Thus, every condition $c$ (with $nl(c) \geq 2$) is of the form $Q(a_1 : C_0 \hookrightarrow C_1, Q(a_2 : C_1 \hookrightarrow C_2, \ldots))$, where $Q \in \{\exists, \forall\}$, $\exists = \forall$ and $\forall = \exists$.

   Note that, by definition, the above introduced semantics for graph conditions in ANF coincides with the one these conditions have when they are defined as general nested graph conditions. In particular, we implicitly allow for negation as classically: $\forall (C_1, d)$ is an abbreviation for the graph condition $\neg \exists (C_1, \neg d)$; for example, $\forall (C_1, false) \equiv \neg \exists C_1$.

   The following new lemma states that the sets of existential and universal conditions over a graph $C_0$ are semantically disjoint. This means that, in the following, we can meaningfully differentiate between these two subsets.
Lemma 1 (Non-equivalence of existential and universal conditions). Let \( c \) be an existential condition over a graph \( C_0 \) and \( c' \) be a universal one. Then \( c \not\equiv c' \).

The identity morphism \( id_{C_0} \) is a suitable morphism to show this lemma as it satisfies \( c' \) but not \( c \).

**Remark 1.** The alternating recursive definition of conditions in ANF allows us to use a special kind of induction to prove statements about them: The base case is to prove some property \( S_0 \) for \( \text{true} \) and some property \( S_3 \) for \( \text{false} \). For the induction step, we assume that \( S_3 \) holds for some existential condition \( d \) over a graph \( C_0 \) and show that the property \( S_3 \) holds for every universal condition of the form \( \forall (a_1 : C_0 \leftrightarrow C_1, d) \). Analogously, we assume that \( S_3 \) holds for some universal condition \( d \) over a graph \( C_1 \) and show that the property \( S_3 \) holds for every existential condition of the form \( \exists (a_1 : C_0 \rightarrow C_1, d) \). In this way, we have proved that every existential condition satisfies \( S_3 \) and every universal one satisfies \( S_0 \). We will heavily use this kind of induction in our proofs. Sometimes, we will start such an induction with the conditions of nesting level 1, i.e., with conditions of the forms \( \forall (C, \text{false}) \) and \( \exists (C, \text{true}) \).

### 3.3. Graph transformation

Graph transformation is the rule-based modification of graphs. The following definition recalls graph transformation as a double-pushout \([1]\). A rule mainly consists of two graphs: \( L \) is the left-hand side (LHS) of the rule representing a pattern that has to be found to apply the rule. After the rule application, a pattern equal to \( R \), the right-hand side (RHS), has been created. Their intersection \( K \), which is presupposed to constitute a graph, is the graph part that is not changed.

A graph transformation step between two instance graphs \( G \) and \( H \) along rule \( r \) is defined by first finding a match \( m \) of the left-hand side \( L \) of rule \( r \) in the current instance graph \( G \) such that \( m \) is structure-preserving and type-compatible, and second by constructing \( H \) in two steps (see Fig. 2): (1) building \( D := G \setminus m(L \setminus K) \), i.e., erasing all the graph items that are to be deleted, and (2) constructing \( H := D \cup n(R \setminus K) \), i.e., adding a new copy \( n(R \setminus K) \) of all the graph items that are to be created. Note that \( m \) has to fulfil the dangling condition, i.e., all adjacent graph edges of a graph node to be deleted have to be deleted by the rule as well, such that \( D \) becomes a graph. Additionally, a rule might be equipped with left and right application conditions that further restrict their application; these are allowed to be general (nested) graph conditions, also in this work. The described operational behaviour can be characterised (based on the notion of pushout coming from category theory) as follows:

**Definition 5 (Rule and transformation).** A plain rule \( r \) is defined by \( p = (L \leftarrow K \rightarrow R) \) with \( L, K, \) and \( R \) being graphs related by two injective graph morphisms. An application condition \( ac = (ac_L, ac_R) \) for \( p \) is a pair of nested graph conditions; \( ac_L \) is defined over \( L \) and \( ac_R \) is defined over \( R \). A rule \( r = (p, ac) \) consists of a plain rule \( p \) and an application condition \( ac \). If \( ac_R \) is equal to \( true \), we shortly write \( r = (p, ac_L) \) and call \( ac_L \) just application condition. A transformation (step) \( t : G \Rightarrow r,m H \) (also called rule application) which applies rule \( r \) at \( m \) to a graph \( G \) consists of the pushouts \((PO_1)\) and \((PO_2)\) as depicted in Fig. 2. Rule \( r \) is applicable at the injective morphism \( m : L \rightarrow G \), then called match, if \( m \models ac_L \), there exists a graph \( D \) such that the left square in Fig. 2 is a pushout, and if \( n \models ac_R \); morphism \( n \) is then called co-match. Morphisms \( g \) and \( h \) are called transformation morphisms and \( D \) the context object of the transformation. The track morphism \([19]\) of a transformation step \( t : G \\
Rightarrow r,m H \) is the partial morphism \( tr_t : G \rightarrow H \) defined by \( tr_t(x) = h(g^{-1}(x)) \) for \( x \in g(D) \) and undefined otherwise. A transformation sequence \( G_0 \Rightarrow r_1,m_1 G_1 \cdots \Rightarrow r_n,m_n G_n \) is shortly denoted as \( (G_{i-1} \Rightarrow r_{i,m_i}, G_i)_{1 \leq i \leq n} \).

Obviously, transformations interact with the validity of graph constraints. Two well-studied notions are constraint-guaranteeing and -preserving transformations \([5]\).

**Definition 6 (c-guaranteeing and c-preserving transformation).** Given a constraint \( c \), a transformation \( G \Rightarrow r,m H \) is c-guaranteeing if \( H \models c \). Such a transformation is c-preserving if \( G \models c \implies H \models c \). A rule \( r \) is c-guaranteeing (c-preserving) if every transformation via \( r \) is.
There is a well-known technique that, given a rule and a constraint, computes an application condition for the rule such that it becomes guaranteeing (or preserving) w.r.t. the constraint. This technique has been developed for general nested constraints in the context of \( \mathcal{M} \)-adhesive categories \[5\]. In the following, we recall a small part of this technique, namely how an (unnested) negative application condition is shifted over a rule. This suffices for the construction of consistency-sustaining application conditions that we develop in Sect. 5.1.

**Definition 7 (Shifting negative application conditions over rules).** Let a plain rule \( p = (L \leftarrow K \rightarrow R) \) and a right application condition \( ac_R = \neg \exists (c_R : R \leftarrow C, true) \equiv \forall (c_R : R \leftarrow C, false) \) be given. Construction \( \text{Left}(ac_R, p) \) shifts the application condition \( ac_R \) from \( R \) to \( L \) by applying rule \( p^{-1} = (R \leftarrow K \leftarrow L) \) (i.e., the inverse rule of \( p \)) to \( C \) at match \( c_R \) if applicable (as depicted in Fig. 3). The result is a left application condition \( ac_L = \neg \exists (c_L : L \leftarrow C', true) \), called negative application condition or shortly NAC, where \( c_L \) is the co-match of the transformation. If \( p^{-1} \) is not applicable at \( c_R \), the resulting application condition is \( false \).

This construction preserves the semantics of the application condition, i.e., the so computed negative application condition \( ac_L \) is characterised by the following property.

**Fact 1 ([5, Theorem 6]).** Given a rule \( r = (p, ac_R) \) with (unnested negative) right application condition \( ac_R = \neg \exists (c_R : R \leftarrow C, true) \), for any transformation \( G \Rightarrow_{r,m} H \) with co-match \( n \) via the plain rule \( p \), we have \( m \models ac_L \iff n \models ac_R \).

As also arbitrary right application conditions can be shifted (in a semantics-preserving manner; cf. [5, Theorem 6]) to the left-hand side of a rule, we assume all rules to be only equipped with left application conditions in the following.

3.4. Conflicts and dependencies

As we will present criteria for consistency-sustainment and consistency-improvement based on conflicts and dependencies of rules, we recall these notions here following [20, 21]. Intuitively, a transformation step causes a conflict on another one if it blocks the execution of the second one. A transformation step \( t_2 \) is dependent on another one \( t_1 \) if \( t_2 \) can only take place after \( t_1 \) has been performed.
Definition 8 (Conflict). Let a pair of transformations \((t_1, t_2) : (G \Rightarrow_{m_1, r_1} H_1, G \Rightarrow_{m_2, r_2} H_2)\) be given that apply rules \(r_i = (L_i \leftrightarrow K_i \leftarrow R_i, ac_i)\), where \(i = 1, 2\), such that \(t_1\) yields transformation morphisms \(G \xrightarrow{r_1} D_1 \xrightarrow{h_1} H_1\). Transformation \(t_1\) causes a conflict on \(t_2\) if there does not exist a morphism \(x : L_2 \rightarrow D_1\) such that \(g_1 \circ x = m_2\) and \(h_1 \circ x \models ac_2\). Rule \(r_1\) causes a conflict on \(r_2\) if there exists a transformation pair \((t_1, t_2) : (G \Rightarrow_{m_1, r_1} H_1, G \Rightarrow_{m_2, r_2} H_2)\) such that \(t_1\) causes a conflict on \(t_2\). If rule \(r_1\) does not cause a conflict on \(r_2\) and vice versa, rule pair \((r_1, r_2)\) is called parallel independent.

![Diagram of critical pairs](image)

A weak critical pair is a pair \((t_1, t_2) : (X \Rightarrow_{m_1, r_1} H_1, X \Rightarrow_{m_2, r_2} H_2)\) of transformations such that \(t_1\) causes a conflict on \(t_2\), \(m_1\) and \(m_2\) are jointly surjective, and \(m_i\) is not required to satisfy \(ac_i\) \((i = 1, 2)\).

Definition 9 (Dependency). Let a sequence \((t_1; t_2) : G \Rightarrow_{m_1, r_1} H_1 \Rightarrow_{m_2, r_2} H_2\) of transformations be given that apply rules \(r_i = (L_i \leftrightarrow K_i \leftarrow R_i, ac_i)\), where \(i = 1, 2\), such that \(t_1\) yields transformation morphisms \(G \xrightarrow{r_1} D_1 \xrightarrow{h_1} H_1\). Transformation \(t_2\) is dependent on \(t_1\) if there does not exist a morphism \(x : L_2 \rightarrow D_1\) such that \(h_1 \circ x = m_2\) and \(g_1 \circ x \models ac_2\). Rule \(r_2\) is dependent on rule \(r_1\) if there exists a transformation sequence \((t_1; t_2) : G \Rightarrow_{m_1, r_1} H_1 \Rightarrow_{m_2, r_2} H_2\) such that \(t_2\) is dependent on \(t_1\). If \(r_1\) is not dependent on \(r_2\) and \(r_2\) is not dependent on \(r_1\), rule pair \((r_1, r_2)\) is called sequentially independent.

![Diagram of dependencies](image)

A weak critical sequence is a sequence \((t_1; t_2) : G \Rightarrow_{m_1, r_1} X \Rightarrow_{m_2, r_2} H_2\) of transformations such that \(t_2\) depends on \(t_1\), \(n_1\) and \(m_2\) are jointly surjective (where \(n_1\) is the co-match of \(t_1\)), and \(m_i\) is not required to satisfy \(ac_i\) \((i = 1, 2)\).

Note that a weak critical pair for a pair of transformations \((t_1, t_2)\) applying rules \(r_i = (p_i, ac_i)\) for \(i = 1, 2\) is a critical pair [1] where the plain rules \(p_i\) are applied. (There is an analogous relation between weak critical sequences and critical sequences.)

As rule \(r_2\) in a rule pair \((r_1, r_2)\) will always be plain in this paper, a transformation step with \(r_1\) can cause a conflict on another one applying \(r_2\) if and only if it deletes an element that the second transformation step matches (see, for example, [1, Sect. 3.3] for a discussion). Thus, we can say that a transformation \(t_1 : G \Rightarrow_{r_1, m_1} H_1\) causes a conflict for transformation \(t_2 : G \Rightarrow_{r_2, m_2} H_2\) if and only if

\[m_1(L_1 \setminus K_1) \cap m_2(L_2) \neq \emptyset,\]

where \(L_1 \setminus K_1\) denotes the set-theoretic difference on graphs that is defined componentwise on node and edge sets. (In particular, \(L_1 \setminus K_1\) does not need to constitute a graph.)

Similarly, a transformation step can depend on another one if and only if the first step creates an element that the second one uses or the first one deletes an edge that is adjacent to a node the second one deletes. As in our application scenario, the second rule will always be non-deleting, this second case cannot happen. Thus, for our application scenario, the dependency of a transformation \(t_2 : H_1 \Rightarrow_{r_2, m_2} H_2\) on a transformation \(t_1 : G \Rightarrow_{r_1, m_1} H_1\) is equivalently expressed by saying that

\[n_1(R_1 \setminus K_1) \cap m_2(L_2) \neq \emptyset\]

holds, where \(n_1\) is the co-match of transformation \(t_1\).
Example 1. In this work, we will only be concerned with conflicts or dependencies in which the second rule is a trivial rule; it does not create or delete any elements but just checks for the existence of a certain graph. This will be graphs that stem from a constraint.

