Rule-based Graph Repair using Minimally Restricted Consistency-Improving Transformations

Alexander Lauer

February 23, 2023

Selbstständigkeitserklärung

Hiermit versichere ich, Alexander Lauer, dass ich die vorliegende Arbeit mit dem Titel *Rule-based Graph Repair using Minimally Restricted Consistency-Improving Transformations* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Masterarbeit wurde in der jetzigen oder ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen anderen Prüfungszwecken gedient.

Amöneburg, den 23. Februar 2023

Abstract

Model-driven software engineering is a suitable method for dealing with the ever-increasing complexity of software development processes. Graphs and graph transformations have proven useful for representing such models and changes to them. These models must satisfy certain sets of constraints. An example are the multiplicities of a class structure. During the development process, a change to a model may result in an inconsistent model that must be transformed into a consistent model. This problem is called *rule-based graph repair* and is defined as follows: Given a graph G, a constraint c such that G does not satisfy c, and a set of rules $\mathcal{R}$, use the rules of $\mathcal{R}$ to construct a graph that satisfies c.

In this paper we introduce new notions of consistency, which we call *consistency-maintaining* and *consistency-increasing* transformations and rules, respectively. These notions are finer than those already known. Finer in the sense that even the smallest changes, insertions or deletions of individual edges or nodes can be considered as increasing or decreasing consistency. Furthermore, we extend these notions to *direct consistency-maintaining* and *direct consistency-maintaining* transformations and rules respectively, which prohibit the insertion of new violations altogether, and compare our notions with the existing ones to reveal connections and differences.

We present methods for constructing application conditions that are *direct consistency-maintaining* or *direct consistency-increasing*. In the latter case, we present two types of application conditions, one for general rules and one for a particular set of rules, which we call *basic increasing rules*. The application conditions for *basic increasing rules* have the advantage that they are less complex and restrictive than the application conditions for general rules.

Finally, we present an *rule-based graph repair* approach that is able to repair certain constraints, which we call *circular conflict free constraints*, and certain sets of constraints, which we call *circular conflict free set of constraints*.

Zusammenfassung

Model-driven software engineering ist eine geeignete Methode, um die ständig wachsende Komplexität von Softwareentwicklungsprozessen zu bewältigen. Graphen und Graphtransformationen haben sich bewährt, um solche Modelle und Änderungen der Modelle darzustellen. Diese Modelle müssen bestimmte Mengen von Bedingungen (constraints) erfüllen. Ein einfachen Beispiel hierfür sind die Multiplizitäten einer Klassenstruktur. Während des Entwicklungsprozesses kann eine Änderung eines Modelles jedoch zu einem inkonsistenten Modell führen, welches wieder in ein konsistentes Modell überführt werden muss. Dieses Problem heißt *rule-based graph repair* und ist folgendermaßen definiert: Seien ein Graph G, ein constraint c, sodass G nicht c erfüllt, und eine Menge von Regeln $\mathcal{R}$ gegeben, verwende die Regeln von $\mathcal{R}$, um einen Graphen zu konstruieren, der c erfüllt.

In dieser Arbeit führen wir neue Begriffe von Konsistenz ein, die wir *consistency-maintaining* bzw. *consistency-increasing* Transformationen und Regeln nennen. Diese

Begriffe sind feiner im Vergleich zu bereits bekannten Begriffen. Feiner in dem Sinne, dass auch kleinste Änderungen, das Einfügen oder Löschen von einzelnen Kanten oder Knoten als eine Vergrößerung oder Verschlechterung der Konsistenz angesehen werden kann. Des Weiteren erweitern wir diese Begriffe zu den Begriffen *direct consistency-maintaining* bzw. *direct consistency-increasing* Transformationen und Regeln, die das Einfügen von neuen Verletzungen gänzlich verbieten, und vergleichen unsere Begrifflichkeiten mit den bereits existierenden, um Zusammenhänge und Unterschiede aufzudecken.

Wir präsentieren Methoden, um Awendungsbedingungen zu konstruieren, die *direct consistency-maintaining* bzw. *direct consistency-increasing* sind. Im zweiten Fall präsentieren wir zwei Arten von Anwendungsbedingungen, einmal für allgemeine Regeln und einmal für eine bestimmte Menge von Regeln, die wir *basic increasing rules* nennen. Anwendungsbedingungen für *basic increasing rules* haben den Vorteil, dass diese weniger komplex und restriktiv sind als die Anwendungsbedingungen für allgemeine Regeln.

Schließlich stellen wir einen *rule-based graph repair* Ansatz vor, der in der Lage ist, bestimmte constraints, die wir *circular conflict free constraints* nennen, und bestimmte Mengen von constraints, die wir *circular conflict free set of constraints* nennen, zu reparieren.

Contents

1 Introduction

Model-driven software engineering is a suitable tool to deal with the increasing complexity of software development processes. Graphs and graph transformations have emerged as a suitable framework for model-driven software engineering, where models are represented by graphs and changes of models are represented by graph transformations. These models need to be consistent with respect to certain constraints, e.g. multiplicities if the model represents an object diagram. The concept of nested graph constraints has proven to be suitable for expressing these constraints [9]. As models are changed during development, the new model may become inconsistent and consistency must be restored.

The problem of restoring this consistency is called *graph repair*: Given a constraint c and an inconsistent graph G, derive a graph H such that H satisfies the constraint c. In particular, we will consider the problem of *rule-based graph repair*, which is defined as: Given a graph G, a constraint c and a set of rules $\mathcal{R}$, derive a graph H that satisfies c using the rules of $\mathcal{R}$. Because of the versatility of graphs and graph transformations, the concept of graph repair can be used to resolve inconsistencies for all kinds of graph-like structures. For example, in traffic light control or road networks.

There are several rule-based graph repair approach and we will now briefly discuss some of them. Habel and Pennemann have introduced the notions of *consistency-preserving* and *consistency-guaranteeing* transformations [10]. As the name suggests, via these binary notions make it possible to decide whether a transformation guarantees or preserves the consistency of the derived graph. In addition, they presented application conditions which guarantee that a transformation via rules equipped with it them is consistency-preserving or consistency-maintaining respectively. Kosiol et al. have presented the graduated notions of *consistency-sustaining* and *consistency-improving* transformations and rules [12]. In contrast to the notions of consistency-preserving and consistency-guaranteeing it is possible to evaluate the magnitude of inconsistency of a graph. They have also presented consistency-sustaining application conditions. Nassar et al. have optimised the construction of the application conditions mentioned above [14]. There is a rule-based graph repair approach for so-called proper constraints introduced by Sandmann and Habel [18], but this approach is not able to repair multiple constraints. Nassar et al. have presented a repair approach to repair multiplicities for EMF models [16, 15].

In this paper we will introduce a rule-based graph repair process for a certain subset of the set of nested conditions in alternating normal-form (ANF), so-called *circular conflict free constraints*. The main idea of our approach is to increase the consistency of a graph level by level. That is, given a constraint $c = \forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \ldots))))$ and a inconsistent graph G, in first step we repair the graph such that the derived graph satisfies $\forall(C_1, \exists(C_2, \mathsf{true}))$. In the next set, we repair the graph until it satisfies $\forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \mathsf{true}))))$ and so on, until finally the derived graph satisfies c. Note that we always repair two nesting levels at a time, since $\forall(C_1, \mathsf{true})$ is always satisfied and $\forall(C_1, \mathsf{false})$ immediately implies the satisfaction of c. Therefore, a process that repairs

only one nesting level would return a consistent graph after at most two iterations, with the drawback that many occurrences of graphs are simply deleted.

In order to do this, we will introduce new notions of consistency, called *consistency-maintaining* and *consistency-increasing* transformations and rules. As the name suggests, *consistency-maintaining* transformations do not decrease consistency, and *consistency-increasing* transformations actually increase the consistency. These notions are based on the first two levels of a constraint that are not satisfied. That is, given a graph G that satisfies $\forall(C_1, \exists(C_2, \mathsf{true}))$ but not $\forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \mathsf{true}))))$ only occurrences of C_3 are considered. This is because only occurrences of C_3 need to be repaired in order to derive a graph from G that satisfies $\forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \mathsf{true}))))$. In particular, our notions are also able to detect the smallest changes in consistency, namely, the insertion or deletion of single elements, which will lead to a more consistent graph. Therefore, our notions are more fine-grained than the notions of consistency-preserving, consistency-guaranteeing, consistency-sustaining and consistency-improving transformations and rules.

In addition, we will refine these notions to the notions of *direct consistency-maintaining* and *direct consistency-increasing* transformations and rules that completely prohibit the insertion of new violations. We will compare our new notions with those described above to ensure that they are indeed new notions of consistency, and to point out similarities and differences. We will present direct consistency-maintaining and two types of direct consistency-increasing application conditions. One for general rules and one for a specific set of rules called *basic increasing rules*. For basic increasing rules, we are able to construct less restrictive and less complex direct consistency-increasing application conditions compared to the general ones.

Finally, we present two rule-based graph repair approaches. The first for one constraint and the second for specific sets of constraints, called *circular conflict free sets of constraints*, which uses our repair approach for one constraint. Both processes make use of a given set of rules $\mathcal{R}$ and we present a characterisation of when such a set of rules is able to repair the constraint or the set of constraints, respectively. Of course, this is only a sufficient criterion, since it depends strongly on the input graph whether a consistent graph can be derived by applying the rules of $\mathcal{R}$.