Figure 4 shows a weak critical sequence for an application of the rule `createClass`, followed by an application of a rule that just checks for the existence of a `Class`. This means that it checks for the existence of the graph `P_{c2}` that is the first graph of the constraint `c2` of our running example when formalizing it as a graph constraint in ANF (cf. Example 2 and Fig. 6). Names starting with ‘r’ denote elements stemming from the rule `createClass`, names starting with ‘c’ denote elements stemming from the constraint, the morphisms are indicated by the names, and ‘=’ denotes identifications of elements. The morphisms `n` and `p_1'` are jointly surjective, and no morphism `x : P_{c2} \rightarrow D` exists (as `D` does not contain a `Class`). Therefore, the sequence is a weak critical one. Alternatively, it can be noted that the intersection of `n(R_c \setminus K_c)` with `p_1'(P_{c2})` (in `H`) is not empty; it contains the `rc = cc:Class`. The application of `createClass` first creates the `Class` that the second rule matches.

Figure 5 shows a pair of transformations that is not critical; the notation is identical to the one used for Fig. 4. The first transformation shows an application of (the plain version of) `createClass` again. The morphisms `m` and `p_2'` are again jointly surjective, however, the pair is not critical because the transformations are not conflicting. A morphism `x : P_{c2}' \rightarrow D` exists such that `g \circ x = p_2'` holds. Alternatively, it can be noted that the intersection of `m(L_c \setminus K_c)` with `p_2'(P_{c2}')` (in `G`) is empty (because `L_c \setminus K_c` is empty). The application of `createClass` does not delete any element that the second rule matches.
4. Consistency-sustaining and consistency-improving rules and transformations

In this section, we introduce our new key concepts. We do so in three stages, first introducing foundational definitions for partial consistency, followed by a generic definition of consistency-sustainment and -improvement. Finally, we give stronger definitions for consistency sustaining and improving transformations which we will use to provide a static analysis in Sect. 5.

4.1. Partial consistency

To support the discussion and analysis of rules and transformations that improve graph consistency but do not produce a fully consistent graph in one step, we introduce the notion of partial consistency. We base this notion on relating the number of constraint violations to the total number of relevant occurrences of a constraint. Remember that, in this paper, a constraint is always linear and in alternating normal form with nesting level \( \geq 1 \). Requiring nesting level \( \geq 1 \) is no real restriction as constraints with nesting level 0 are Boolean combinations of true which means they are equivalent to true or false, anyhow. In contrast, restricting to linear constraints actually excludes some interesting cases. We believe that the extension of our definitions and results to also include the non-linear case will be doable. At the very least, the case in which every nesting level is either a conjunction or disjunction (and not a combination of both) seems tractable. Restricting to the linear case first, however, makes the statements much more accessible and succinct. For the satisfaction of an existential constraint, a single valid occurrence is enough. In contrast, universal constraints require the satisfaction of some sub-constraint for every occurrence in the graph. Hence, the resulting notion is binary in the existential case, but graduated in the universal one. To ensure that the occurrences can be enumerated and counted in finite time, we assume that all graphs are actually finite.

Definition 10 (Occurrences and violations). Let \( c = Q(\emptyset \rightarrow C, d) \) with \( Q \in \{ \exists, \forall \} \) be a constraint. An occurrence of \( c \) in a graph \( G \) is an injective morphism \( p : C \hookrightarrow G \), and \( \text{occ}(G, c) \) denotes the number of such occurrences.

If \( c \) is universal, its number of relevant occurrences in a graph \( G \), denoted as \( \text{ro}(G, c) \), is defined as \( \text{ro}(G, c) := \text{occ}(G, c) \) and its number of constraint violations, denoted as \( \text{ncv}(G, c) \), is the number of occurrences \( p \) for which \( p \not\models d \).

If \( c \) is existential, \( \text{ro}(G, c) := 1 \), and \( \text{ncv}(G, c) := 0 \) if there exists an occurrence \( p : C \hookrightarrow G \) such that \( p \models d \) but \( \text{ncv}(G, c) := 1 \) otherwise.

Definition 11 (Partial consistency). Given a graph \( G \) and a constraint \( c \), \( G \) is consistent w.r.t. \( c \) if \( G \models c \). The consistency index of \( G \) w.r.t. \( c \) is defined as

\[
\text{ci}(G, c) := 1 - \frac{\text{ncv}(G, c)}{\text{ro}(G, c)}
\]

where we set \( \frac{0}{0} := 0 \). We say that \( G \) is partially consistent w.r.t. \( c \) if \( \text{ci}(G, c) > 0 \).

The next proposition makes precise that the consistency index runs between 0 and 1 and indicates the degree of consistency a graph \( G \) has w.r.t. a constraint \( c \). It follows directly from the definitions.

Proposition 1 (Consistency index). Given a graph \( G \) and a constraint \( c \), then \( 0 \leq \text{ci}(G, c) \leq 1 \) and \( G \models c \) if and only if \( \text{ci}(G, c) = 1 \). Consistency implies partial consistency. Moreover, \( \text{ci}(G, c) \in \{0, 1\} \) for an existential constraint.

Example 2 (Consistency index). Based on Fig. 6, we can express the three informal constraints from Sect. 2 as graph constraints. They are all in ANF with nesting level \( \geq 1 \). Constraint \( c_1 \) can be expressed as \( \forall (P_{c_1}, \text{false}) \equiv \neg \exists P_{c_1} \), it has nesting level 1. Constraint \( c_2 \) becomes \( \forall (P_{c_2}, \exists P'_{c_2}) \), and constraint \( c_3 \) becomes \( \forall (P_{c_3}, \exists P'_{c_3}) \); both have nesting level 2. Graph \( G \) (in the left top corner of Fig. 6) satisfies \( c_1 \) and \( c_2 \). It does not satisfy \( c_3 \) since we cannot find an occurrence of \( P'_{c_2} \) for the occurrence of \( P_{c_2} \) in \( G \) where \( f_1 \) and \( f_2 \) are mapped to \( f_1 \) and \( f_3 \), respectively. A second occurrence of \( P_{c_3} \) in \( G \) maps Features \( f_1 \) and \( f_2 \) to \( f_3 \) and \( f_1 \), respectively. This mapping can be extended to an occurrence of \( P'_{c_3} \) in \( G \). Thus, graph \( G \) has the consistency index 0.5 with regard to \( c_3 \) since one violation exists and two non-violating occurrences are required.
4.2. Consistency-sustainment and -improvement

In the remainder of this section, our goal is to introduce the notions of consistency-sustaining and consistency-improving rule applications, which refine the established notions of preserving and guaranteeing rule applications [5].

Definition 12 (Consistency-sustainment and -improvement). Given a graph constraint $c$ and a rule $r$, a transformation $t : G \Rightarrow_{r,m} H$ is consistency-sustaining w.r.t. $c$ if $ncv(G, c) \geq ncv(H, c)$. It is consistency-improving if $ncv(G, c) > ncv(H, c)$.

A rule $r$ is consistency-sustaining if all of its applications are. It is consistency-improving if it is consistency-sustaining and there exists a graph $G \in Graph_{TG}$ with $ncv(G, c) > 0$ and a consistency-improving transformation $G \Rightarrow_{r,m} H$. A consistency-improving rule is strongly consistency-improving if all of its applications to graphs $G$ with $ncv(G, c) > 0$ are consistency-improving transformations.

In the above definition, we use the number of constraint violations and not the consistency index to avoid undesirable side-effects: The consistency index is susceptible to manipulation through the creation or deletion of valid occurrences. For example, defining improvement via a growing consistency index would lead to consistency-improving transformations (w.r.t. a universal constraint) that do not repair existing violations but only create new valid occurrences of the constraint. Hence, there would exist infinitely long transformation sequences where every step increases the consistency index but validity is never restored. The following proposition states which forms of repairing transformation sequences are finite ones. Consistency-improving transformations require that the number of constraint violations strictly decreases in each step. Therefore, using only such kind of transformations, we cannot construct infinite transformation sequences. Similarly, we cannot construct infinite transformation sequences where each step is an application of a strongly consistency-improving rule to an inconsistent graph as each step has to strictly decrease the number of constraint violations.

Proposition 2 (Repairing sequences). 1. There do not exist infinitely long sequences $(G_{i-1} \Rightarrow G_i)_{i \geq 1}$ of consistency-improving transformations.

\[\text{In the conference version of this paper [13], we actually defined consistency-sustainment using the consistency index. However, then the statement that consistency-sustainment implies consistency-preservation (see Theorem 1) is only true under additional preconditions. We are indebted to Lei Xu who constructed a plethora of example rules, which ultimately led to the discovery of that error.}\]
There do not exist infinitely long sequences \((G_{i-1} \Rightarrow r_i, m_i, G_i)_{i \geq 1}\) of applications of rules \(r_i\) that are strongly consistency-improving w.r.t. a constraint \(c\) such that \(nvc(G_i, c) > 0\) for all \(i \geq 0\).

Rules that are consistency-improving but not strongly so are allowed to be applicable to invalid graphs without actually reducing the number of violations (as long as this number is not increased). There just needs to exist an improving transformation via the rule. In contrast, strongly consistency-improving rules are always required to reduce the amount of inconsistency if there is any. On valid graphs, both variants just have to be consistency-sustaining, i.e., they are not allowed to introduce violations. Any consistency-improving rule can be turned into a strongly consistency-improving rule if suitable pre-conditions can be added that restrict the applicability of the rule to only those cases where it can actually repair a constraint violation.

This links the two forms of consistency-improving rules to their practical applications: in model repair \([8, 10]\) we want to use rules that only make changes to graphs when there are violations to be repaired—strongly consistency-improving rules. However, in evolutionary search \([11]\), we want to allow rules to be able to make changes even when there is no need for repair, but to fix violations when they occur; consistency-improving rules are well-suited here as they can be applied for optimization purposes.

4.3. Direct consistency-sustainment and -improvement

While constraint violation and partial consistency are easy to define and to understand, it turns out that they are inherently difficult to investigate. Comparing numbers of (relevant) occurrences and violations allows for very disparate behavior of consistency-sustaining (-improving) transformations: For example, a transformation is allowed to destroy as many valid occurrences as it repairs violations and is still considered to be consistency-sustaining w.r.t. a universal constraint. (In our running example, rule \(moveFeature\) moves a \texttt{Feature} from one \texttt{Class} to another and can thereby remove a violation but also introduce another one.)

Therefore, we introduce refined notions of consistency-sustainment and -improvement which we call direct. The idea behind these refinements is to retain the validity of occurrences of a universal constraint: valid occurrences that are preserved by a transformation are to remain valid. In this way, sustainment and improvement become more direct as it is no longer possible to compensate for introduced violations by introducing additional valid occurrences. The notions of (direct) sustainment and improvement are related to one another and also to the already known ones that preserve and guarantee constraints (Definition 6). In Sect. 5 we will show how these stricter definitions allow for static analysis techniques to identify (directly) consistency-sustaining and -improving rules.

The following definitions assume a transformation step \(t : G \Rightarrow H\) to be given; they relate occurrences of a constraint \(c\) in graphs \(G\) and \(H\) as depicted in Fig. 7. When a transformation step is not allowed to destroy an occurrence of \(c\), there has to be a morphism \(p : C \rightarrow D\) since graph \(D\) comprises the part of \(G\) that is preserved during the transformation.

The existence of a morphism \(p_D : C \rightarrow D\) such that the left triangle commutes (and \(p'\) is defined as \(h \circ p_D\)) is equivalent to the track morphism \(tr_t : G \rightarrow H\) being a total morphism when restricted to \(p(C)\). We formulate this equivalence in the following technical lemma, which we are going to use in the proofs of our results in the following sections.
Lemma 2 (Preservation of constraint occurrence). Let a transformation step \( t : G \Rightarrow H \) with transformation morphisms \( g : D \leftrightarrow G \) and \( h : D \leftrightarrow H \) and an occurrence \( p : C \leftrightarrow G \) of a constraint \( c \) in \( G \) be given (Fig. 7).

1. The track morphism \( t_r : G \rightarrow H \) of \( t \) is total when restricted to \( p(C) \) (i.e., \( t_r \circ p \) is a total morphism) if and only if there exists an injective morphism \( p_D : C \rightarrow D \) such that \( p = g \circ p_D \).

2. Analogously, given an injective morphism \( p' : C \rightarrow H \), \( p'(C) \) is contained in \( t_r(G) \) if and only if there exists an injective morphism \( p_D : C \rightarrow D \) such that \( p' = h \circ p_D \).