This paper is structured as follows: Formal prerequisites are introduced in section 2. The notions of (direct) consistency-maintaining and (direct) consistency-increasing transformations and rules are given in section 3 and the construction of application conditions and characterisation of basic rules is given in section 4. The repair process is presented in section 5, and we summarise related graph repair approaches in section 6, before concluding the paper with section 7.

2 Preliminaries

Our graph repair process is based on the concept of the double-pushout approach [7]. In this chapter we introduce some formal prerequisites such as graphs, graph morphisms, nested graph conditions and constraints, and graph transformations.

2.1 Graphs and Graph morphisms

We start by introducing graphs and graph morphisms according to [7].

Definition 2.1 (**graph**). *A* graph $G = (V, E, \mathrm{src}, \mathrm{tar})$ *consists of a set of vertices (or nodes)* V*, a set of edges* E *and two mappings* $\mathrm{src}, \mathrm{tar} : E \to V$ *that assign the source and target vertices to an edge. The edge* $e \in E$ *connects the vertices* $\mathrm{tar}(e)$ *and* $\mathrm{src}(e)$*.*

If no tuple as above is given, V_G*,* E_G*,* tar_G *and* src_G *denote the sets of vertices, edges and target and source mappings, respectively.*

For the rest of this paper we will assume that all graphs are finite, i.e. given a graph G, the sets V_G and E_G are finite.

Definition 2.2 (**graph morphism**). *Let graphs* G *and* H *be given. A* graph morphism $f : G \to H$ *consists of two mappings* $f_V : V_G \to V_H$ *and* $f_E : E_G \to E_H$ *such that the source and target functions are preserved. This means*

$$f_V \circ \mathrm{src}_G = \mathrm{src}_H \circ f_E \text{ and}$$
$$f_V \circ \mathrm{tar}_G = \mathrm{tar}_H \circ f_E$$

holds. A graph morphism f *is called injective (surjective) if* f_E *and* f_V *are injective (surjective) mappings. An injective morphism* $f : G \to H$ *is called* inclusion *if* $f_E(e) = e$ *and* $f_V(v) = v$ *for all edges* $e \in E_G$ *and all nodes* $v \in V_G$*. If* f *is injective, it is denoted with* $f : G \hookrightarrow H$*. Two morphisms* $f_1 : G_1 \to H$ *and* $f_2 : G_2 \to H$ *are called* jointly surjective *if for each element* e *of* H *either an element* $e' \in G_1$ *with* $f_1(e') = e$ *or an element* $e' \in G_2$ *with* $f_2(e') = e$ *exists.*

For our newly introduced notions of consistency increase and maintainment, we also need to consider *subgraphs*, *overlaps* of graphs, and so-called *intermediate graphs*. Intuitively, intermediate graphs are graphs G' which lie between two given graphs G and H. That is, G is a subgraph of G' and G' is a subgraph of H.

Definition 2.3 (**subgraph**). *Let the graphs* G *and* H *be given. Then* G *is called a* subgraph *of* H *if an inclusion* $p : G \hookrightarrow H$ *exists.* G *is called a* proper subgraph of H *if the* p *is not bijective.*

Note that since the inclusion can also be surjective, by this definition every graph G is a subgraph of itself.

Definition 2.4 (**intermediate graph**). *Let* G *and* H *be graphs such that* G *is a subgraph of* H*. A graph* C *is called an* intermediate graph *of* G *and* H*, if* G *is a proper subgraph of* C *and* C *is a subgraph of* H*. The set of intermediate graphs of* G *and* H *is denoted by* $\mathrm{IG}(G, H)$*.*

Definition 2.5 (**overlap**). *Let the graphs* G_1 *and be* G_2 *be given. An* overlap $P = (H, i_{G_1}, i_{G_2})$ *consists of a graph* H *and a jointly surjective pair of injective morphisms* $i_{G_1} : G_1 \hookrightarrow H$ *and* $i_{G_2} : G_2 \hookrightarrow H$ *with* $i_{G_1}(G_1) \cap i_{G_2}(G_2) \neq \emptyset$*. The set of all overlaps of* G_1 *and* G_2 *is denoted by* $\mathrm{ol}(G_1, G_2)$*. If a tuple as above is not given, then* G_P*,* $i^P_{G_1}$ *and* $i^P_{G_2}$ *denote the graph and morphisms of a given overlap* $P \in \mathrm{ol}(G_1, G_2)$*.*

Note that (H, i_{G_1}, i_{G_2}) with i_{G_1} and i_{G_2} being jointly surjective and $i_{G_1}(G_1) \cap i_{G_2}(G_2) = \emptyset$ could also be considered as an overlap of G_1 and G_2. In this paper we only need to consider overlaps with $i_{G_1}(G_1) \cap i_{G_2}(G_2) \neq \emptyset$. So we have embedded this property directly into the definition.

As mentioned above, our approach also considers intermediate graphs. Therefore a notion of restricted graph morphisms is needed. For this, we introduce the notion of *restricted morphisms*, which intuitively is the restriction of the domain and co-domain of a morphism $p : G \hookrightarrow H$ with subgraphs of G and H respectively.

Definition 2.6 (**restriction of a morphism**). *Let the graphs G, H and a morphism $f : G \to H$ be given. Then, a morphism $f' : G' \to H'$ is called a* restriction *of p if inclusions $i : G' \hookrightarrow G$ and $i' : H' \hookrightarrow H$ exist such that*

$$i'_E \circ f'_E = f_E \circ i_E \text{ and}$$
$$i'_V \circ f'_V = f_V \circ i_V.$$

A restriction of p is denoted by p^r.

Note that given a morphism $p : G \to H$ a restriction $p^r : G' \to H'$ of p is uniquely determined by G' and H'. Assume two restrictions $p^r : G' \to H'$ and $q^r : G' \to H'$ of p are given. It holds that $i' \circ p^r = p \circ i = i' \circ q^r$ and $p^r = q^r$ follows because i' is injective.

2.2 Nested Graph Conditions and Constraints

Nested graph constraints are useful for specifying graph properties. The more general notion of *nested graph conditions* allows the specification of properties for graph morphisms and the definition of graph conditions and constraints in a recursive manner. Within these conditions, only quantifiers and Boolean operators are used [10].

Definition 2.7 (**nested graph condition**). *A* nested graph condition *over a graph C_0 is defined recursively as*

1. true *is a graph condition over every graph.*
2. $\exists(a_0 : C_0 \hookrightarrow C_1, d)$ *is a graph condition over C_0 if a_0 is a inclusion and d is a graph condition over C_1.*
3. $\neg d$, $d_1 \wedge d_2$ *and* $d_1 \vee d_2$ *are graph conditions over C_0 if d, d_1 and d_2 are graph conditions over C_0.*

Conditions over the empty graph $\emptyset$ are called constraints. *We use the abbreviations* $\forall(a_0 : C_0 \hookrightarrow C_1, d) := \neg\exists(a_0 : C_0 \hookrightarrow C_1, \neg d)$ *and* $\mathsf{false} = \neg\,\mathsf{true}$.

Conditions of the form $\exists(a_0 : C_0 \hookrightarrow C_1, d)$ are called existential *and the graph C_1 is called* existentially bound. *Conditions of the form $\forall(a_0 : C_0 \hookrightarrow C_1, d)$ are called* universal *and the graph C_1 is called* universally bound.

Since these are the only types of conditions that will be used in this paper, we will refer to them only as *conditions* and *constraints*. We will use the more compact notations $\exists(C_1, d)$ for $\exists(a_0 : C_0 \hookrightarrow C_1, d)$ and $\forall(C_1, d)$ for $\forall(a_0 : C_0 \hookrightarrow C_1, d)$ if C_0 and a_0 are clear from the context.

Example 2.1. *Given a condition* $c = \forall(a_0 : \emptyset \hookrightarrow C_1, \exists(a_2 : C_1 \hookrightarrow C_2, \mathsf{true}))$. *The compact notation of this condition is given by* $\forall(C_1, \exists(C_2, \mathsf{true}))$.

Definition 2.8 (**semantic of graph conditions**)**.** *Given a graph* G*, a condition* c *over* C_0 *and a graph morphism* $p : C_0 \hookrightarrow G$*. Then* p *satisfies* c*, denoted by* $p \models c$*, if*

1. $c = \mathsf{true}$.
2. $c = \exists(a_0 : C_0 \hookrightarrow C_1, d)$ *and there exists an injective morphism* $q : C_1 \hookrightarrow G$ *with* $p = q \circ a_0$ *and* $q \models d$.
3. $c = \neg d$ *and* $p \not\models d$.
4. $c = d_1 \wedge d_2$ *and* $p \models d_1$ *and* $p \models d_2$.
5. $c = d_1 \vee d_2$ *and* $p \models d_1$ *or* $p \models d_2$.

A graph G *satisfies a constraint* c*, denoted by* $G \models c$*, if the morphism* $p : \emptyset \hookrightarrow G$ *satisfies* c.

Our approach is designed to repair a specific type of constraint, constraints without any boolean operators. Each of these conditions can be transformed into an equivalent condition in so-called *alternating quantifier normal form* [18]. As the name suggests, these are conditions with alternating quantifiers and without any Boolean operators.