Now, we are ready to define direct consistency-sustainment: For existential constraints, a transformation just has to be \( c \)-preserving. For universal constraints, two requirements are checked: (1) Occurrences of the constraint that were already valid in \( G \) are still valid in \( H \), unless they have been removed; that is, the transformation must not make existing valid occurrences invalid. (2) Every “new” occurrence of the constraint in \( H \) satisfies the constraint; that is, the transformation must not introduce fresh violations.

Definition 13 (Direct consistency-sustainment). Given a graph constraint \( c \), a transformation \( t : G \Rightarrow_{m,r} H \) via rule \( r \) at match \( m \) with track morphism \( t_r \) is directly consistency-sustainning w.r.t. \( c \) if either \( c \) is existential and the transformation is \( c \)-preserving or \( c = \forall (C, d) \) is universal and

\[
\forall p : C \leftrightarrow G ((p \models d \land t_r \circ p \text{ is total}) \implies t_r \circ p \models d) \land \\
\forall p' : C \leftrightarrow H (\neg \exists \exists p : C \leftrightarrow G (p' = t_r \circ p) \implies p' \not\models d).
\]

A rule \( r \) is directly consistency-sustainning w.r.t. \( c \) if all its applications are.

Note that in the first case, we do not require the subcondition \( d \) to be satisfied at \( p' := t_r \circ p \) in \( H \) in the same way as it was at \( p \) in \( G \). It is enough if there is still a way to satisfy \( d \) at occurrence \( p' \).

The following theorem relates the new notions of (direct) consistency-sustainment to preservation and guarantee of constraints.

Theorem 1 (Sustainment relations). Given a graph constraint \( c \), every \( c \)-guaranteeing transformation is directly consistency-sustainning, every directly consistency-sustainning transformation is consistency-sustainning, and every consistency-sustainning transformation is \( c \)-preserving. The analogous implications hold on the rule level (as summarised in Fig. 8).

The following example illustrates these notions and shows that (direct) sustainment is different from constraint guarantee or preservation, i.e., that all implications of the theorem are proper.

Example 3 (Sustainment relations). Table 1 denotes for each rule of the running example whether it is consistency-sustainning w.r.t. each of the given constraints. Rule \textit{createClass} is directly consistency-sustainning w.r.t. \( c_1 \) (no double assignments) and \( c_2 \) (no empty classes), since it cannot assign an already assigned \textit{Feature} or remove existing assignments. However, it is not consistency-guaranteeing, since it cannot remove any violation either. Rule \textit{moveFeature} is consistency-sustainning w.r.t. \( c_1 \) but not directly so, since it can introduce new violations but only while removing another violation at the same time, leading to a neutral outcome.
Table 1: Properties of example rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Consistency sustaining</th>
<th>Consistency improving</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_1$ $c_2$ $c_3$</td>
<td>$c_1$ $c_2$ $c_3$</td>
</tr>
<tr>
<td>assignFeature</td>
<td>+ + -</td>
<td>- + -</td>
</tr>
<tr>
<td>createClass</td>
<td>+ + -</td>
<td>- - -</td>
</tr>
<tr>
<td>moveFeature</td>
<td>(+) -</td>
<td>- - -</td>
</tr>
<tr>
<td>deleteEmptyClass</td>
<td>+ + +</td>
<td>- +* -</td>
</tr>
</tbody>
</table>

Legend: + denotes directly, (+) denotes non-directly, * denotes strongly.

Furthermore, a constraint-preserving rule is not necessarily consistency-sustaining as the following example shows: Starting with the plain version of rule `createClass` and computing a preserving application condition for constraint $c_1$ according to the construction provided by Habel and Pennemann [5] results in the application condition depicted in Fig. 9. By construction, equipping the plain version of `createClass` with that application condition results in a consistency-preserving rule. The premise of the implication depicted checks whether rule `createClass` is applied only to graphs that fulfil $c_1$. Whenever applied to an invalid graph, this premise evaluates to false and, hence, the whole application condition to true. Therefore, `createClass` with this application condition can introduce further violations of $c_1$ and is not consistency-sustaining.

![Figure 9: Generated preserving application condition for `createClass` w.r.t. constraint $c_1$. The Feature named rf stems from the LHS of `createClass`.](image)

Similarly, the direct notion of consistency-improvement fixes the validity of an already existing occurrence or deletes an invalid occurrence in the case of universal constraints and degenerates to the known concept of constraint-guarantee in the existential case.

**Definition 14 (Direct consistency-improvement).** Given a graph constraint $c$, a transformation $t : G \Rightarrow^i H$ via rule $r$ at match $m : L \hookrightarrow G$ with track morphism $tr_t$ is directly consistency-improving w.r.t. $c$ if $G \not\models c$, the transformation is directly consistency-sustaining, and either $c$ is existential and the transformation is $c$-guaranteeing or $c = \forall (C, d)$ is universal and

$$
\exists p : C \hookrightarrow G(p \not\models d \land p' \models d) \lor \\
\exists p : C \hookrightarrow G(p \not\models d \land p' \models d) \lor \exists p : C \hookrightarrow G(p \not\models d \land p' \models d)
$$

We lift the notion of directly consistency-improving transformations to the level of rules in the same way as in Def. 12. This leads to directly consistency-improving rules and a strong form of directly consistency-improving rules.

Note that direct consistency-improvement is orthogonal to strong consistency-improvement. (Direct) consistency-improvement is related to, but different from, constraint guarantee and consistency-sustainment as made explicit in the next theorem.

**Theorem 2 (Improvement relations).** Given a graph constraint $c$, every directly consistency-improving transformation is a consistency-improving transformation and every consistency-improving transformation is consistency-sustaining w.r.t. $c$. Moreover, every $c$-guaranteeing transformation starting from a graph $G$
with \( G \not\models c \) is a directly consistency-improving transformation. The analogous implications hold on the rule level (as summarised in Fig. 10), provided that there exists a match for the respective rule \( r \) in a graph \( G \) with \( G \not\models c \).

As for consistency-sustainment, we illustrate the different notions of consistency-improvement showing that all implications in Theorem 2 are proper.

**Example 4 (Improvement relations).** Table 1 denotes for each rule of the running example whether it is consistency-improving w.r.t. each of the given constraints. The rule `deleteEmptyClass`, for example, is directly strongly consistency-improving but not -guaranteeing w.r.t. \( c_2 \) (no empty classes), since it always removes a violation (i.e., an empty `Class`) but generally not all violations in one step. Rule `assignFeature` is directly consistency-improving w.r.t. \( c_2 \) but not strongly so: It can turn empty `Classes` into non-empty ones but does not do so in every possible application. Rule `createClass` is consistency-sustaining but not -improving w.r.t. \( c_2 \), as it cannot reduce the number of empty classes.

Our running example does not include a rule which is consistency-improving but not directly so. An example can be given by a variant of the rule `moveFeature`, where a single `Feature` is removed from two `Classes` in which it is contained and moved to a third `Class`, in which it was not contained yet. Every application of this rule decrements the number of `Classes` in which the moved `Feature` is contained by one. With regard to constraint \( c_1 \) (no double assignments) this means that the number of occurrences of the forbidden pattern \( P_{c_1} \) (see Fig. 6), which coincides with the number of constraint violations in this case, is reduced (from \( \binom{k}{2} \) to \( \binom{k-1}{2} \), where \( k \) is the number of `Classes` the `Feature` is contained in). For the special case of a constraint of the form \( c = \neg \exists C \equiv \forall (C, \text{false}) \), we have \( ncv(G, c) = ro(G, c) \), since no occurrence can satisfy `false`. When applying that rule as long as possible, this means that the consistency index \( ci(G, c) \) remains 0 constantly until the very last violation is removed. At that point the consistency index switches to 1. Summarizing, this rule is consistency-improving and even strongly so. However, the rule is not directly consistency-sustaining and, hence, not directly consistency-improving: Whenever a `Feature` is contained in more than two `Classes`, applying that rule reduces the overall number of violations at the cost of introducing new ones. This behaviour is excluded by the definition of direct consistency-sustainment.

**5. Static Analysis for Direct Consistency-Sustainment and -Improvement**

In this section, we present a static analysis technique for direct consistency-sustainment and -improvement. This technique comprises two criteria for deciding whether a rule is directly consistency-sustaining or directly consistency-improving w.r.t. given constraints. The criterion for direct consistency-sustainment is a sufficient but not necessary one; hence, it can recognize directly consistency-sustaining rules but not all of them. So if the criterion does not hold for a rule, the rule has to be checked manually. Conversely, the criterion for direct consistency-improvement is a necessary but not sufficient one. Hence, it can be used to decide that a given rule is not directly consistency-improving. When the criterion does not hold for a rule, the rule has to be checked manually.

The general idea behind our static analysis technique is to check for the validity of a constraint by applying a trivial (non-modifying) rule that just checks for the existence of a graph occurring in the constraint. This idea is in line with the representation of constraints in GROOVE as presented in [22]. For a graph \( C \) occurring in a given constraint, we consider the interaction of a rule \( r = (L \leftrightarrow K \leftrightarrow R, ac) \) with the (plain) rule \( \text{check}_C := (C \leftrightarrow C \leftrightarrow C) \). This allows us to present our analysis technique in the terminology
of conflicts and dependencies, which has been developed to characterise the possible interactions between rule applications [19, 1]. As a bonus, we obtain tool support for an automated analysis based on Henshin since the efficient detection of such conflicts and dependencies has been the focus of recent theoretical and practical research [21, 23]. The intuition behind the following results is that sequential independence of the (non-modifying) rule check\(_C\) from \(r\) means that \(r\) cannot create a new occurrence of \(C\). Similarly, parallel independence of check\(_C\) from \(r\) means that \(r\) cannot destroy an occurrence of \(C\).

5.1. Consistency-sustaining interaction

We start by stating a criterion for direct consistency-sustainment: Intuitively it says the following: A rule is directly consistency-sustaining if it cannot destroy an occurrence of a constraint graph that is bound by an existential quantifier and cannot cause a new occurrence of a constraint graph that is bound by a universal quantifier.

We start with defining this kind of interaction and give an example thereafter. To be able to reason about interactions by structural induction in our proofs, we define them in terms of conditions instead of constraints.

**Definition 15 (Consistency-sustaining interaction).** Let a condition \(c = Q(a_1 : C_0 \hookrightarrow C_1, Q(a_2 : C_1 \hookrightarrow C_2, \ldots))\) with \(n(c) \geq 1\) and a rule \(r = (L \leftarrow K \rightarrow R, ac)\) be given. We say that \(r\) interacts with \(c\) in a consistency-sustaining manner if the two following properties are met.

1. **Unproblematic deletions in conflicts:** For all graphs \(C_i\) of \(c\) with \(i \geq 1\) that are bound by an existential quantifier and for all weak critical pairs \((t_1, t_2): (G \Rightarrow_{r,m} H, G \Rightarrow_{\text{check}_{C_1}, p_i} G)\), transformation \(t_1\) causes a conflict on \(t_2': G \Rightarrow_{\text{check}_{C_{i-1}}, p_i, o_a}, G\) (with \(a_i : C_{i-1} \hookrightarrow C_i\)).

2. **Unproblematic additions in dependencies:** For all graphs \(C_i\) of \(c\) with \(i \geq 1\) that are bound by a universal quantifier and for all weak critical sequences \((t_1; t_2): (G \Rightarrow_{r,m} H \Rightarrow_{\text{check}_{C_1}, p'_i} H)\), transformation \(t_2': H \Rightarrow_{\text{check}_{C_{i-1}}, p'_i, o_a}, H\) is dependent on transformation \(t_1\) (with \(a_i : C_{i-1} \hookrightarrow C_i\)).

We call deletions and creations problematic, if they are not unproblematic.

**Remark 2.** Rule \(r\) causes a conflict on check\(_C\) when we have \(m(L \setminus K) \cap p_i(C_i) \neq \emptyset\). Criterion 1 stated above checks if this conflict is concerned with the subgraph \(a_i(C_{i-1})\) of \(C_i\) already. That is, every such conflict satisfies

\[
m(L \setminus K) \cap p_i(a_i(C_{i-1})) \neq \emptyset.
\]

In that case, the deletions by rule \(r\) are unproblematic as the occurrence of \(C_{i-1}\) is also removed. Intuitively, together with the required existential occurrence, such conflicts remove the reason for which the occurrence had been required. Similarly, a dependency of check\(_C\) on \(r\) is unproblematic when we have

\[
n(R \setminus K) \cap p_i(a_i(C_{i-1})) \neq \emptyset,
\]

where \(n : R \hookrightarrow H\) is the co-match of the first transformation. That is, Criterion 2 above checks if this dependency is concerned with the subgraph \(a_i(C_{i-1})\) of \(C_i\) already. In that case, the newly introduced occurrence of \(C_i\) is not relevant for the degree to which \(c\) is valid, because the preceding occurrence of \(C_{i-1}\) is also newly added by the same transformation.

**Theorem 3 (Criterion for direct consistency-sustainment).** Let a constraint \(c\) and a rule \(r\) be given. If \(r\) interacts with \(c\) in a consistency-sustaining manner, then \(r\) is directly consistency-sustaining w.r.t. \(c\).

**Example 5.** We stated in Example 3 that rule createClass is directly consistency-sustaining with respect to constraint \(c_1\) (no double assignment of Features). The above theorem can be used to recognise this. There is no dependency of check\(_{P_{c_1}}\) on createClass, as the only two possibilities for such a dependency (createClass creating one of the Classes of \(P_{c_1}\)) are excluded by its NAC. Hence, createClass interacts consistency-sustainingly with \(c_1\). As rule createClass does not delete anything, Criterion 1 of Definition 15 is automatically fulfilled.
To illustrate Criterion 1 of Def. 15, consider the rule \textit{deleteClass} (not be confused with \textit{deleteEmptyClass} of our running example) that is basically inverse to \textit{createClass}, namely a rule that deletes a \texttt{Class} as well as a \texttt{contains}-edge that connects the deleted \texttt{Class} to a preserved \texttt{Feature}. This rule causes a conflict for the existentially bound graph \(P_{e2}\) of constraint \(c_2\) (no empty classes). Namely, it might delete the \texttt{Class} \(c_1\) together with its outgoing \texttt{contains}-edge. However, Criterion 1 states that this conflict does not need to be considered because (at least) one of the elements inducing the conflict, here \texttt{Class} \(c_1\), stems from the preceding universally bound graph \(P_{e2}\) already. Thus, \textit{deleteClass} interacts consistency-sustainingly with \(c_2\). This example also illustrates the intuitive reason why conflicts like this one do not need to be considered: Here, the existence of a \texttt{Feature} is only required to complement a \texttt{Class}. Thus, the deletion of the \texttt{contains}-edge is unproblematic when the \texttt{Class} is removed simultaneously. As rule \textit{deleteClass} does not create anything, Criterion 2 of Definition 15 is automatically fulfilled.

We can use the criteria given in Definition 15 to automatically equip a given rule with an application condition such that the resulting rule is consistency-sustaining w.r.t. the given constraint. The idea is to construct a set of NACs that forbid all applications of the rule that (i) would introduce new occurrences of a universally bound graph of the constraint and (ii) would destroy occurrences of an existentially bound graph of the constraint.

**Construction 1 (Consistency-sustaining application condition).** Let a plain rule \(r = (L \leftrightarrow K \leftrightarrow R)\) and a constraint \(c\) with \(\text{nl}(c) \geq 1\) be given. The application condition \(ac_{sus}\) of \(r\) is defined as

\[
ac_{sus} := ac_\forall \land ac_\exists
\]

where \(ac_\forall\) and \(ac_\exists\) are constructed as follows:

For every universally bound graph \(C_i\) of \(c\), let \(CR_i\) be the set of all graphs \(P_j\) that arise as overlaps of \(C_i\) and \(R\) such that there exists a pair of injective and jointly surjective morphisms \(e_{1,j} : C_i \hookrightarrow P_j\) and \(e_{2,j} : R \hookrightarrow P_j\) with

\[
e_{2,j}(R \setminus K) \cap e_{1,j}(C_i) \neq \emptyset.
\]

Then

\[
ac_\forall := \bigwedge_{C_i \in c_\forall} \bigwedge_{P_j \in CR_i} \left( \neg \exists (e_{2,j} : R \hookrightarrow P_j, r), \right),
\]

where \(c_\forall\) denotes the set of the universally bound graphs of \(c\).

For every existentially bound graph \(C_i\) of \(c\), let \(CL_i\) be the set of all graphs \(P_j\) that arise as overlaps of \(C_i\) and \(L\) such that there exists a pair of injective and jointly surjective morphisms \(e_{1,j} : C_i \hookrightarrow P_j\) and \(e_{2,j} : L \hookrightarrow P_j\) with

\[
\emptyset \subset e_{2,j}(L \setminus K) \cap e_{1,j}(C_i) \subseteq e_{1,j}(C_i \setminus a_i(C_{i-1})).
\]

Then

\[
ac_\exists := \bigwedge_{C_i \in c_\exists} \bigwedge_{P_j \in CL_i} \neg \exists (e_{2,j} : L \hookrightarrow P_j),
\]

where \(c_\exists\) denotes the set of the existentially bound graphs of \(c\).

We prove the correctness of this construction by showing that a rule enhanced in such a way interacts with the given constraint in a consistency-sustaining manner, i.e., by applying Theorem 3.

**Proposition 3 (Correctness of consistency-sustaining application conditions).** Given a constraint \(c\) and a plain rule \(p\), the rule \(r := (p, \text{ac}_{sus})\), i.e., \(p\) equipped with the application condition as constructed above, is consistency-sustaining w.r.t. \(c\).

**Example 6 (Construction of NACs).** Given the rule \textit{moveFeature} and the constraint \(c_3\), the construction given above results in the application condition that is depicted in Fig. 11. The first two lines constitute \(ac_\forall\) and the last line constitutes \(ac_\exists\). This means that the subconditions on the first two lines prevent the
creation of a new occurrence of graph $P_{c_3}$ (compare Fig. 6). The NACs in these lines are constructed from all possible overlaps of the contains-edge running between rf and rc2 with contains-edges of $P_{c_3}$. As the contains-edge is the only element that is deleted by the rule moveFeature, we do not have to consider further overlaps of $P_{c_3}$ with the right-hand side. Shifting these overlaps to the left-hand side with the construction Left deletes the overlapping contains-edge in each graph and inserts a contains-edge to another Class.

The two subconditions on the last line prevent the destruction of occurrences of graph $P'_{c_3}$. Note that it is enough to overlap Feature rf with cf3 since cf3 is in $P'_{c_3} \setminus P_{c_3}$ but cf1 and cf2 are not.

The above example shows a limitation of our construction of NACs: The resulting set of NACs is very strict, clearly stricter than necessary to make a rule directly consistency-sustaining. In our example, the computed set of NACs almost prevents the movement of a Feature that has an incoming or outgoing dependency (the exception is the case where the second Feature is not contained in a Class yet). The advantage of our construction is that it computes NACs and not more general application conditions. Those are comparatively easy to check and, moreover, conflicts and dependencies involving rules with NACs have been characterised and their computation has been implemented in Henshin [20, 4]. This is not the case for more general application conditions yet. Determining whether our construction is useful in practice or potentially too restrictive is left to future work.

5.2. Constraint-compatible dependencies and conflicts

The criteria presented in Theorem 3 are sufficient but not necessary. Next, we present further criteria for direct consistency-sustainment; they are also sufficient but not necessary but strengthen the previous ones. Intuitively they state the following: A rule is directly consistency-sustaining if it deletes only occurrences of existentially quantified graphs that are redundant. Moreover, it creates new valid occurrences only when they are of universally quantified graphs of the given constraint. For constraints of the form $\forall (C_1, \exists C_2)$, for example, $r$ may create a new valid occurrence of $C_1$ or delete an occurrence of $C_2$ but leaves another one intact. These cases are not taken into account by the criteria of Def. 15. The next proposition strengthens the theorem above by partially remedying it. As above, we first introduce the new criteria.

**Definition 16 (Constraint-compatible interaction).** Let a condition $c = Q(a_1 : C_0 \hookrightarrow C_1, Q(a_2 : C_1 \hookrightarrow C_2, \ldots))$ with $n(c) \geq 1$ and a rule $r = (L \hookrightarrow K \hookrightarrow R, ac)$ be given. We say that $r$ and $c$ interact constraint-compatibly if the two following properties are met.

1. $r$ conflicts constraint-compatibly with $c$: For all graphs $C_i$ in $c$ that are bound by an existential quantifier, for all weak critical pairs $(t_1, t_2) : (G \Rightarrow_{r,m} H, G \Rightarrow_{\text{check}_{C_i,p_1}} G)$ with problematic deletions (i.e., Criterion 1 in Def. 15 is violated), there exists an injective morphism $p'_i : C_i \hookrightarrow H$ such that $p'_i \circ a_i = p'_{i-1}$ (where morphism $p'_{i-1} : C_{i-1} \hookrightarrow H$ is guaranteed to exist since transformation $t_1$ does not cause a conflict on transformation $t'_2$).
2. \( c \) depends constraint-compatibly on \( r \): For all graphs \( C_i \) in \( c \) that are bound by a universal quantifier, if there exists a weak critical sequence \((t_1; t_2) : (G \Rightarrow r, m \Rightarrow H \Rightarrow p_{c_2} H)\) with problematic creation (i.e., Criterion 2 in Def. 15 is violated), then \( C_i \) is not the last graph occurring in \( c \) and there exists an injective morphism \( p'_{i+1} : C_{i+1} \hookrightarrow H \) such that \( p'_{i+1} \circ a_{i+1} = p'_i \) (where \( a_{i+1} : C_i \hookrightarrow C_{i+1} \) is the next morphism of \( c \)).

Proposition 4. Let a constraint \( c \) with \( nl(c) \geq 1 \) and a rule \( r = (L \leftarrow K \leftarrow R, ac) \) be given. If \( r \) and \( c \) interact constraint-compatibly, then \( r \) is directly consistency-sustaining w.r.t. \( c \).

Example 7 (Constraint-compatible interaction). The rule \( \text{check}_{p_{c_2}} \) that checks for the existence of the universally bound first graph of the constraint \( c_2 \) (no empty class) has a dependency on the rule \( \text{createClass} \) (see Fig. 4 in Example 1). However, in the weak critical sequence \((G \Rightarrow \text{createClass} H \Rightarrow \text{check}_{p_{c_2}} H)\) that captures this dependency, the graph \( G \) consists of a single \( \text{Feature} \). Consequently, \( H \) has a newly created \( \text{Class} \) containing this \( \text{Feature} \) (both rules have only one possible match each). There is also an occurrence \( p'_2 \) of the (existentially bound) graph \( P_{c_2} \) in \( H \) which suitably extends the newly created occurrence of \( P_{c_2} \), namely such that \( p'_2 \circ a_2 = p'_1 \) (depicted in Fig. 12). Moreover, as \( \text{createClass} \) does not delete elements, it cannot cause a conflict on \( \text{check}_{p_{c_2}} \) (compare Fig. 5 in Example 1). Thus, \( c_2 \) depends constraint-compatibly on \( \text{createClass} \). According to Proposition 4, \( \text{createClass} \) is directly consistency-sustaining with respect to constraint \( c_2 \).

5.3. Consistency-improving interaction

For consistency-improvement we state criteria on rules as well: If a rule is directly consistency-improving w.r.t. a constraint, it is able to either (1) destroy an occurrence of a universally quantified graph (by deleting at least a part of it) or (2) create a new occurrence of an existentially quantified one (by creating at least a part of it).

Definition 17 (Consistency-improving interaction). Let a condition \( c = \{Q(a_1 : C_0 \hookrightarrow C_1, Q(a_2 : C_1 \hookrightarrow C_2, \ldots)\ldots\} \) with \( nl(c) \geq 1 \) and a rule \( r = (L \leftarrow K \leftarrow R, ac) \) be given. We say that \( r \) has a consistency-improving interaction with \( c \) if at least one of the following properties is met.
1. **Deletion of a universal occurrence:** For some universally quantified graph $C_i$ of $c$, there exists a weak critical pair $(t_1, t_2) : (G \Rightarrow_{r,m} H, G \Rightarrow_{\text{check}_{C_i},p_i} G)$ such that the transformation $t_1$ does not cause a conflict on the transformation $t_2 : G \Rightarrow_{\text{check}_{C_{i-1}},p_{i-1}} G$ (with $a_i : C_{i-1} \hookrightarrow C_i$).

2. **Creation of existential occurrence:** For some existentially quantified graph $C_i$ of $c$, there exists a weak critical sequence of transformations $(t_1; t_2) : (G \Rightarrow_{r,m} H \Rightarrow_{\text{check}_{C_i},p_i'} H)$ such that the transformation $t_2 : H \Rightarrow_{\text{check}_{C_{i-1}},p_{i-1}} H$ is not dependent on $t_1$ (with $a_i : C_{i-1} \hookrightarrow C_i$).

**Remark 3.** As in the case of consistency-sustaining interaction (Def. 15), such conflicts resp. dependencies can be characterised with simple set-theoretical equations. Here, Criterion 1 ensures that the conflict is not already concerned with the subgraph $a_i(C_{i-1})$ of $C_i$. That is, there exists a conflict that satisfies

$$m(L \setminus K) \cap p_i(a_i(C_{i-1})) = \emptyset.$$  

Similarly, Criterion 2 requires that the newly created occurrence of $C_i$ extends an existing occurrence of $C_{i-1}$. That is, there exists a dependency that satisfies

$$n(R \setminus K) \cap p'_i(a_i(C_{i-1})) = \emptyset,$$

where $n$ is the co-match of the first transformation.

The two criteria just presented can be used as follows: If none of the two criteria is fulfilled, rule $r$ is not consistency-improving w.r.t. constraint $c$. Hence, they are necessary conditions. This relation is stated in the following theorem.