Definition 2.9 (**alternating quantifier normal form (ANF)**)**.** *Conditions in* alternating quantifier normal form *(ANF) are defined recursively as*

1. true *and* false *are conditions in ANF.*
2. $\exists(a_0 : C_0 \hookrightarrow C_1, d)$ *is a condition in ANF if either* d *is an universally bound condition over* C_1 *in ANF or* $d = \mathsf{true}$.
3. $\forall(a_0 : C_0 \hookrightarrow C_1, d)$ *is a condition in ANF if either* d *is an existentially bound condition over* C_1 *in ANF or* $d = \mathsf{false}$.

Every condition is a subcondition *of itself. In cases 2 and 3,* d *is called a* subcondition *of* $\exists(a : C_0 \hookrightarrow C_1, d)$ *or* $\forall(a : C_0 \hookrightarrow C_1, d)$ *respectively. All subcondition of* d *are also subconditions of* $\exists(a : C_0 \hookrightarrow C_1, d)$ *or* $\forall(a : C_0 \hookrightarrow C_1, d)$ *respectively. The* nesting level $\mathrm{nl}(c)$ *of a condition* c *is recursively defined as* $\mathrm{nl}(\mathsf{true}) = \mathrm{nl}(\mathsf{false}) = 0$ *and* $\mathrm{nl}(\exists(a : P \hookrightarrow Q, d)) = \mathrm{nl}(\forall(a : P \hookrightarrow Q, d)) := \mathrm{nl}(d) + 1$.

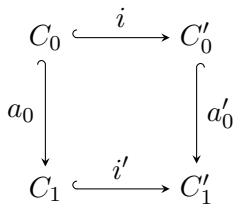

Figure 1: Diagram for the Shift operator.

In the literature, conditions in ANF also allow conditions that end with conditions of the form $\exists(C_1, \mathsf{false})$ or $\forall(C_1, \mathsf{true})$. We exclude these cases so that conditions in ANF can only end with conditions of the form $\exists(C_1, \mathsf{true})$ or $\forall(C_1, \mathsf{false})$, since it is easily seen that every morphism $p : C_0 \hookrightarrow G$ satisfies $\forall(C_1, \mathsf{true})$ and does not satisfy $\exists(C_1, \mathsf{false})$. Therefore, these conditions can be replaced by true and false respectively.

In the following, we assume that all graphs and the nesting level of a condition are finite. c

Using the *shift over morphism* construction, we are able to transform a nested condition over C into a nested condition over C' via an injective morphism $i : C \hookrightarrow C'$ [10].

Definition 2.10 (shift over morphism). *Let a condition c over C_0 and a morphism $i : C_0 \hookrightarrow C_0'$ be given. The* shift of c over i, *denoted by* $\mathrm{Shift}(c, i)$, *is given by*

1. *If $c = \mathsf{true}$,* $\mathrm{Shift}(c, i) = \mathsf{true}$.
2. *If $c = \exists(a_1 : C_0 \hookrightarrow C_1, d)$,* $\mathrm{Shift}(c, i) = \bigvee_{(a', i') \in \mathcal{F}} \exists(a', \mathrm{Shift}(d, i'))$ *with $\mathcal{F}$ being the set of all pairs (a', i') of injective morphisms that are jointly surjective and $i' \circ a = a' \circ i$, i.e., the diagram shown in Figure 1 commutates.*
3. *If $c = \neg d$,* $\mathrm{Shift}(c, i) = \neg\,\mathrm{Shift}(d, i)$
4. *If $c = d_1 \wedge d_2$,* $\mathrm{Shift}(c, i) = \mathrm{Shift}(d_1, i) \wedge \mathrm{Shift}(d_2, i)$
5. *If $c = d_1 \vee d_2$,* $\mathrm{Shift}(c, i) = \mathrm{Shift}(d_1, i) \vee \mathrm{Shift}(d_2, i)$

Lemma 2.11 ([10])**.** *Let a condition c over C_0 and a morphism $i : C_0 \hookrightarrow C_0'$ be given. Then, for each morphism $m : C_0' \hookrightarrow G$,*

$$m \models \mathrm{Shift}(c, i) \iff m \circ i \models c$$

2.3 Rules and Graph Transformations

Via *rules* and *graph transformation* graphs can be modified by inserting or deleting nodes and edges. We will use the concept of the double-pushout approach for rules and transformations, which is based on category theory [7]. A rule consists of the three graphs

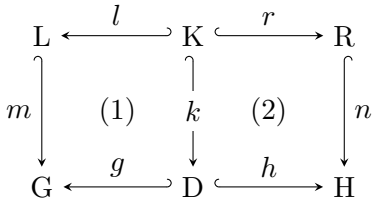

Figure 2: Diagram of a transformation in the double-pushout approach.

L, called the *left-hand side*, K, called *context*, and R, called *right-hand side*, where K is a subgraph of L and R. During a transformation, denoted by $G \Longrightarrow H$, elements of $L \setminus K$ are removed and elements of $R \setminus K$ are inserted so that a new morphism $p : R \hookrightarrow H$ is created. In addition, the so-called *dangling edge condition* must be satisfied, which means that for every edge $e \in E_H$ there are vertices $u, v \in V_H$ such that $\mathrm{tar}(e) = u$ and $\mathrm{src}(e) = v$ or vice versa. We also define application conditions. These are nested conditions over L and R that prevent the transformation if they are not satisfied. Later, we will use application conditions to ensure that transformations cannot reduce consistency. For example, application conditions that prevent a transformation if $G \models c$ and $H \not\models c$.

Definition 2.12 (**rules and application conditions**). *A* plain rule $\rho = L \xhookleftarrow{l} K \xhookrightarrow{r} R$ *consists of graphs* L, K, R *and inclusions* $l : K \hookrightarrow L$ *and* $r : K \hookrightarrow R$. *The rule* $\rho^{-1} = R \xhookleftarrow{r} K \xhookrightarrow{l} L$ *is called the* inverse rule of ρ'.

An application condition *is a nested condition over* L *or* R *respectively. A* rule $(\mathrm{ap}_L, \rho, \mathrm{ap}_R)$ *consists of a plain rule* ρ *and application conditions* ap_L *over* L*, called* left application condition*, and* ap_R *over* R*, called* right application condition *respectively.*

Definition 2.13 (**graph transformation**). *Let a rule* $\rho = (\mathrm{ap}_L, \rho', \mathrm{ap}_R)$*, a graph* G *and a morphism* $m : L \hookrightarrow G$*, called the* match*, be given. Then, a* graph transformation $t : G \Longrightarrow_{\rho,m} H$ *is given in Figure 2 if the squares* (1) *and* (2) *are pushouts in the sense of category theory,* $m \models \mathrm{ap}_L$ *and the morphism* $n : L \hookrightarrow H$*, called the co-match of* t*, satisfies* ap_R*. The morphisms* $g : D \hookrightarrow G$ *and* $h : D \hookrightarrow H$ *are called the* transformations morphisms *of* t.

The presence of right applications conditions leads to unpleasant side effects. The satisfaction of a right application condition can only be checked after the transformation. The transformation must therefore be reversed if the co-match does not satisfy this condition. To avoid this, we introduce the *shift over rule* operation, which is capable of transforming a right into an equivalent left application condition [10].

Definition 2.14 (**shift over rule**). *Let a plain rule* $\rho = L \xhookleftarrow{l} K \xhookrightarrow{r} R$ *and a right application condition* ap *of* ρ *be given. The* shift of ap over ρ*, denoted with* $\mathrm{Left}(\mathrm{ap}, \rho)$*, is defined as*

1. *If* $\mathrm{ap} = \mathit{true}$*,* $\mathrm{Left}(\mathrm{ap}, \rho) := \mathit{true}$.

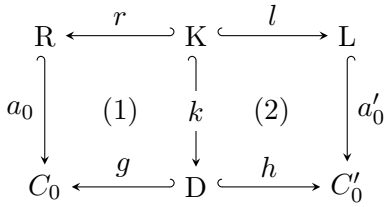

Figure 3: Transformation for the shift over rule operator.

2. *If* $\mathrm{ap} = \neg d$, $\mathrm{Left}(\mathrm{ap}, \rho) := \neg\,\mathrm{Left}(d, \rho)$.
3. *If* $\mathrm{ap} = d_1 \wedge d_2$, $\mathrm{Left}(\mathrm{ap}, \rho) := \mathrm{Left}(d_1, \rho) \wedge \mathrm{Left}(d_2, \rho)$.
4. *If* $\mathrm{ap} = d_1 \vee d_2$, $\mathrm{Left}(\mathrm{ap}, \rho) := \mathrm{Left}(d_1, \rho) \vee \mathrm{Left}(d_2, \rho)$.
5. *If* $\mathrm{ap} = \exists(a_0 : R \hookrightarrow C_1, d)$, $\mathrm{Left}(\mathrm{ap}, \rho) := \exists(a_0' : L \hookrightarrow C_0', \mathrm{Left}(d, \rho'))$ *where* $\rho' = C_0 \overset{g}{\hookleftarrow} D \overset{h}{\hookrightarrow} C_0'$ *is the rule derived in Figure 3 by applying* ρ^{-1} *at match* a_0. *If this transformation does not exist, we set* $\mathrm{Left}(\mathrm{ap}, \rho) :=$ *false*.

Shift over rule produces an equivalent left application condition, meaning that, given a right application condition ap and a plain rule ρ , a match of a transformation satisfies $\mathrm{Left}(\mathrm{ap}, \rho)$ if and only if the co-match satisfies ap [10].

Lemma 2.15. *Let a plain rule* $\rho = L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R$, *a right application condition* ap *for* ρ *and a transformation* $t : G \Longrightarrow_{\rho,m} H$ *be given. Then,*

$$m \models \mathrm{Left}(\mathrm{ap}, \rho) \iff n \models \mathrm{ap}\,.$$

Since every right application condition can be transformed into an equivalent left application condition, we will assume from now on that each rule contains only left application conditions. These rules are denoted by (ap, ρ).

Via the *track morphism* it is possible to track elements across a transformation [17].

Definition 2.16 (track morphism). *Consider the transformation* t *shown in figure 2. The* track morphism, $\mathrm{tr}_t : G \dashrightarrow H$, *of* t *is a partial morphism, defined as*

$$\mathrm{tr}_t = \begin{cases} h(g^{-1}(e)) & \text{if } e \in g(D) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

For example, given a transformation $t : G \Longrightarrow H$, the track morphism can be used to check whether a morphism $p : C \hookrightarrow G$ extends to the derived graph H by checking whether $\mathrm{tr}_t \circ p$ is total, or whether a new morphism $q : C \hookrightarrow H$ has been inserted by checking that no morphism $p : C \hookrightarrow H$ with $q = \mathrm{tr}_t \circ p$ exists [12]. We will use these results later on.

Lemma 2.17 ([12])**.** *Let a transformation* $t : G \Longrightarrow G$ *with transformation morphisms* $g : D \hookrightarrow G$, $h : D \hookrightarrow H$ *and an occurrence* $p : C \hookrightarrow G$ *of a graph* C *be given. Then,*

1. *The track morphism* tr_t *of* t *is total when restricted to* $p(C)$*, i.e.* $\mathrm{tr}_t \circ p$ *is total, if and only if there is an injective morphism* $p' : C \hookrightarrow D$ *such that* $p = g \circ p'$*.*

2. *Given an injective morphism* $p : C \hookrightarrow H$*,* $p(C)$ *is contained in* $\mathrm{tr}_t(G)$ *if and only if there is an injective morphism* $p' : C \hookrightarrow D$ *such that* $p = h \circ p'$*.*

Lemma 2.18 ([12])**.** *Given a transformation* $t : G \Longrightarrow H$ *with the transformation morphisms* $g : D \hookrightarrow G$*,* $h : D \hookrightarrow H$ *and a constraint* c *in ANF that contains the morphism* $a_i : C_{i-1} \hookrightarrow C_i$*. Given injective morphisms* $p_{i-1} : C_{i-1} \hookrightarrow G$ *and* $p_i : C_i \hookrightarrow G$ *such that* $p_{i-1} ? p_i \circ a_i$ *and the track morphism* $\mathrm{tr}_t : G \dashrightarrow H$ *is total when restricted to* $p_i(C_i)$*. Then,* $p'_{i-1} = p'_i \circ a_i$ *where* $p'_{i-1} = \mathrm{tr}_t \circ p_{i-1}$ *and*$p'_i = \mathrm{tr}_t \circ p_i$*. Also, given injective morphisms* $p'_{i-1} : C_{i-1} \hookrightarrow H$ *and* $p'_i : C_i \hookrightarrow H$ *such that* $p'_{i-1} = p'_i \circ a_i$ *and* $p_i(C_i)$ *is contained in* $\mathrm{tr}_t(G)$*, then* $p_{i-1} = p_i \circ a_i$ *where* $p_{i-1} = \mathrm{tr}_t^{-1} \circ p'_{i-1}$ *and* $p_i = \mathrm{tr}_t^{-1} \circ p'_i$*.*

Definition 2.19 (**new and removed occurrences**)**.** *Given a transformation* $t : G \Longrightarrow H$ *and a graph* C*. An occurrence* $p : C \hookrightarrow G$ *is called a* removed occurrence of C by t *if* $\mathrm{tr}_t \circ p$ *is not total. An occurrence* $p' : C \hookrightarrow H$ *is called a* inserted occurrence of C by t *if there is no morphism* $p : C \hookrightarrow G$ *such that* $p' = \mathrm{tr}_t \circ p$*.*

Given a sequence of transformations, the notion of *concurrent rules* can be used to describe this sequence by a rule. In other words, any sequence of transformations can be replaced by a transformation via its concurrent rule [8].

Definition 2.20 (**concurrent rule**)**.** *Let the rules* $\rho_1 = L_1 \overset{l_1}{\hookleftarrow} K_1 \overset{r_1}{\hookrightarrow} R_1$*,* $\rho_2 = L_2 \overset{l_2}{\hookleftarrow} K_2 \overset{r_2}{\hookrightarrow} R_2$ *and a sequence of transformations*

$$G_1 \Longrightarrow_{\rho_1, m_1} G_2 \Longrightarrow_{\rho_2, m_2} G_3$$

be given. Then, $\rho' = G_1 \overset{l'}{\hookleftarrow} K \overset{r'}{\hookrightarrow} G_3$ *is called the* concurrent rule of the transformation sequence *if the square* (5) *in Figure 4 is a pullback.*

A transformation sequence $G_1 \Longrightarrow_{\rho_1, m_1} G_2 \Longrightarrow_{\rho_2, m_2} G_3$ can be replaced by a transformation $G_1 \Longrightarrow_{\rho', \mathrm{id}} G_3$ via its concurrent rule. By inductive application, a concurrent rule for a transformation sequence $G_1 \Longrightarrow_{\rho_0} \ldots \Longrightarrow_{\rho_n} G_n$ of arbitrary finite length can be derived.

2.4 Concepts of Consistency

Now, we will introduce familiar consistency concepts. Namely, the notions of consistency preserving and guaranteeing transformations [10] and the notions of (direct) consistency sustaining and improving transformations [12]. Later, we will examine how these concepts differ from and correlate to our newly introduced concept of consistency.

Definition 2.21 (**consistency preserving and guaranteeing transformations**)**.** *Let a constraint* c *and a transformation* $t : G \Longrightarrow H$ *be given. Then,* t *is called* c-preserving *if*

$$G \models c \implies H \models c.$$

The transformation t *is called* c-guaranteeing *if* $H \models c$*.*

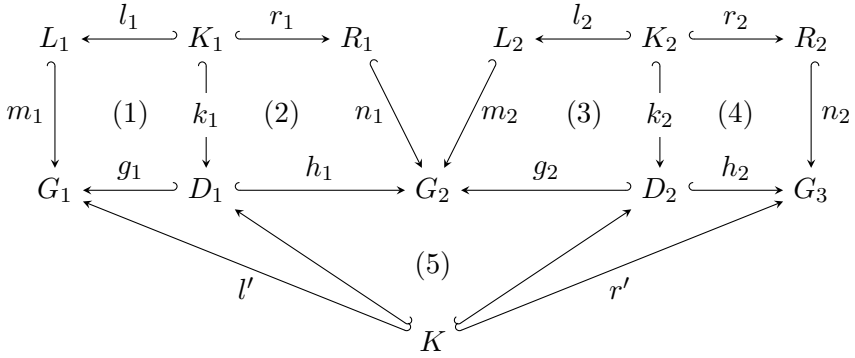

Figure 4: Pushout diagram of the transformation sequence $G_1 \Longrightarrow_{\rho_1,m_1} G_2 \Longrightarrow_{\rho_2,m_2} G_3$ using the rules $\rho_1 = L_1 \xleftarrow{l_1} K_1 \xhookrightarrow{r_1} R_1$ and $\rho_2 = L_2 \xleftarrow{l_2} K_2 \xhookrightarrow{r_2} R_2$.

While consistency preserving and guaranteeing transformations are defined for nested conditions, the finer-grained notions of (direct) consistency sustaining and improving transformations are defined only for conditions in ANF.

Definition 2.22 (**consistency sustaining and improving transformations**). *Let a constraint c in ANF and a transformation $t : G \Longrightarrow_\rho H$ be given. If c is existentially bound, t is called* consistency sustaining w.r.t. *c if it is c-preserving and t is called* consistency improving w.r.t. *c if it is c-guaranteeing. If $c = \forall(a_0 : \emptyset \hookrightarrow C_1, d)$ is universally bound, t is called* consistency sustaining w.r.t. *c if*

$$|\{p : C_1 \hookrightarrow G \mid p \not\models d\}| \geq |\{p : C_1 \hookrightarrow H \mid p \not\models d\}|$$

and t is called consistency improving w.r.t. *c if*

$$|\{p : C_1 \hookrightarrow G \mid p \not\models d\}| > |\{p : C_1 \hookrightarrow H \mid p \not\models d\}|.$$

The number of elements of these sets are called the number of violations in *G and* number of violations in *H respectively.*

The even stricter notion of direct sustaining and improving transformations prohibits the insertion of new violations altogether.