**Theorem 4 (Criterion for direct consistency-improvement).** Let a constraint $c$ with $\text{nl}(c) \geq 1$ and a rule $r$ be given. If $r$ is directly consistency-improving w.r.t. $c$, then $r$ interacts consistency-improvingly with $c$.

Since we want to have a criterion that helps us to decide whether a rule is directly consistency-improving w.r.t. a given constraint, we will use this theorem in its negated form: **If** $r$ **does not interact consistency-improvingly** with $c$, **$r$ is not directly consistency-improving** w.r.t. $c$. This form shows that the property of not interacting consistency-improvingly is a sufficient one to decide that a rule is not directly consistency-improving.

**Example 8 (Consistency-improving interaction).** Considering the rule `assignFeature` and constraint $c_1$ (no double assignment of a `Feature`), we see that `assignFeature` cannot cause a conflict on `check_{P_1}`, as `assignFeature` does not delete anything. Hence, `assignFeature` is not directly consistency-improving w.r.t. $c_1$.

Considering the rule `moveFeature` and constraint $c_2$ (no empty class), we can state that `moveFeature` fails to be consistency-improving (directly or not), since it is not consistency-sustaining w.r.t. $c_2$.

Considering the rule `createClass` and constraint $c_2$, we know that `createClass` is consistency-sustaining w.r.t. $c_2$ according to Example 7. To check whether `createClass` is not additionally consistency-improving w.r.t. $c_2$, we can state that `createClass` does not cause a conflict on `check_{P_2}` as it does not delete anything. Additionally, we have to check whether there may be dependencies of `check_{P_2}` on `createClass`. This may happen as `check_{P_2}` has a `contains`-edge that may be created by `createClass`. Hence, the criterion is not fulfilled and we have to check the rule by hand.

**5.4. Summary**

Table 2 offers a simplified summary of the *sufficient criteria* we obtain in Theorems 3 and 4 for constraints up to nesting level 3. We ignore the fact that not all but rather specific kinds of conflicts and dependencies are forbidden by our results.
Table 2: Simplified summary of the sufficient criteria from Theorems 3 and 4 for constraints up to nesting level 3. Here, \( ck_C \) is short for \( check_C \), \( r_1 <_D r_2 \) denotes dependency of \( r_2 \) on \( r_1 \), \( r_1 <_C r_2 \) denotes \( r_1 \) causing a conflict for \( r_2 \), and crossed out versions denote the respective absence.

<table>
<thead>
<tr>
<th>type of constr.</th>
<th>crit. for directly consist. sust.</th>
<th>crit. for not directly consist. impr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall (C, \text{false}) \equiv \neg \exists C )</td>
<td>( r \not&lt;_D ck_C )</td>
<td>( r \not&lt;_C ck_C )</td>
</tr>
<tr>
<td>( \exists C )</td>
<td>( r \not&lt;_C ck_C )</td>
<td>( r \not&lt;_D ck_C )</td>
</tr>
<tr>
<td>( \forall (C_1, \exists C_2) )</td>
<td>( r \not&lt;<em>D ck</em>{C_1} \land r \not&lt;<em>C ck</em>{C_2} )</td>
<td>( r \not&lt;<em>C ck</em>{C_1} \land r \not&lt;<em>D ck</em>{C_2} )</td>
</tr>
<tr>
<td>( \exists (C_1, \neg \exists C_2) )</td>
<td>( r \not&lt;<em>C ck</em>{C_1} \land r \not&lt;<em>D ck</em>{C_2} )</td>
<td>( r \not&lt;<em>D ck</em>{C_1} \land r \not&lt;<em>C ck</em>{C_2} )</td>
</tr>
<tr>
<td>( \forall (C_1, \exists (C_2, \neg \exists C_3)) )</td>
<td>( r \not&lt;<em>D ck</em>{C_1} \land r \not&lt;<em>C ck</em>{C_2} \land r \not&lt;<em>D ck</em>{C_3} )</td>
<td>( r \not&lt;<em>C ck</em>{C_1} \land r \not&lt;<em>D ck</em>{C_2} \land r \not&lt;<em>C ck</em>{C_3} )</td>
</tr>
<tr>
<td>( \exists (C_1, \forall (C_2, \exists C_3)) )</td>
<td>( r \not&lt;<em>C ck</em>{C_1} \land r \not&lt;<em>D ck</em>{C_2} \land r \not&lt;<em>C ck</em>{C_3} )</td>
<td>( r \not&lt;<em>D ck</em>{C_1} \land r \not&lt;<em>C ck</em>{C_2} \land r \not&lt;<em>D ck</em>{C_3} )</td>
</tr>
</tbody>
</table>

6. Validation

In Sect. 5, we introduce a set of criteria that, if implemented in a tool, allow automated analysis of direct consistency-sustainment and direct consistency-improvement in rules. Since the criteria for direct consistency-sustainment are sufficient ones, the resulting analysis is under-approximating: it can only classify some, but not all rules as sustaining with certainty. Conversely, the criteria of direct consistency-improving rules are necessary ones, rendering the analysis over-approximating: they can only tell us with certainty that a given rule is not improving. In our validation, we study the quality of both approximations on a real rule set.

Specifically, we address the following research questions:

RQ1: What is the extent of over-approximation of the analysis regarding direct consistency-sustainment?
RQ2: What is the extent of under-approximation of the analysis regarding direct consistency-improvement?

6.1. Methodology

The considered rule set was generated by an automated tool, based on the rule generation approach by Burdusel et al. [11]. The goal of that approach is to generate, from a given meta-model, search operator rules that are consistency-sustaining with regard to the meta-model’s multiplicity constraints. Hence, it is a particularly relevant application scenario for our analysis, since our analysis essentially tests whether the approach achieves its goal. The rules for our evaluation were generated from the meta-model for the CRA case [15] (as introduced in Sect. 2) in the rule format of Henshin [3, 4]. The overall rule set consists of seven rules related to the creation and deletion of classes and the creation and deletion of containment references. Two of the generated rules had positive application conditions (PACs) which are of the form \( \exists (L \mapsto P) \).

Since the definition of constraint compatible interaction (Def. 16 being the basis for Proposition 4) does not yet take elements into account whose existence is required by a PAC, we included the elements of these PACs in the rules’ left-hand sides leading to equivalent rules.

Since the rule set was generated in a way that was tailored towards multiplicity constraints, we considered direct consistency-sustainment and direct consistency-improvement with regard to three multiplicity constraints. The first two are the same as \( c_1 \) and \( c_2 \) in the running example (see Fig. 6). As a third constraint \( c_3' \), we considered another real constraint from the CRA case: each feature needs to be assigned to at least one class. This constraint can be expressed as a nested graph constraint in a similar way as \( c_2 \). In particular, like every multiplicity constraint these constraints are naturally expressed as constraints in ANF, i.e., our definitions and methods are applicable.

Our criteria from Sect. 5 rely on conflicts and dependencies between certain rule pairs, in which the first rule is the rule to be analysed, and the second rule is a check rule for some component of the constraint at hand. We performed this analysis automatically, using the conflict and dependency analysis capabilities provided by the Henshin tool [4]. Our criteria further specify certain properties of the identified conflicts and dependencies, the analysis of which could, in principle, be automated. Since a full automation of the
analysis is out of scope of the present work, we inspected the obtained conflicts and dependencies manually. To reduce the possibility of manual errors, two authors double-checked the results.

In total, we check our criteria for direct consistency-sustainment and direct consistency-improvement for 21 cases, arising from 7 rules and 3 constraints. To address the research questions, we consider the “recognition ratio”, defined as the number of cases covered by the criterion at hand, divided by the number of all cases. In RQ1, this number quantifies the ratio of cases where we can classify a rule as sustaining. In RQ2, this number quantifies the ratio of cases where we can classify a rule as non-improving. In addition, we consider the “coverage ratio”, defined as the number of cases covered by the criterion at hand, divided by the number of directly consistency-sustaining rules for RQ1 and the number of rules that are not directly consistency-improving for RQ2.

In all cases, we obtain a real number between 0 and 1, where a number close to 1 indicates a low extent of under- or over-approximation, respectively. We discuss the practical implications of these numbers by comparing with the alternative of a fully manual assessment.

The artifacts for our validation (implementation, subject rules with representation as images, spreadsheet with results analysis) are publicly available at https://github.com/dstrueber/conssus/.

6.2. Results

Table 3 shows the results of applying the criteria to the subject rules. In the table, a “+” indicates that our analysis supports the correct classification of the case as directly consistency-sustaining, a “-” indicates a correct classification as non-improving, and a “?” indicates that our criteria do not support any statement about the case. Each “?” is annotated with a “+” when a rule has the property in question (i.e. is directly consistency-sustaining or -improving, resp.). Otherwise, a “?” is annotated with a “-”.

In total, we find that our criteria lead to 16 of 21 cases being correctly classified as directly consistency-sustaining, leading to a recognition ratio of 0.76. As all rules are classified that are consistency-sustaining, the coverage ratio is 1.0. Furthermore, 7 of 21 cases were correctly classified as non-improving, leading to a recognition ratio of 0.33. As there are further 7 cases where the rule is non-improving that are not covered by our criteria, the coverage ratio is 0.5.

We consider in more detail how these cases arise. For direct consistency-sustainment, 7 of the identified 16 cases follow from the absence of relevant conflicts or dependencies and do not require further inspection. The other 9 cases follow from inspection of the obtained conflicts and dependencies: The occurring conflicts either satisfy Criterion 1 in Def. 15—and therefore do not have to be considered—or direct sustainment is implied by Proposition 4 in these cases. Further considering the 5 cases where we cannot make a statement, we find that 3 of them are in fact consistency-sustaining, but not directly so, and hence, not covered by our criterion (c1 combined with r3, r6, and r7). The other 2 cases are indeed not consistency-sustaining at all and hence, not covered by our criterion (c2 combined with r4 and r5). Thus, even though our criteria are not necessary for direct consistency-sustainment, every such case has been detected by them, which is very promising.

For direct consistency-improvement, 5 of the identified cases follow from the existence of relevant conflicts or dependencies, and 2 from further inspection. Considering the remaining 14 cases, 7 of them are indeed directly consistency-improving (c1 and c2 each combined with r3, r6, and r7; c2 combined with r4), but cannot be distinguished from the other 7 non-improving ones using our criterion. (Remember that, for being consistency-improving, a rule has to be consistency-sustaining but not directly though.)

In conclusion, addressing RQ1, we observe good results concerning our analysis of directly consistency-sustaining rules: The criteria recognized directly consistency-sustaining rules in 76% of all cases which are all in this setup leading to a coverage ratio of 1. Hence, it looks like the criteria for consistency-sustainment work well. Addressing RQ2, the results concerning the analysis of directly consistency-improving rules is not as satisfying, but manual recognition can be avoided in 33% of all cases which is still half of the cases that could be recognized as not directly consistency-improving ones. We conjecture that the analysis for direct consistency-improvement can catch up in the future with improved results if more advanced criteria can be found.
Table 3: Validation results for RQ1 and RQ2.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Dir. cons. sustaining</th>
<th>Dir. cons. improving</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$: addToClass_encapsulates_Feature</td>
<td>$c_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$r_2$: createClass_IN_ClassModel</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$r_3$: createClass_IN_ClassModel_LB</td>
<td>$?_-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$r_4$: deleteClass_IN_ClassModel</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$r_5$: removeFromClass_encapsulates_Feature</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$r_6$: changeFeature_isEncapsulatedBy_TO_Class_S</td>
<td>$?_-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$r_7$: deleteClass_IN_ClassModel_S</td>
<td>$?_-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

6.3. Threats to Validity

The external validity of our validation is threatened as we consider only one rule set (in addition to the one of our running example). Still, this rule set uses the expressivity of transformation rules and includes practically relevant constraints of the type one would also find in other examples. While a standard benchmark of rules together with graph constraints is not readily available, developing one is a desirable direction for future work.

The internal validity of our validation is threatened by possible mistakes in the analysis, either implementation bugs or mistakes in the manual analysis. To reduce the likelihood of implementation errors, we used both internal engines for conflict and dependency analysis offered by Henshin, which led to agreeing results. Since two of the authors double-checked the manual results, we still have a reasonable level of confidence in their correctness.

7. Related Work

In this paper, we introduce a graduated version of a specific logic on graphs, namely of nested graph constraints in alternating normal form. Moreover, we focus on the interaction of this graduation with graph transformations. Therefore, we leave a comparison with fuzzy or multi-valued logics (on graphs) to future work. Instead, we focus on works that also investigate the interaction between the validity of nested graph constraints and the application of transformation rules.

Given a graph transformation (sequence) $G \Rightarrow H$, the validity of graph $H$ can be established with basically three strategies: (1) graph $G$ is already valid and this validity is preserved, (2) graph $G$ is not valid and there is a $c$-guaranteeing rule applied which makes the graph valid in one step, and (3) graph $G$ is made valid by a graph transformation (sequence) step-by-step.