Definition 2.23 (**direct sustaining and improving transformations**). *Let a constraint c in ANF and a transformation $t : G \Longrightarrow_\rho H$ be given. If c is existentially bound, t is called* direct consistency sustaining w.r.t. *c if t is c-preserving and t is called* direct consistency improving w.r.t. *c if t is c-guaranteeing.*

If $c = \forall(a_0 : \emptyset \hookrightarrow C_1, d)$, t is called consistency sustaining w.r.t. *c if*

$$\forall p : C_0 \hookrightarrow G((p \models d \wedge \mathrm{tr}_t \circ p \text{ is total}) \implies \mathrm{tr}_t \circ p \models d) \text{ and}$$
$$\forall p' : C_0 \hookrightarrow H(\neg\exists q : C_0 \hookrightarrow G(p' = \mathrm{tr}_t \circ q) \implies p' \models d))$$

and t is called consistency improving w.r.t. *c if additionally*

$$\exists p : C_0 \in G(p \not\models d \text{ and} \mathrm{tr}_t \circ p \text{ is total} \wedge \mathrm{tr}_t \circ p \models d) \vee$$
$$\exists p : C \hookrightarrow G(p \not\models d \wedge \mathrm{tr}_t \circ p \text{ is not total}).$$

3 Consistency Increase and Maintainment

In the following we will introduce the notions of *Satisfaction until after layer*, *(direct) consistency-maintaining* and *(direct) consistency-increasing* transformations and rules and compare them with the notions of consistency introduced in the previous section.

Definition 3.1 (**layer of a subcondition**). *Let a condition c in ANF and a subcondition d of c be given. The* layer of d *is defined as* $\mathrm{lay}(d) := \mathrm{nl}(c) - \mathrm{nl}(d)$.

Our approach is based on the idea that the consistency of a constraint increases layer by layer, and that even small improvements, such as inserting single elements of existentially bound graphs, should be detectable as increasing. To formalise this, we introduce the notions of *consistency increasing* and *consistency maintaining* transformations and rules, where consistency increasing indicates that the consistency has actually increased and consistency maintaining indicates that the consistency has not decreased.

3.1 Universally quantified ANF

The definition of consistency increase and maintainment requires that each condition begins with a universal quantifier. Otherwise, case discrimination is required. Therefore, we will only consider a subset of the set of conditions in ANF, namely the set of universally quantified conditions in ANF, called *universally quantified ANF* (UANF). Furthermore, we will show that these sets are expressively equivalent by showing that every condition in ANF can be transformed into an equivalent condition in UANF.

Definition 3.2 (**universally quantified alternating quantifier normal form**). *A conditions c in ANF is in* universally quantified ANF *(UANF) if it is universally bound.*

Note that in our notation, given a condition c in UANF, any subcondition of c at layer $0 \leq k \leq \mathrm{nl}(c)$ is universal if k is an even number and existential if k is an odd number. Furthermore, a graph C_k of c is universally bound if k is an odd number and existentially bound if k is an even number. It is already known that an existentially bound condition c can be extended to the equivalent condition $\exists(\mathrm{id}_{C_0} : C_0 \hookrightarrow C_0, d)$ [9]. Analogously, we show that every condition in ANF has an equivalent condition in UANF.

Lemma 3.3. *Any condition in ANF can be transformed into an equivalent condition in UANF.*

Proof. Let a graph G and a constraint c in ANF be given. If c is universally bound, then c is already in UANF. If $c = \exists(a_0 : C_0 \hookrightarrow C_1, d)$, we show that c is equivalent to $c' := \forall(\mathrm{id}_{C_0} : C_0 \hookrightarrow C_0, c)$.

1. Let $p : C_0 \hookrightarrow G$ be a morphism such that $p \models c$. Then $p \models c'$, because p is the only morphism from C_0 to G with $p = p \circ \mathrm{id}_{C_0}$ and $p \models c$.

2. Let $p : C_0 \hookrightarrow G$ be a morphism with $p \models c'$, then all morphisms $q : C_0 \hookrightarrow G$ with $p = q \circ \mathrm{id}_{C_0}$ satisfy c. Since $p = p \circ \mathrm{id}_{C_0}$, it immediately follows that $p \models c$. □

$$c_1 = \forall(C_1^1, \exists(C_2^1, \mathsf{true}))$$
$$c_2 = \forall(C_1^1, \exists(C_2^2, \forall(C_3^2, \exists(C_4^2, \mathsf{true}))))$$

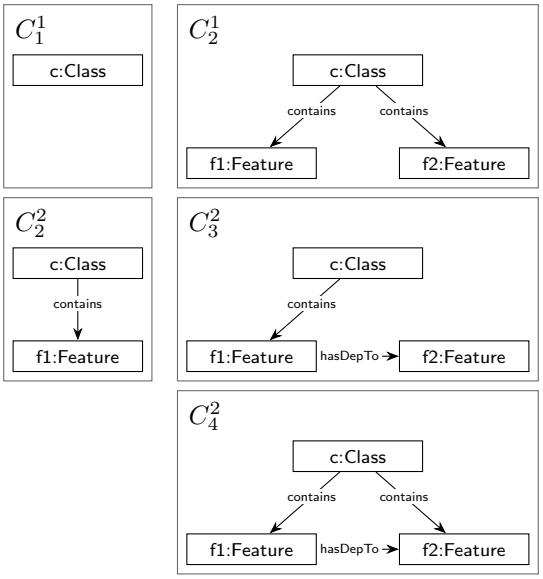

Figure 5: constraints

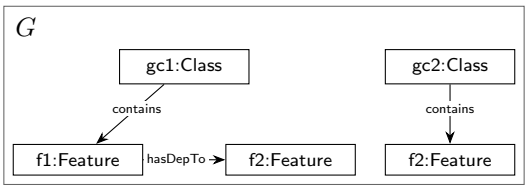

Figure 6: graph

For the rest of this thesis, given a condition $c = \forall(a_0 : C_0 \hookrightarrow C_1, d)$ in UANF, we assume that no morphism in c, except a_0, is bijective, since it can be shown that every condition in ANF can be transformed into an equivalent condition in ANF that satisfies this property, by showing that $\exists(\mathrm{id}_{C_0} : C_0 \hookrightarrow C_0, \forall(a_1 : C_0 \hookrightarrow C_2, d))$ is equivalent to $\forall(a_1 : C_0 \hookrightarrow C_2, d)$ and that $\forall(\mathrm{id}_{C_0} : C_0 \hookrightarrow C_0, \exists(a_1 : C_0 \hookrightarrow C_2, d))$ is equivalent to $\exists(a_1 \circ a_0 : C_0 \hookrightarrow C_2, d)$ [9]. In addition, given a condition c in UANF, we will denote the first graph of c with C_0, the first morphism with a_0, the second graph with C_1, the second morphism with a_1, and so on. This means that we always write constraints as $\forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1 : C_1 \hookrightarrow C_2, \ldots))$.

3.2 Satisfaction until after Layer

The goal of our approach is to increase the consistency of a constraint layer by layer, as we have already mentioned. To do this, we introduce a notion of partial consistency,

called *Satisfaction until after layer*, which allows us to check whether a constraint is satisfied at a particular layer by checking whether the so-called *truncated condition after layer* is satisfied at that layer.

We first define the *subcondition at layer* $-1 \leq k \leq \mathrm{nl}(c)$ of a condition c. As the name suggests, the subcondition at layer $0 \leq k \leq \mathrm{nl}(c)$ denotes the subcondition of c with layer k. We also define the subcondition at layer -1, which is true. This will be useful for defining the satisfaction at layer when a graph does not satisfy any layer of a constraint.

Definition 3.4 (**subcondition at layer**). *Let a condition c in ANF be given. The* subcondition at layer $-1 \leq k \leq \mathrm{nl}(c)$*, denoted by* $\mathrm{sub}_k(c)$*, is the subcondition d of c with* $\mathrm{lay}(d) = k$ *if* $0 \leq k \leq \mathrm{nl}(c)$ *and* true *if* $k = -1$.

Note that by definition the subcondition at layer k is always a condition over the graph C_k and the morphism is denoted by a_k.

Example 3.1. *Consider the condition* $c = \forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1 : C_1 \hookrightarrow C_2, \forall(a_2 : C_2 \hookrightarrow C_3, \mathsf{false})))$*. Then,* $\mathrm{sub}_1(c) = \exists(a_1 : C_1 \hookrightarrow C_2, \forall(a_2 : C_2 \hookrightarrow C_3, \mathsf{false}))$.

1. $\mathrm{sub}_{-1}(c) = \mathsf{true}$.
2. $\mathrm{sub}_0(c) = \forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1 : C_1 \hookrightarrow C_2, \forall(a_2 : C_2 \hookrightarrow C_3, \mathsf{false}))) = c$.
3. $\mathrm{sub}_1(c) = \exists(a_1 : C_1 \hookrightarrow C_2, \forall(a_2 : C_2 \hookrightarrow C_3, \mathsf{false}))$.
4. $\mathrm{sub}_2(c) = \forall(a_2 : C_2 \hookrightarrow C_3, \mathsf{false})$.
5. $\mathrm{sub}_3(c) = \mathsf{false}$.

Let us first introduce an operator which allows to replace a subcondition $\mathrm{sub}_k(c)$ by an arbitrary condition over C_k, called *replacement starting from layer*.