Strategies (1) and (2) are supported by the incorporation of constraints in application conditions of rules as presented in [5] for nested graph constraints in general and implemented in Henshin [24]. As the applicability of rules enhanced in that way can be severely restricted, improved constructions have been considered for specific forms of constraints. For constraints of the form $\forall(C, \exists C')$, for example, a suitable rule scheme is constructed in [25]. In [26], refactoring rules are checked for the preservation of constraints of nesting level $\leq 2$. In [24], two of the present authors suggested certain simplifications of application conditions; the resulting ones are still constraint-preserving. In [27], we even showed that they result in the logically weakest application condition that is still directly consistency-sustaining. However, the result is only shown for negative constraints of nesting level 1. A very similar construction of negative application conditions from such negative constraints has been suggested in [28] very recently.

Strategy (3) is followed in most of the rule-based repair approaches for graphs or models. In [9], the violation of mainly multiplicity constraints is considered. In [7], Habel and Sandmann derived graph programs from graph constraints of nesting level $\leq 2$. In [10], they extended their results to constraints in ANF
which end with $\exists C$ or constraints of one of the forms $\exists(C, \neg\exists C')$ or $\neg\exists C$. They also investigated whether a given set of rules allows to repair such a given constraint. In [29], Dyck and Giese presented an approach to automatically check whether a transformation sequence yields a graph that is valid with relation to specific constraints of nesting level $\leq 2$. In [30, 31], the authors consider model repair approaches where already performed model changes are preserved as far as possible. Models are completed such that they become valid. All these model repair approaches are rule-based. In contrast, Schneider et al. [32] derive graph repairs from consistency constraints by using constraint solving techniques.

All these related approaches have in common that result graphs of transformations are considered either valid or invalid with relation to a graph constraint; intermediate consistency grades have not been made explicit. Thereby, $c$-preserving and $c$-guaranteeing transformations [5] focus on the full validity of the result graphs. Our newly developed notions of consistency-sustainment and improvement are located properly in between existing kinds of transformations (as proven in Theorems 1 and 2). These new forms of transformations make the gradual improvements in consistency explicit. While a detailed and systematic investigation (applying the static methods developed in this paper) is future work, a first check of the kinds of rules generated and used in model editing [33], model repair [9], and search-based model engineering [11] reveals that—in each case—at least some of them are indeed (directly) consistency-sustaining or consistency-improving. Therefore, we are confident that the current paper formalises properties of rules that are practically relevant in diverse application contexts. Work on partial graphs as in [34], for example, investigates the validity of constraints in families of graphs which is not our focus here and therefore, not further considered.

Stevens [35] discussed similar challenges in the specific context of bidirectional transformations. There, consistency is a property of a pair of models (or graphs) rather than between a graph and constraint. It may be argued that our formalisation generalises that of [35], as consistency between two graphs could be represented as consistency between a graph and a constraint by creating a union graph and making the consistency condition explicit as a constraint. Several concepts are introduced in [35] that initially seem to make sense only in the specific context of bidirectional transformations (such as the idea of $\vec{R}$ candidates, which are best-possible consistency restorations that a bi-directional transformation can produce), but may provide inspiration for a further extension of our framework with corresponding concepts (e.g., with a partial ordering of consistency-improving rule applications based on the amount of improvement they produce).

8. Conclusions

In this paper, we have introduced a definition of graph consistency as a graduated property, which allows for graphs to be partially consistent w.r.t. a graph constraint, inducing a partial ordering between graphs based on the number of constraint violations they contain. Two new forms of transformation can be identified as consistency-sustaining and consistency-improving, respectively. They are properly located in between the existing notions of constraint-preserving and constraint-guaranteeing transformations. Lifting them to rules, we have presented criteria for determining whether a rule is consistency-sustaining or -improving w.r.t. a given graph constraint. We have practically validated these criteria in an application case from search-based model engineering. Our validation shows that our criteria, especially the one for consistency-sustainment, allow considerably reducing the manual effort for checking automatically generated rules.

While the criteria presented allow us to check a given rule against a graph constraint, their lifting to a set of constraints is the next step to go. Furthermore, efficient algorithms are needed that implement our static analysis of rules and are able to construct consistency-sustaining or -improving rules from a set of constraints.

Acknowledgement. We thank the anonymous reviewers for their helpful comments. This work was partially funded by the German Research Foundation (DFG), project “Triple Graph Grammars (TGG) 2.0” and research fellowship with project number 413074939.
References

URL https://doi.org/10.1007/3-540-31988-2

URL https://doi.org/10.1007/978-3-030-43916-3

URL https://doi.org/10.1007/978-3-642-16145-2_9

URL https://doi.org/10.1007/978-3-319-61470-0_12

URL https://doi.org/10.1017/S0960129508007202

URL https://doi.org/10.1016/S1095-8568(00)00036-X

URL https://doi.org/10.1007/978-3-030-04771-9_31


URL https://doi.org/10.1007/978-3-319-61473-1_12

URL https://doi.org/10.4204/EPTCS.309.5

URL https://doi.org/10.1109/MODELS.2019.00-10

URL https://doi.org/10.1002/smr.1800

URL https://doi.org/10.1007/978-3-030-51372-6_14

URL https://doi.org/10.1109/TSE.2010.70


[16] D. Strüber, Generating efficient mutation operators for search-based model-driven engineering, in: E. Guerra, M. van den Brand (Eds.), Theory and Practice of Model Transformation – 10th International Conference, ICMT@STAF 2017, Mar-
Proof (of Proposition 1). By definition. Hence, $0 \leq \text{morphism } c$. The last claim for $ncv(G_0, c)$ constitutes an upper bound on the possible length of sequences of consistency-improving transformations (w.r.t. $c$) starting at $G_0$. □

Proof (of Lemma 2). 1. Set $p_D(x) := g^{-1}(p(x))$ for all $x \in C$. Since $g^{-1}(p(C))$ belongs to the domain of $tr_t$ by assumption, this result in a graph morphism with the desired property. For the other direction, the existence of $p_D : C \rightarrow D$ with $p = g \circ p_D$ ensures that $p(C)$ belongs to the domain of $tr_t$, i.e., $tr_t \circ p$ is total.

2. The second statement is completely symmetrical by considering $tr_t^{-1}$ as a partial (injective) morphism from $H$ to $G$. □

Proof (of Theorem 1). Throughout the proof, let $c$ be the relevant constraint and $t : G \rightarrow_r m H$ a transformation.

We first show that a $c$-guaranteeing transformation is directly consistency-sustaining. By definition, guarantee of a constraint implies its preservation [5]. In particular, the statement that guarantee implies direct sustainment is true in the case of existential constraints. For the universal case, by $H \models c$, either $occ(H, c) = \text{ro}(H, c) = 0$ or $ncv(H, c) = 0$. In either case, the definition of direct consistency-sustainment is met: All preserved as well as all newly created occurrences of $c$ in $H$ are valid.
Next, we show that direct consistency-sustainment implies consistency-sustainment. In the case of existential constraints, if $G \models c$, then preservation implies $H \models c$ such that $ncv(G, c) = ncv(H, c) = 0$. If $G \not\models c$, i.e., if $ncv(G, c) = 1$, either $ncv(H, c) = 1$ or $ncv(H, c) = 0$ may hold. In both cases, $ncv(G, c) \geq ncv(H, c)$. In the case of universal constraints, the two conditions together imply that $ncv(G, c) \geq ncv(H, c)$. This is because any constraint violation in $H$ must stem from one that already exists in $G$.

Finally, let a consistency-sustaining transformation be given. If $G \models c$, then $0 = ncv(G, c) \geq ncv(H, c) \geq 0$, and $ncv(H, c) = 0$ implies $H \models c$. This means, the transformation is $c$-preserving.

Since the above statements are true on the transformation level, they can be directly lifted to the rule level.

Proof (of Theorem 2). Again, throughout the proof, let $c$ be the relevant constraint and $G \Rightarrow H$ a transformation. Note that, for both notions of improvement, an improving transformation $G \Rightarrow H$ assumes $G \not\models c$ by definition.

First, let $G \Rightarrow H$ be a $c$-guaranteeing transformation where $G \not\models c$. By Theorem 1, this transformation is consistency-sustaining in particular. Hence, when $c$ is an existential constraint, the transformation is directly consistency-improving by definition. When $c$ is a universal constraint, we have the following: $G \not\models c$ implies that there is an injective morphism $p : C \hookrightarrow G$ with $p \not\models d$. As $H \models c$ by definition of $c$-guaranteeing rule applications, either $tr_t \circ p$ is not total or $tr_t \circ p \models d$. This means that either the first or the second condition of the definition of a directly consistency-improving transformation is met. Therefore, the transformation is directly consistency-improving.

In the following we show that every directly consistency-improving transformation is consistency-improving. The last claim, that every consistency-improving transformation is consistency-sustaining, holds again by definition.

First, every directly consistency-improving transformation is directly consistency-sustaining by definition and by Theorem 1 every directly consistency-sustaining transformation is consistency-sustaining. This means that we only have to check the conditions on the number of constraint violations. By assumption $ncv(G, c) > 0$. When $c$ is an existential constraint, the transformation is even $c$-guaranteeing by definition and we obtain $H \models c$. Hence, $ncv(G, c) = 1 > 0 = ncv(H, c)$ and the transformation is consistency-improving. When $c$ is universal, there exists (at least) one occurrence $p : C \hookrightarrow G$ that meets either the first or the second condition of the formula. In either case, this has the effect of decreasing $ncv(G, c)$ by one. Moreover, direct consistency-sustaining ensures that no new occurrences are introduced that violate the constraint. In summary, $ncv(G, c) > ncv(H, c)$ and the transformation is consistency-improving.

That a consistency-improving transformation is also consistency-sustaining is immediate from the definition.

On the rule level, (direct) consistency-improvement is defined in such a way that at least one (directly) consistency-improving transformation via that rule needs to exist. Hence, the proven statements on the transformation level lift to the rule level as long as there exists a $c$-guaranteeing transformation via that rule starting at an inconsistent graph $G$.

The following proofs, which show the correctness of our criteria for direct consistency-sustainment and direct consistency-improvement, closely study how transformations interact with occurrences of graphs from a constraint. A simple but important observation is that the track morphism of a transformation preserves occurrences in the sense stated in the next lemma.

Lemma 3 (Interaction of track morphism and occurrences). Let a transformation $t : G \Rightarrow_{r,m} H$ with transformation morphisms $G \xrightarrow{\rho} D \xrightarrow{\alpha} H$ and a condition $c$ be given that contains an injective morphism $a_t : C_{i-1} \hookrightarrow C_i$. Given injective morphisms $p_{i-1} : C_{i-1} \hookrightarrow G$ and $p_i : C_i \hookrightarrow G$ such that $p_{i-1} = p_i \circ a_t$ and the track morphism $tr_t : G \hookrightarrow H$ is total when restricted to $p_i(C_i)$, then $p'_{i-1} = p_i' \circ a_t$, where $p_i' := tr_t \circ p_i$ and $p_i' := tr_t \circ p_i$. Likewise, given injective morphisms $p'_{i-1} : C_{i-1} \hookrightarrow C_i \hookrightarrow H$ such that $p_{i-1} = p_i' \circ a_t$ and $p_i(C_i)$ is contained in $tr_t(G)$, then $p_{i-1} = p_i' \circ a_t$, where $p_{i-1} := tr_t^{-1} \circ p_{i-1}$ and $p_i := tr_t^{-1} \circ p_i$. 28
Let a condition results, this lemma is proved inductively as explained in Remark 1.

Proof. First, Lemma 2 ensures that \( t_r \) being total when restricted to \( p_1(C_1) \) can equivalently be expressed by stating that there exists an (injective) morphism \( p_{1,D} : C_1 \hookrightarrow D \) such that \( p_1 = g \circ p_{1,D} \) (compare Fig. 7 or A.13). Moreover, we have \( p_1' := t_r \circ p_1 = h \circ p_{1,D} \). The analogously implied morphism \( p_{1-1,D} : C_{1-1} \hookrightarrow D \) with \( p_{1-1} = g \circ p_{1-1,D} \) is given as \( p_{1-1,D} = p_{1,D} \circ a_i \). We calculate

\[
\begin{align*}
g \circ p_{1,D} \circ a_i &= p_1 \circ a_i \\
&= p_{1-1} \\
&= g \circ p_{1-1,D},
\end{align*}
\]

which implies

\[ p_{1,D} \circ a_i = p_{1-1,D} \]

since \( g \) is injective. Composing this equation with \( h \) results in \( p_1' \circ a_i = p_1'_{-1} \) as desired.

The second statement follows in exactly the same manner, switching the roles of \( h \) and \( g \).

The following technical lemma is the key to showing that rules that interact in a consistency-sustaining manner with a constraint are indeed directly consistency-sustaining w.r.t. it. As well as most of the following results, this lemma is proved inductively as explained in Remark 1.

**Lemma 4 (Preservation and reflection).** Let a condition \( c = Q(a_1 : C_0 \hookrightarrow C_1, d) \) in ANF with \( nl(c) \geq 1 \) and a rule \( r \) be given such that \( r \) interacts in a consistency-sustaining manner with \( c \). In case \( c \) is an existential condition, all transformations \( t : G \Rightarrow_{r,m} H \) preserve satisfying morphisms: For every injective morphism \( p_0 : C_0 \hookrightarrow G \) such that \( p_0 \models c \) and \( t_r \circ p_0 \) is total, i.e., there exists an injective morphism \( p_{0,D} : C_0 \hookrightarrow D \) with \( g \circ p_{0,D} = p_0 \), the induced injective morphism \( p'_0 := h \circ p_{0,D} \) satisfies the condition \( c \).

In case \( c \) is a universal condition, all transformations \( t : G \Rightarrow_{r,m} H \) reflect occurrences and sustain validity of preserved occurrences: For every injective morphism \( p'_1 : C_1 \hookrightarrow H \) such that for \( p'_0 := p'_1 \circ a_1 \) we have \( p'_0(C_0) \subseteq tr_t(G) \) there is an injective morphism \( p_{1,D} : C_1 \hookrightarrow D \) such that \( h \circ p_{1,D} = p'_1 \). Moreover, for every injective morphism \( p_1 : C_1 \hookrightarrow G \) with \( p_1 \models d \), the existence of an injective morphism \( p_{1,D} : C_1 \hookrightarrow D \) such that \( g \circ p_{1,D} = p_1' \) implies \( p_1' := h \circ p_{1,D} \models d \).

Proof. We prove the statement using structural induction starting at conditions of nesting level 1. Throughout the whole proof let \( G \xleftarrow{g} D \xrightarrow{h} H \) be the transformation morphisms of a transformation step \( t : G \Rightarrow_{r,m} H \) via \( r \) at match \( m \) and compare Fig. A.13 for different occurring morphisms and their relations.

As induction basis, first assume \( c = \exists (a_1 : C_0 \hookrightarrow C_1, \text{true}) \) and let \( p_0 : C_0 \hookrightarrow G \) be such that \( p_0 \models c \) and \( t_r \circ p_0 \) is total. Satisfaction of \( c \) via \( p_0 \) implies the existence of an injective morphism \( p_1 : C_1 \hookrightarrow G \) such that \( p_1 \circ a_1 = p_0 \). Consider \( p_1 \) as a match for \( \text{check}_{C_1} \) in \( G \). Suppose transformation \( t \) to cause a conflict for the application of \( \text{check}_{C_1} \) at \( p_1 \). Without loss of generality, we can assume this conflict to be a weak critical pair; otherwise we can consider the weak critical pair that embeds into it. (We exemplify provide the full details for this line of reasoning in the proof of Lemma 5; this constitutes the most complex situation where we need this argument.) Since \( C_1 \) is bound by an existential quantifier, unproblematic deletions in conflicts implies that \( t \) already causes a conflict for the transformation \( G \Rightarrow_{\text{check}_{C_0}, p_0} G \). But this contradicts the totality of \( t_r \circ p_0 \). Thus, \( t \) cannot cause a conflict, which means that there exists an injective morphism \( p_{1,D} : C_1 \hookrightarrow D \) such that \( g \circ p_{1,D} = p_1 \). Moreover, by Lemma 3 we obtain \( p'_0 \circ a_1 = p'_0 \) for \( p' := h \circ p_{1,D} \), where \( j = 0,1 \). In particular \( p'_0 = c \) as \( p'_0 \models \text{true} \).

Secondly, assume \( c = \forall (a_1 : C_0 \hookrightarrow C_1, \text{false}) \). Sustainment of validity is trivially true in this case, as no morphism satisfies \( \text{false} \). Thus, assume \( p'_1 : C_1 \hookrightarrow H \) to be an occurrence of \( C_1 \) in \( H \) such that for \( p'_0 := p'_1 \circ a_1 \) we have \( p'_0(C_0) \subseteq tr_t(G) \). We can consider \( p'_1 \) as a match for rule \( \text{check}_{C_1} \) in \( H \). By assumption, any possible dependency of this match on the transformation \( t \) is unproblematic (as specified in Condition 2 of Def. 15). However, an unproblematic dependency implies that \( p'_0(C_0) \subseteq tr_t(G) \) does not hold (as at least one of the elements from \( p'_0(C_0) \) gets newly created by transformation \( t \)). Therefore, the application of \( \text{check}_{C_1} \) at match \( p'_1 \) is sequentially independent from \( t \), i.e., there is an injective morphism \( p_{1,D} : C_1 \hookrightarrow D \) such that \( h \circ p_{1,D} = p'_1 \) as desired.
For the inductive step, first let \( c = \exists (a_1 : C_0 \hookrightarrow C_1, d) \) be an existential condition with \( d = \forall (a_2 : C_1 \hookrightarrow C_2, d') \) being a universal one such that \( nl(d) \geq 1 \). Let \( p_0 : C_0 \rightarrow G \) be such that \( p_0 \models c \). Like above, we obtain \( p'_1 : C_1 \hookrightarrow H \) such that \( p'_1 \circ a_1 = p'_0 \); in particular, \( p'_1(C_1) \subseteq tr_t(G) \). We have to show that \( p'_1 \models d \), i.e., that \( p'_2 \models d' \) for any injective morphism \( p'_2 : C_2 \hookrightarrow H \) with \( p'_2 \circ a_2 = p'_1 \). Thus, let \( p'_2 \) be such a morphism. By the inductive hypothesis, reflection of occurrences holds for the condition \( d \). This means, we obtain an injective morphism \( p_{2,D} : C_2 \hookrightarrow D \); moreover, by Lemma 3, the induced morphism \( p_2 := g \circ p_{2,D} \) satisfies \( p_2 \circ a_2 = p_1 \). In particular, since \( p_0 \models c \), necessarily \( p_2 \models d' \). By the induction hypothesis, since \( t \) preserves this valid occurrence, it also sustains its validity, which means that \( p'_2 \models d' \).

Secondly, let \( c = \forall (a_1 : C_0 \hookrightarrow C_1, d) \) be a universal condition with \( d = \exists (a_2 : C_1 \hookrightarrow C_2, d') \) being an existential one such that \( nl(d) \geq 1 \). Reflection of occurrences is exactly shown as in the base step of the induction. Thus, let \( p_1 : C_1 \hookrightarrow G \) be such that \( p_1 \models d \). We have to show that, whenever \( p_{1,D} : C_1 \hookrightarrow D \) with \( g \circ p_{1,D} = p_1 \) exists, also \( p'_1 := h \circ p_{1,D} \models d \). But existence of \( p_{1,D} \) just means that \( tr_t \circ p_1 \) is total. By inductive hypothesis and since \( d \) is existential, the validity of \( d \) is preserved by the transformation \( t \), i.e., \( p'_1 \models d \).

**Proof (of Theorem 3).** The whole proof is a specialisation of the above lemma to the case of constraints. Let \( t : G \Rightarrow_{r,m} H \) be a transformation via \( r \) and \( c \) be a constraint in ANF with \( nl(c) \geq 1 \) such that \( r \) and \( c \) interact in a consistency-sustaining manner.

First, assume that \( c = \exists (a_1 : \emptyset \hookrightarrow C_1, d) \) is existential. In this case, we have to show that \( G \models c \) implies \( H \models c \). If \( G \models c \), i.e., if \( i_G : \emptyset \hookrightarrow G \) is the empty morphism, Lemma 4 implies that \( i_H : \emptyset \hookrightarrow H \models c \) since \( tr_t \) is always total on the empty graph.

If \( c = \forall (a_1 : \emptyset \hookrightarrow C_1, d) \) is universal, we have to check the two requirements defining direct consistency-sustainment. The first requirement is directly stated in Lemma 4. Moreover, this lemma also ensures that the second condition is true, namely for trivial reasons: There is no \( p'_1 : C_1 \hookrightarrow H \) such that no \( p_1 : C_1 \hookrightarrow G \) exists with \( p'_1 = tr_t \circ p_1 \). □

**Proof (of Proposition 3).** We show that \( r \) interacts sustainingly with \( c \). Then, Theorem 3 ensures the correctness of the statement.

Hence, first, let \( C_i \) be an existentially bound graph of \( c \). Let there be a pair of transformations \( (t_1 : G \Rightarrow_{r,m} H, t_2 : G \Rightarrow_{check,c_1,p_1} G) \) such that \( t_1 \) causes a conflict for \( t_2 \). This means, the set \( m(L \setminus K) \cap p_1(C_i) \) is not empty. We have to show that Eq. (1) holds, i.e., that one of the elements already stems from \( C_{i-1} \). For this, consider the graph \( P \) that arises by restricting \( G \) to the images of \( m \) and \( p_1 \). The morphisms \( m \)

![Relations of satisfying morphisms](image-url)
and \( p_i \) (considered as morphisms with codomain \( P \)) are a pair of injective and jointly surjective morphisms. Since \( m \) is a match for \( r \), in particular \( m \models ac_2 \). This means, there is no \( j \in CL_i \) such that \( P \cong P_j \). Therefore, \( m(L \setminus K) \cap p_i(C_i) \not\subseteq p_i(C_i \setminus a_i(C_i - 1)) \) which is equivalent to Eq. (1).

Next, let \( C_i \) be a universally bound graph of \( c \). Let there be a sequence of transformations \( G \Rightarrow_{r,m} H \Rightarrow_{check_{c,p}} H \). We have to show that the second transformation is sequentially independent from the first. Since \( check_{c_i} \) does not delete elements and is not equipped with an application condition, for this it is enough to show that \( n(R) \cap p_i(C_i) \subseteq n(K) \) or, equivalently, \( n(R \setminus K) \cap p_i(C_i) = \emptyset \) where \( n \) is the comatch of the first transformation step. We consider the graph \( P \) that arises by restricting \( H \) to the images of \( n(R) \) and \( p_i(C_i) \). The morphisms \( n \) and \( p_i \) (considered as morphisms with codomain \( P \)) are a pair of injective and jointly surjective morphisms. Again, since \( m \) is a match for \( r, m \models ac_2 \). In particular, if there were a graph \( P_j, j \in CR_i \), such that \( P \cong P_j \), by construction of Left, \( ac_2 \) would have blocked the application of \( r \) at \( m \). This means, \( p_i(C_i) \cap n(R \setminus K) = \emptyset \), as desired. \( \square \)

Again, we first introduce a technical lemma on which we will base the proof of Proposition 4. In this proof, we exemplify the details of how a property we require for all weak critical pairs (resp. all weak critical sequences) is even valid for all pairs of conflicting transformations (sequentially dependent transformations), then.

**Lemma 5 (Preservation and reflection II).** Let a condition \( c = Q(a_1 : C_0 \leftarrow C_1, d) \) in ANF with \( n(c) \geq 1 \) and a rule \( r \) be given such that \( r \) and \( c \) interact constraint-compatibly. In case \( c \) is an existential condition, all transformations \( t : G \Rightarrow_{r,m} H \) preserve satisfying morphisms: For every injective morphism \( p_0 : C_0 \leftarrow G \) such that \( p_0 \models c \) and \( t r \circ p_0 \) is total, i.e., there exists an injective morphism \( p_{0,D} : C_0 \leftarrow D \) with \( g \circ p_{0,D} = p_0 \), the induced injective morphism \( p_0' := h \circ p_{0,D} \) satisfies the condition \( c \).

In case \( c \) is a universal condition, all transformations \( t : G \Rightarrow_{r,m} H \) satisfy newly created occurrences and sustain validity of preserved occurrences: For every injective morphism \( p_1 : C_1 \leftarrow H \), if there is no injective morphism \( p_{1,D} : C_1 \leftarrow D \) such that \( h \circ p_{1,D} = p_1' \), then \( p_1' \models d \). Moreover, for every injective morphism \( p_1 : C_1 \leftarrow G \) with \( p_1 \models d \), the existence of an injective morphism \( p_{1,D} : C_1 \leftarrow D \) such that \( g \circ p_{1,D} = p_1' \) implies \( p_1' \models h \circ p_{1,D} \models d \).

**Proof.** Again, we prove the statement using structural induction for conditions in ANF with nesting level \( \geq 1 \). Throughout the whole proof let \( G \Rightarrow_{r,m} D \Rightarrow_{h} H \) be a span resulting from a transformation \( t : G \Rightarrow_{r,m} H \) via \( r \) at match \( m \).