Definition 3.5 (**replacement starting from layer**). *Given a condition $c = Q(a_0 : C_0 \hookrightarrow C_1, d)$, with $Q \in \{\forall, \exists\}$, in ANF, and a condition e over C_k in ANF. The* replacement starting from layer k in c by e*, denoted by* $\mathrm{rep}_k(c, e)$*, is defined recursively as*

$$\mathrm{rep}_k(c, e) := \begin{cases} e & \text{if } k = 0 \\ Q(a_0 : C_0 \hookrightarrow C_1, \mathrm{rep}_{k-1}(d, e)) & \text{otherwise.} \end{cases}$$

Example 3.2. *Consider the conditions* $c := \forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1 : C_1 \hookrightarrow C_2, \mathsf{true}))$ *and* $e = \exists(a_1' : C_1 \hookrightarrow C_3, d)$*. The replacement of layer 1 in c by e is given by*

$$\mathrm{rep}_1(c, e) = \forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1' : C_1 \hookrightarrow C_3, d)).$$

We now define *truncated conditions after layer* using the concept of replacement starting from layer. Intuitively, a condition is truncated after a particular layer by replacing the subcondition at the next layer with true or false, depending on which quantifier the replaced subcondition is bound by.

Definition 3.6 (**truncated condition after layer**). *Let a condition c in UANF be given. The* truncated condition of c after layer $-1 \leq k < \mathrm{nl}(c)$, *denoted by* $\mathrm{cut}_k(c)$, *is defined as*

$$\mathrm{cut}_k(c) := \begin{cases} \mathit{true} & \text{if } k = -1 \\ \mathrm{rep}_{k+1}(c, \mathit{true}) & \text{if } \mathrm{sub}_k(c) \text{ is a existential condition, i.e. } k \text{ is odd} \\ \mathrm{rep}_{k+1}(c, \mathit{false}) & \text{if } \mathrm{sub}_k(c) \text{ is a universal condition, i.e. } k \text{ is even.} \end{cases}$$

Example 3.3. *Consider constraint c_2 given in Figure 5. The truncated conditions of c_2 after layer $-1 \leq k < 3$ are given by*

1. $\mathrm{cut}_{-1}(c_2) = \mathit{true}$.
2. $\mathrm{cut}_0(c_2) = \forall(C_1^1, \mathit{false})$.
3. $\mathrm{cut}_1(c_2) = \forall(C_1^1, \exists(C_2^2, \mathit{true}))$.
4. $\mathrm{cut}_2(c_2) = \forall(C_1^1, \exists(C_2^2, (\forall C_3^2, \mathit{false})))$.
5. $\mathrm{cut}_3(c_2) = \forall(C_1^1, \exists(C_2^2, (\forall C_3^2, \exists(C_4^2, \mathit{true})))) = c_2$.

Note that the truncated condition of a condition c at layer $\mathrm{nl}(c) - 1$ is c itself. With these prerequisites we can now introduce *satisfaction until after layer*, which allows us to check whether a condition is satisfied up to a given layer. A morphism or graph satisfies a condition or constraint until after a layer if it satisfies the truncated condition after that layer.

Definition 3.7 (**satisfaction until after layer**). *Let a graph G and a condition c in UANF be given. A morphism $p : C_0 \hookrightarrow G$* satisfies c until after layer $-1 \leq k < \mathrm{nl}(c)$, *denoted by $p \models_k c$, if*

$$p \models \mathrm{cut}_k(c).$$

A graph G satisfies a constraint c until after layer $-1 \leq k < \mathrm{nl}(c)$, denoted by $G \models_k c$, if $q : \emptyset \hookrightarrow G$ satisfies $\mathrm{cut}_k(c)$. The largest $-1 \leq k < \mathrm{nl}(c)$ such that $G \models_k c$ and there is no $k < j < \mathrm{nl}(c)$ with $G \models_j c$, called the largest satisfied layer, *is denoted by $\mathrm{k}_{\max}(c, G)$. When c and G are clear from the context, we use the abbreviation $\mathrm{k}_{\max}$.*

Note that given a graph G and a constraint c, $\mathrm{k}_{\max}(c, G)$ always exists, since $\mathrm{cut}_{-1}(c) =$ `true` and every graph satisfies `true`. Moreover, if $p \models_{\mathrm{nl}(c)-1} c$, i.e. $\mathrm{k}_{\max} = \mathrm{nl}(c) - 1$, it immediately follows that $p \models c$.

Example 3.4. *Consider the graph G given in Figure 6 and the constraint c_2 given in Figure 5. This graph does not satisfy c_2 because the occurrence of* `Class` *denoted with* `gc2` *does not satisfy $\mathrm{sub}_1(c_2) = \exists(C_2^2, (\forall C_3^2, (\exists C_4^2, \mathit{true})))$, but it satisfies $\mathrm{cut}_1(c_2) = \forall(C_1^1, \exists(C_2^2, \mathit{true}))$ and therefore*

$$G \models_1 c_2 \text{ and } \mathrm{k}_{\max} = 1.$$

$p \models_k c$	$p \models_{j<k} c$		$p \models_{j>k} c$		$p \models c$
	j even	j odd	j even	j odd	
k even	?	✓	✓	✓	✓
k odd	?	✓	?	?	?

Table 1: Overview of the inferences made about satisfaction at layer, with "✓" indicating that $p \models_j c$ and $p \models c$, respectively, if $p \models_k c$, and "?" indicating that it cannot be inferred from $p \models_k c$ whether $p \models_j c$ or $p \not\models_j c$.

Given a graph G, a condition c and a morphism $p : C_0 \hookrightarrow G$. Suppose that $p \models_k c$ with $0 \leq k < \mathrm{nl}(c)$. Then we can infer results for the satisfaction until after other layers. If k is even, i.e. $\mathrm{sub}_k(c)$ is a universal condition, we can conclude that $p \models_j c$ for all $k < j < \mathrm{nl}(c)$ and especially $p \models c$. It also follows that $p \models_j c$ for all odd $0 \leq j < k$, i.e. $\mathrm{sub}_j(c)$ is an existential condition. We present these results in the following lemmas, and an overview is given in Table 1.

We start by examining the consequences for the satisfaction until after layer $\mathrm{nl}(c) > j > k$ if $p \models_k c$. Our first lemma shows that replacing the subcondition $\mathrm{sub}_{k+1}(c)$ by any condition over C_{k+1} leads to a condition that is satisfied by p if k is even.

Lemma 3.8. *Given a graph G, a condition c in UANF and a morphism $p : C_0 \hookrightarrow G$ with $p \models_k c$ and $-1 \leq k < \mathrm{nl}(c)$ even. Then, for any condition f over C_{k+1} it holds that*

$$p \models \mathrm{rep}_{k+1}(c, f).$$

Proof. We start by showing the statement for the smallest $-1 \leq j < \mathrm{nl}(c)$ such that $\mathrm{sub}_j(c)$ is universally bound and $p \models_j c$. After this, we can conclude that this statement holds for all $-1 \leq i < \mathrm{nl}(c)$ such that $\mathrm{sub}_i(c)$ is universally bound and $p \models_i c$.

Let $q : C_j \hookrightarrow G$ be a morphism such that $q \models \forall(a_j : C_j \hookrightarrow C_{j+1}, \mathsf{false})$. This morphism must exist, since j is the smallest even number with $p \models_j c$. Therefore, there does not exist a morphism $q' : C_{j+1} \hookrightarrow G$ with $q = q' \circ a_j$. Hence, for every condition f over C_{j+1} a morphism $q' : C_{j+1} \hookrightarrow G$ with $q \not\models f$ and $q = q' \circ a_j$ cannot exist. It follows immediately that $q \models \forall(a_j : C_j \hookrightarrow C_{j+1}, f)$ and with that $p \models \mathrm{rep}_{j+1}(c, f)$.

We can now conclude that for every even $j < k \leq \mathrm{nl}(c)$, such that $p \models_k c$, and every condition d over C_{k+1} it holds that $p \models \mathrm{rep}_{k+1}(c, d)$ because $\mathrm{rep}_{k+1}(c, d) = \mathrm{rep}_{j+1}(c, \mathrm{sub}_{j+1}(\mathrm{rep}_{k+1}(c, d)))$. □

As a direct consequence of the previous lemma, a morphism which satisfies the condition at layer k, where k is even, also satisfies the condition at layer j for all $j > k$.

Lemma 3.9. *Given a graph G, a morphism $p : C_0 \hookrightarrow G$ and a condition c in UANF. If $0 \leq k < \mathrm{nl}(c)$ is even, i.e. $\mathrm{sub}_k(c)$ is a universal condition, then for all $k < j < \mathrm{nl}(c)$ it holds that*

$$p \models_k c \implies p \models_j c.$$

Proof. Follows immediately by using Lemma 3.8 and setting f equal to $\mathrm{sub}_{k+1}(\mathrm{cut}_j(c))$. □

$p \models_k c$	$\mathrm{k_{max}} < \mathrm{nl}(c) - 1$ $k \leq \mathrm{k_{max}}$	$\mathrm{k_{max}} = \mathrm{nl}(c) - 1$ $k < \mathrm{k_{max}}$
k even	✗	?
k odd	✓	✓

Table 2: Overview of the satisfaction until after layer k with respect to $\mathrm{k_{max}}$, where "✓" indicates that $p \models_k c$, "✗" indicates that $p \not\models_k c$ and "?" indicates that it cannot be concluded from $p \models_k c$ whether $p \models_j c$ or $p \not\models_j c$.

Since a morphism p satisfies a condition c in UANF if and only if p satisfies c at layer $\mathrm{nl}(c) - 1$, we can conclude the following.