As induction basis, first assume \( c = \exists(a_1 : C_0 \leftarrow C_1, \text{true}) \) and let \( p_0 : C_0 \leftarrow G \) be such that \( p_0 \models c \) and \( t r \circ p_0 \) is total, i.e., there exists an injective morphism \( p_{0,D} : C_0 \leftarrow D \) such that \( g \circ p_{0,D} = p_0 \) (compare Lemma 2); again \( p_0' := h \circ p_{0,D} \). We have to show \( p_0' \models c \). Satisfaction of \( c \) via \( p_0 \) implies the existence of an injective morphism \( p_1 : C_1 \leftarrow G \) such that \( p_1 \circ a_1 = p_0 \). Either \( t \) does not cause a conflict for the application of \( check_{c_1} \) at match \( p_1 \). Then we have \( p_0' \models c \) exactly as in the proof of Lemma 4. Or \( t \) causes such a conflict. By completeness of critical pairs and every critical pair being a weak critical pair, there exists a weak critical pair \( (X \Rightarrow_{r,a_1} H_X, X \Rightarrow_{check_{c_1},a_2} X) \) that embeds into the conflicting pair of transformations via an injective morphism \( f : X \leftarrow G \) such that \( f \circ a_1 = m \) and \( f \circ o_2 = p_1 \) (see, e.g., [36, Definition 5.40 and Theorem 5.41]). This whole situation is depicted in Fig. A.14, where all four squares a pushouts and \( p_0 = p_1 \circ a_1 \) is omitted.

First, since \( D_X \) is also a pullback object and \( g \circ p_{0,D} = p_0 = p_1 \circ a_1 = f \circ o_2 \circ a_1, \) there exists an injective morphism \( p_{0,D_X} : C_0 \leftarrow D_X \) such that \( p_{0,D} = f_D \circ p_{0,D_X} \) and \( p_{0,X} := o_2 \circ a_1 = g_X \circ p_{0,D_X} \). Moreover, there cannot exist any injective morphism \( p_{1,D_X} : C_1 \leftarrow D_X \) such that \( g_X \circ p_{1,D_X} = o_2 \). Otherwise, we would obtain \( g \circ f_D \circ p_{1,D_X} = f \circ g_X \circ p_{1,D_X} = o_2 \circ f = p_1 \), which would contradict the transformation \( t \) to cause a conflict for \( G \Rightarrow_{check_{c_1},p_1} G \). This means, the weak critical sequence constitutes a problematic conflict (i.e., Eq. 1 is violated). Thus, by assumption, there is an injective morphism \( p_{1,X} : C_1 \leftarrow H_X \) such that
Figure A.14: Preservation of satisfaction via constraint-compatible conflicts

\[ p'_{0,X} := h_X \circ p_{0,D_X} = \bar{p}'_{1,X} \circ a_1. \]  
For \( p'_1 := f_H \circ \bar{p}'_{1,X} \) we can then compute

\[
p'_1 \circ a_1 = f_H \circ \bar{p}'_{1,X} \circ a_1 = f_H \circ h_X \circ p_{0,D_X} = h \circ f_D \circ p_{0,D_X} = h \circ p_0,\]

Hence, \( p'_0 \models c. \)

Secondly, assume \( c = \forall (a_1 : C_0 \leftrightarrow C_1, \text{false}). \) Sustainment of validity is trivially true in this case, as no morphism satisfies \text{false}. Moreover, there cannot be any new occurrence of \( C_1 \) in \( H \) because this would induce a weak critical sequence for \( r \) and \text{check}_{C_1}, \) which is excluded since \( C_1 \) is the last graph of \( c. \)

For the inductive step, first let \( c = \exists (a_1 : C_0 \leftrightarrow C_1, d) \) be an existential condition with \( d = \exists (a_2 : C_1 \leftrightarrow C_2, d') \) being a universal one such that \( \#(d) \geq 1. \) Let \( p_0 : C_0 \leftrightarrow G \) be such that \( p_0 \models c. \) Like in the base step, we obtain \( p'_1 : C_1 \leftrightarrow H \) such that \( p'_1 \circ a_1 = p'_0. \) We have to show that \( p'_1 \models d, \) i.e., that \( p'_2 \models d' \) for any injective morphism \( p'_2 : C_2 \leftrightarrow H \) with \( p'_2 \circ a_2 = p'_1. \) Thus, let \( p'_2 \) be any such morphism. Either (i), there exists an injective morphism \( p_{2,D} : C_2 \leftrightarrow D \) such that \( p'_2 = h \circ p_{2,D}. \) By Lemma 3, we obtain \( p_2 \circ a_2 = p_1 \) for \( p_2 := g \circ p_{2,D}. \) In particular, \( p_2 \models d' \) as \( p_1 \models d. \) Thus, by inductive hypothesis, \( p'_2 \models d'. \)

Or (ii), no such morphism \( p_{2,D} \) exists. In that case, we directly obtain \( p'_2 \models d' \) by the inductive hypothesis.

Finally, let \( c = \forall (a_1 : C_0 \leftrightarrow C_1, d) \) be a universal condition with \( d = \exists (a_2 : C_1 \leftrightarrow C_2, d') \) being an existential one such that \( \#(d) \geq 1. \) First, let \( p'_1 : C_1 \leftrightarrow H \) be a new occurrence, i.e., assume no injective morphism \( p_{1,D} : C_1 \leftrightarrow D \) to exist such that \( p'_1 = h \circ p_{1,D}. \) In this case, the transformation \( H \Rightarrow \text{check}_{c,X}, p'_2, H \) sequentially depends on the given transformation \( t. \) As for the conflicting case above, completeness of critical pairs (and the duality between conflicts and dependencies) provides us with a weak critical sequence \( G_X \Rightarrow r.o. : X \Rightarrow \text{check}_{C_1}, a_2, X \) that can be embedded into the given sequence via an injective morphism \( f : X \Rightarrow H; \) in particular, \( f \circ a_2 = p'_1. \) By assumption, there thus exists an injective morphism \( p'_2 : C_2 \leftrightarrow X \) such that \( p'_2 \circ a_2 = a_2. \) By composition with \( f \) we obtain \( p'_2 := f \circ p'_2 \) which satisfies \( p'_2 \circ a_2 = f \circ p'_2 \circ a_2 = p_0; \) \( \circ a_2 = p'_1. \) We have to show that \( p'_2 \models d'. \) However, \( d' \) is either \text{true} such that \( p'_2 \models d' \) trivially holds, or it is another universal condition with \( \#(d') < \#(c). \) Then \( d' = \forall (a_3 : C_2 \leftrightarrow C_3, d'' \text{ and every occurrence of \( C_3 \) in \( H \) extending \( p'_1 \) is necessarily new (otherwise the original one of \( C_1 \) could not be). Thus, by the inductive hypothesis on universal conditions, again \( p'_2 \models d'. \)
Second, let \( p_1 : C_1 \hookrightarrow G \) be such that \( p_1 \models d \) and an injective morphism \( p_{1,D} : C_1 \hookrightarrow D \) with \( p_1 = g \circ p_{1,D} \) exists. As \( tr_r \circ p_1 \) is total (as guaranteed by the existence of \( p_{1,D} \)) and \( d \) is an existential constraint, the induced morphism \( p'_1 := h \circ p_{1,D} \) satisfies \( d \) by the inductive hypothesis on existential conditions. \( \square \)

**Proof (of Proposition 4).** Basically, this proposition follows from the above lemma in the same way as Theorem 3 follows from Lemma 4. The only difference is that in the case of universal constraints, Lemma 5 directly states the second requirement for direct sustainment (instead of ensuring it in a trivial way). \( \square \)

**Lemma 6.** Let \( c = Q(a_1 : C_0 \hookrightarrow C_1, d) \) be a condition over \( C_0 \) in ANF with \( nl(c) \geq 1 \) and \( t : G \Rightarrow_{r,m} H \) a transformation.

Whenever \( c \) is existential, \( p_0 : C_0 \rightarrow G \not\models c \), \( p'_0 := tr_r \circ p_0 \) is total, and \( p'_0 \models c \), then \( r \) has a consistency-improving interaction with \( c \).

Whenever \( c \) is universal and there exists an injective morphism \( p_1 : C_1 \hookrightarrow G \) with \( p_1 \not\models d \) such that for \( p_0 := p_1 \circ a_1 \) the morphism \( tr_r \circ p_0 \) is total and \( p'_1 := tr_r \circ p_1 \) is either (i) total and \( p'_1 \models d \) or (ii) is not total, then \( r \) has a consistency-improving interaction with \( c \).

**Proof.** Again, we prove the statement using structural induction for conditions in ANF. Let the transformation \( t \) be given by the span \( G \xrightarrow{g} D \xleftarrow{d} H \).

For the inductive basis, first assume \( c = \exists (a_1 : C_0 \hookrightarrow C_1, \text{true}) \) to be existential and \( p_0 : C_0 \rightarrow G \) be such that \( p_0 \not\models c \), \( p'_0 := tr_r \circ p_0 \) is total, and \( p'_0 \models c \). This means, there exists an injective morphism \( p'_1 : C_1 \hookrightarrow H \) such that \( p'_1 \circ a_1 = p'_0 \). In particular, the transformation \( H \Rightarrow_{\text{check}, C_1, p'_1} H \) sequentially depends on \( t \); otherwise, the morphism \( p_1 := g \circ p_{1,D} \) would witness \( p_0 \models c \) (see Lemma 3). Moreover, this dependency satisfies

\[
    n(R \setminus K) \cap p'_1(a_1(C_0)) = \emptyset, \tag{A.1}
\]

where \( n \) is the co-match of the transformation, i.e., Eq. 4 holds; otherwise, the morphism \( p_{0,D} \) relating \( p_0 \) and \( p'_0 \) could not exist. Again, without loss of generality, this dependency constitutes a weak critical sequence. In particular, \( r \) has a consistency-improving interaction with \( c \).

Second, assume \( c = \forall (a_1 : C_0 \hookrightarrow C_1, \text{false}) \) to be universal and \( p_1 : C_1 \hookrightarrow G \) an injective morphism such that \( tr_r \circ p_0 \) is total, where \( p_0 := p_1 \circ a_1 \), and \( p_1 \not\models d \) = false (which is trivially true). If \( p'_1 := tr_r \circ p_1 \) is total, \( p'_1 \models d = \text{false} \) cannot hold. So we only need to consider the case where \( tr_r \circ p_1 \) is not total. In that case, the transformation \( t \) causes a conflict for the transformation \( G \Rightarrow_{\text{check}, C_1, p_1} G \) such that

\[
    m(L \setminus K) \cap p_1(a_1(C_0)) = \emptyset, \tag{A.2}
\]

i.e., Eq. 3 is satisfied. Otherwise, the morphism \( p'_0 \) could not be total. Again, without loss of generality, this conflict constitutes a weak critical pair. In particular, \( r \) has a consistency-improving interaction with \( c \).

For the inductive step, again first assume \( c = \exists (a_1 : C_0 \hookrightarrow C_1, d) \) to be existential, \( d \) a universal condition with \( nl(d) \geq 1 \), and \( p_0 : C_0 \rightarrow G \) be such that \( p_0 \not\models c \), \( p'_0 := tr_r \circ p_0 \) is total, and \( p'_0 \models c \). First, \( p'_0 \models c \) implies that there exists an injective morphism \( p'_1 : C_1 \hookrightarrow H \) such that \( p'_1 \circ a_1 = p'_0 \) and \( p'_1 \models d \). Either (i), there exists an injective morphism \( p_{1,D} : C_1 \hookrightarrow D \) such that \( h \circ p_{1,D} = p'_1 \). Then, by Lemma 3, \( p_1 := g \circ p_{1,D} \) satisfies \( p_1 \circ a_1 = p_0 \). Hence, \( p_1 \not\models d \) (otherwise, \( p_0 \models c \)). But \( tr_r \circ p_1 = p'_1 \) is total. Thus, the inductive hypothesis (on universal conditions) applies to \( p_1 \) and \( d \) and \( r \) has a consistency-improving interaction with \( c \). Or (ii), no such morphism \( p_{1,D} \) exists. In that case, a weak critical sequence satisfying Eq. 4 exists exactly as shown in the inductive base step. Again, \( r \) has a consistency-improving interaction with \( c \).

Finally, let \( c = \forall (a_1 : C_0 \hookrightarrow C_1, d) \) be universal, \( d \) an existential condition with \( nl(d) \geq 1 \), and \( p_1 : C_1 \hookrightarrow G \) an injective morphism such that \( tr_r \circ p_0 \) is total, where \( p_0 := p_1 \circ a_1 \), and \( p_1 \not\models d \). First, assume that \( p'_1 := tr_r \circ p_1 \) is total and \( p'_1 \models d \). In that case, the inductive hypothesis (on existential conditions) applies to \( p_1 \) and \( d \). Hence, \( r \) has a consistency-improving interaction with \( c \). Secondly, assume \( p'_1 \) to not be total. In that case, a weak critical pair satisfying Eq. 3 exists exactly as shown in the inductive base step. Again, \( r \) has a consistency-improving interaction with \( c \). \( \square \)

**Proof (of Theorem 4).** Again, to prove Theorem 4 one just instantiates Lemma 6 to the special case of conditions, noting that on the rule level, direct consistency-improvement in particular implies the existence of a directly consistency-improving transformation \( G \Rightarrow_{r,m} H \). \( \square \)