Corollary 3.10. *Given a graph G, a morphism $p : C_0 \hookrightarrow G$ and a condition c in UANF. If $0 \leq k < \mathrm{nl}(c)$ is even, it holds that*

$$p \models_k c \implies p \models c.$$

In the following, we will mostly assume that $\mathrm{k_{max}}$ is odd. This is because an even $\mathrm{k_{max}}$ implies that the condition is already satisfied. Furthermore, this allows us to make statements about the satisfaction of other conditions. Given a graph G, a morphism $p : C_0 \hookrightarrow G$ and a condition c such that $p \models_k c$ for an even $-1 \leq k < \mathrm{nl}(c)$. It follows that $p \models c$ and in particular $p \models c'$ for each condition c' with $\mathrm{cut}_k(c) = \mathrm{cut}_k(c')$.

Let us now examine the satisfaction until after layer j with $-1 < j < k$. If j is odd, i.e. $\mathrm{sub}_j(c)$ is an existential condition, we can conclude that $p \models_j c$ as shown in the next lemma. If j is even, i.e. $\mathrm{sub}_j(c)$ is universally bound, we can only make statements that depend on $\mathrm{k_{max}}$. If $\mathrm{k_{max}} < \mathrm{nl}(c) - 1$, then $p \not\models_j c$. Otherwise $p \models c$ and therefore $\mathrm{k_{max}} = \mathrm{nl}(c) - 1$ would immediately follow Corollary from 3.10. If $\mathrm{k_{max}} = \mathrm{nl}(c) - 1$, we can say that there is at least one even $j \leq \mathrm{k_{max}}$ with $p \models_j c$ if c ends with $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$. An overview of these relations is given in Table 2.

Lemma 3.11. *Given a graph G, a morphism $p : C_0 \hookrightarrow G$ and a constraint c in UANF. Then for all odd $-1 \leq k \leq \mathrm{k_{max}}$, i.e. $\mathrm{sub}_k(c)$ is an existential condition, we have*

$$p \models_k c.$$

Proof. If there is an even $0 \leq j < \mathrm{k_{max}}$, i.e. $\mathrm{sub}_j(c)$ is universal, with $p \models_j c$, let j' be the smallest of these. Lemma 3.8 implies that $p \models_\ell c$ for all $j' \leq \ell < \mathrm{nl}(c)$. Otherwise we set $j' = \mathrm{k_{max}}$.

Let $\ell < j'$, such that $\mathrm{sub}_\ell(c)$ is a existential condition and let $d = \mathrm{sub}_\ell(\mathrm{cut}_{j'}(c)) = \exists(a_\ell : C_\ell \hookrightarrow C_{\ell+1}, e)$ be the subcondition at layer ℓ of the truncated condition after layer j' of c. Since $\ell < j'$, there must be a morphism $q : C_\ell \hookrightarrow G$ with $q \models d$ and therefore there must be a morphism $q' : C_{\ell+1} \hookrightarrow G$ with $q = q' \circ a_\ell$ and $q' \models e$. It follows that $q \models \exists(a_\ell : C_\ell \hookrightarrow C_{\ell+1}, \mathsf{true})$ and thus $p \models_\ell c$. □

Example 3.5. *We will show counterexamples for all "?" in Table 1 and Table 2. Consider constraint $c_2 = \forall(C_1^1, \exists(C_2^2, \forall(C_3^2, \exists(C_4^2, \mathsf{true}))))$ given in Figure 5. We begin with Table 1.*

1. *$j < k$, j and k are even. We set $k = 2$ and $j = 0$. It follows that* $\mathrm{cut}_k(c) = \forall(C_1^1, \exists(C_2^2, \forall(C_3^2, \mathsf{false})))$ *and* $\mathrm{cut}_j(c) = \forall(C_1^1, \mathsf{false})$. *Then, $C_2^2 \models_2 c_2$, $C_2^2 \not\models_0 c_2$ and $\emptyset \models_2 c_2$ and $\emptyset \models_0 c_2$.*

2. *$j < k$, k is odd and j is even. We set $k = 3$ and $j = 0$. It follows that* $\mathrm{cut}_k(c_2) = c_2$ *and* $\mathrm{cut}_j(c) = \forall(C_1^1, \mathsf{false})$. *Then, $C_4^2 \models_3 c_2$, $C_4^2 \not\models_0 c_2$ and $\emptyset \models_3 c_2$ and $\emptyset \models_0 c_2$.*

3. *$j > k$, k is odd and j is even. We set $k = 1$ and $j = 2$. It follows that* $\mathrm{cut}_k(c) = \forall(C_1^1, \exists(C_2^2, \mathsf{true}))$ *and* $\mathrm{cut}_j(c) = \forall(C_1^1, \exists(C_2^2, \forall(C_3^2, \mathsf{false})))$. *Then, $C_2^2 \models_1 c_2$, $C_2^2 \models_2 c_2$ and $C_3^2 \models_1 c_2$ and $C_3^2 \not\models_2 c_2$.*

4. *$j > k$, k and j are odd. We set $k = 1$ and $j = 3$. It follows that* $\mathrm{cut}_k(c) = \forall(C_1^1, \exists(C_2^2, \mathsf{true}))$ *and* $\mathrm{cut}_j(c) = c_2$. *Then, $C_2^2 \models_1 c_2$, $C_2^2 \models_3 c_2$ and $C_3^2 \models_1 c_2$ and $C_3^2 \not\models_3 c_2$. Since* $\mathrm{cut}_j(c) = c_2$ *this is also a counterexample for the "?" in the column "$p \models c$".*

For the "?" in Table 2 the graph C_2^2 satisfies c_2 and therefore $\mathrm{k_{max}} = 3 = \mathrm{nl}(c_2) - 1$. *But, $C_2^2 \models_2 c_2$ and $C_2^2 \not\models_0 c_2$.*

3.3 Consistency Increasing and Maintaining Transformations and Rules

Using satisfaction until after layer, an increase of consistency can be detected in the following way: Let $t : G \Longrightarrow H$ be a transformation. If the largest satisfied layer in H is greater than the largest satisfied layer in G, i.e. $\mathrm{k_{max}}(c, G) < \mathrm{k_{max}}(c, H)$, we consider the transformation as consistency increasing. However, the notion of consistency increasing should also be able to detect the smallest changes made by a transformation that lead to an increase of consistency, namely the insertion of a single edges or nodes of an existentially bound graph. To remedy this problem, we introduce *intermediate conditions*, which are used to detect this type of increase by checking whether an intermediate condition not satisfied by G is satisfied by H. Obviously, a decrease of consistency can be detected in a similar way, by checking whether an intermediate condition satisfied by G is not satisfied by H. Intuitively, the last graph of a truncated condition c will be replaced by an intermediate graph of the penultimate graph and the last graph of this truncated condition.

If c ends with an existentially bound condition, the constructed intermediate condition is weaker than c, in the sense that the satisfaction of c implies the satisfaction of the intermediate condition as shown by Lemma 3.13.

Conversely, if c ends with a universally bound condition, the opposite holds: satisfying an intermediate condition implies the satisfaction of c. This is why we have designed intermediate conditions so that they only replace graphs on existential layers.

Definition 3.12 (intermediate condition). *Let a condition c in UANF be given and let $0 \leq k < \mathrm{nl}(c)$ be odd, i.e. $\mathrm{sub}_k(c)$ is a existential condition. The* intermediate condition*, denoted by $\mathrm{IC}_k(c, C')$, of c at layer k with $C' \in \mathrm{IG}(C_k, C_{k+1})$ is defined as*

$$\mathrm{IC}_k(c, C') := \mathrm{rep}_k(c, \exists(a_k^r : C_k \hookrightarrow C', \mathsf{true})).$$

Lemma 3.13. *Given a condition c in UANF, a graph G, $0 \leq k < \mathrm{nl}(c)$ odd, i.e. $\mathrm{sub}_k(c)$ is a existential condition, and $C' \in \mathrm{IG}(C_k, C_{k+1})$. Then,*

$$G \models \mathrm{cut}_k(c) \implies G \models \mathrm{IC}_k(c, C').$$

Proof. Assume that $G \models \mathrm{cut}_k(c)$, i.e. $G \models_k c$. If there is an even $-1 \leq j < k$ such that $G \models_j c$, $G \models \mathrm{IC}_k(c, C')$ follows with Lemma 3.8. Otherwise, if there is no such j, for all morphism $p : C_k \hookrightarrow G$ such that there is a morphism $p' : C_{k+1} \hookrightarrow G$ with $p = p' \circ a_k$, there is also a morphism $q : C' \hookrightarrow G$ with $p = q \circ a_k^r$ where $a_k^r : C_k \hookrightarrow C$ is the restriction of a_k and q is restriction of p to the domain C. It follows that $G \models \mathrm{IC}_k(c, C')$. □

Example 3.6. *Consider the constraint c_1 given in Figure 5. Since $C_2^2 \in \mathrm{IG}(C_1^1, C_2^1)$, we can construct an intermediate condition of c_1 at layer 1 with C_2^2 as $\mathrm{IC}_1(c_1, C_2^2) = \forall(C_1^1 \exists(C_2^2, \mathsf{true}))$. While c_1 checks whether each node of type* `Class` *is connected to at least two nodes of type* `Feature`*, the intermediate condition checks whether each node of type* `Class` *is connected to at least one node of type* `Feature` *which is trivially satisfied if c_1 is satisfied.*

With the results above, we are now ready to define the notions of *consistency increasement* and *maintainment*, where increasement is a special case of maintainment. A transformation t is considered as consistency maintaining if it does not decrease consistency in the finer grained sense as described above, while t is considered as consistency increasing if it increases the consistency.

These notions are designed to detect only transformations that maintain (or increase) the consistency of the first two unsatisfied layers of a constraint c. That means, given a graph G and a constraint c, a transformation $t : G \Longrightarrow H$ is considered as consistency maintaining if the largest satisfied layer has not decreased, i.e. if $\mathrm{k_{max}}(c, G) \leq \mathrm{k_{max}}(c, H)$, and at least as many increasing insertions or deletions have been made as decreasing ones. An increasing deletion is the deletion of an occurrence of $C_{\mathrm{k_{max}}(c,G)+2}$ that does not satisfy $\exists(C_{\mathrm{k_{max}}(c,G)+3}, \mathsf{true})$, an increasing insertion is the insertion of elements, such that for at least one occurrence p of $C_{\mathrm{k_{max}}(c,G)+2}$ it holds that $p \not\models \exists(C', \mathsf{true})$ and $\mathrm{tr}_t \circ p \models \exists(C', \mathsf{true})$ for an intermediate graph $C' \in \mathrm{IG}(C_{\mathrm{k_{max}}(c,G)+2}, C_{\mathrm{k_{max}}(c,G)+3})$. Decreasing insertions and deletions are the opposite of increasing ones. That is, a decreasing insertion is the insertion of an occurrence of $C_{\mathrm{k_{max}}(c,G)+2}$ that does not satisfy $\exists(C_{\mathrm{k_{max}}(c,G)+3}, \mathsf{true})$ and a decreasing deletion is the deletion of elements such that for one occurrence p of $C_{\mathrm{k_{max}}(c,G)+2}$ with $p \models \exists(C', \mathsf{true})$ it holds that $\mathrm{tr}_t \circ p \not\models \exists(C', \mathsf{true})$ for an intermediate graph $C' \mathrm{IG}(C_{\mathrm{k_{max}}(c,G)+2}, C_{\mathrm{k_{max}}(c,G)+3})$. If $\mathrm{k_{max}}(c, G) < \mathrm{k_{max}}(c, H)$ or the number of increasing insertions and deletions is greater than the number of decreasing ones, t is considered as consistency increasing.

To evaluate this, we define the *number of violations*. Intuitively, for all occurrences p of $C_{\mathrm{k_{max}}+2}$ the number of graphs $C' \in \mathrm{IG}(C_{\mathrm{k_{max}}+2}, C_{\mathrm{k_{max}}+3})$ with $p \not\models \exists C'$ is added up, and by comparing these numbers for G and H it can be determined whether there have been more increasing insertions and deletions than decreasing ones.

The number of violations is defined for each layer of the constraint, but only for the first unsatisfied layer the sum is calculated as described above. For all layers k with

$k \leq \mathrm{k_{max}}$ it is set to 0 and for all layers k with $k > \mathrm{k_{max}}+1$ it is set to ∞. In this way, a transformation $t : G \Longrightarrow H$ that increases the largest satisfied layer can be easily detected, since the number of violations in H at layer $\mathrm{k_{max}}+1$ will be set to 0.

Definition 3.14 (**number of violations**). *Given a graph G and a constraint c in UANF, and let $e = \mathrm{sub}_{\mathrm{k_{max}}+2}(c)$. The* number of violations $\mathrm{nv}_j(c, G)$ at layer $-1 \leq j < \mathrm{nl}(c)$ in G *is defined as:*

$$\mathrm{nv}_j(c,G) := \begin{cases} 0 & \text{if } j < \mathrm{k_{max}}+1 \\ \sum_{C' \in \mathrm{IG}(C_{j+1}, C_{j+2})} |\{q \mid q : C_{j+1} \hookrightarrow G \wedge q \not\models \mathrm{IC}_0(e, C')\}| & \text{if } e \neq \mathit{false} \text{ and } j = \mathrm{k_{max}}+1 \\ |\{q \mid q : C_{j+1} \hookrightarrow G\}| & \text{if } e = \mathit{false} \text{ and } j = \mathrm{k_{max}}+1 \\ \infty & \text{if } j > \mathrm{k_{max}}+1 \end{cases}$$

Note that the second and third cases of Definition 3.14 only occur if $G \not\models c$ and $\mathrm{sub}_{\mathrm{k_{max}}}(c)$ is a existential condition. So e is also existentially bound or equal to false if c ends with $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$ and $\mathrm{k_{max}} = \mathrm{nl}(c) - 2$. Also note that the sets described above do contain occurrences of $C_{\mathrm{k_{max}}+2}$ whose removal (or repair such that they satisfy $\exists(C_{\mathrm{k_{max}}+3}, \mathsf{true})$) will never lead to an increase of the largest satisfied layer. In particular, only the occurrences of $p : C_{\mathrm{k_{max}}+2} \hookrightarrow G$ that are so-called *potentially increasing occurrences at layer* $\mathrm{k_{max}}$ *w.r.t.* c need to be considered.

Definition 3.15 (**potentially increasing occurrences at layer**). *Given a graph G, a constraint c in UANF and an occurrence $p : C_{k+2} \hookrightarrow G$ of an universally bound graph C_k. Then p is called a* potentially increasing occurrence at layer k w.r.t. c *if*

1. $p \not\models \mathrm{cut}_0(\mathrm{sub}_{k+2}(c))$.
2. $p = a_{k+1} \circ \ldots \circ a_0 \circ q$ *where* $a_i \circ \ldots \circ q \models \mathrm{sub}_{i+1}(\mathrm{cut}_k(c))$ *for all* $0 \leq i \leq k$ *and* $q : \emptyset \hookrightarrow G$ *is the empty morphism.*

We will show this in the following lemma.

Lemma 3.16. *Given a graph G, a constraint c in UANF and an odd $-1 \leq k < \mathrm{nl}(c) - 3$ such that $G \models_k c$. Then,*

$$G \models_{k+2} c$$

if for all occurrences $p : C_{k+2} \hookrightarrow G$ of C_{k+2} where $p = a_{k+1} \circ \ldots \circ a_0 \circ q$ and $a_i \circ \ldots \circ q \models \mathrm{sub}_{i+1}(\mathrm{cut}_k(c))$ for all $0 \leq i \leq k$ with the empty morphism $q : \emptyset \hookrightarrow G$ it holds that $p \models \mathrm{cut}_0(\mathrm{sub}_{\mathrm{k_{max}}+2}(c))$.

Proof. Assume that $G \not\models_{k+2} c$ and for all occurrence $p : C_{k+2} \hookrightarrow G$ where $p = a_{k+1} \circ \ldots \circ a_0 \circ q$ and $a_i \circ \ldots \circ q \models \mathrm{sub}_{i+1}(\mathrm{cut}_k(c))$ for all $0 \leq i \leq k$ it holds that $p \models \mathrm{cut}_0(\mathrm{sub}_{\mathrm{k_{max}}+2}(c))$. Since $G \models_k c$ and $G \not\models_{k+2} c$ there must be a morphism $p : C_{k+2} \hookrightarrow G$ such that $p \not\models \mathrm{cut}_0(\mathrm{sub}_{\mathrm{k_{max}}+2}(c))$, $p = a_{k+1} \circ \ldots \circ q$ and $a_i \circ \ldots \circ q \models \mathrm{sub}_{i+1}(\mathrm{cut}_k(c))$ for all $0 \leq i \leq k$. This is a contradiction. □

Only considering the occurrence of $C_{\mathrm{k_{max}}+2}$ as described above will lead to a more precise definition of the number of violations, and therefore of the consistency maintaining and increasing rules, with the drawback that the application conditions designed for this more precise version will be much more complex, since it will be necessary to check that repaired occurrences satisfy the additional condition. We have therefore decided to use this less precise definition of the number of violations.

Using the number of violations, we are now ready to define *consistency maintaining* and *increasing* transformations and rules by checking that the number of violations has not increased or, in the case of consistency increasing, has decreased. In addition, we will also introduce a weaker notion, called *consistency maintaining and increasing rules at layer*. Intuitively, a rule is consistency-maintaining or consistency-increasing w.r.t. c at layer k if all of its applications at graphs G with $G \models_k c$ are consistency-maintaining or consistency-increasing w.r.t. c. This notion will be important for our application conditions, since these only consider graphs of a constraint up to a certain layer. Therefore, it is clear, that rule equipped with these application conditions are not consistency-maintaining or consistency-increasing rules with the advantage of less complex application conditions.

Definition 3.17 (**consistency maintaining and increasing transformations and rules**)**.** *Given a graph G, a constraint c in UANF and a rule ρ. A transformation $t : G \Longrightarrow_{\rho,m} H$ is called* consistency-maintaining w.r.t. *c, if*

$$\mathrm{nv}_k(c, H) \leq \mathrm{nv}_k(c, G)$$

for all $-1 \leq k < \mathrm{nl}(c)$. The transformation is called consistency-increasing w.r.t. *c if it is consistency-maintaining w.r.t. c and*

$$\mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, H) < \mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, G).$$

A rule ρ is called consistency maintaining or increasing w.r.t *c, if all of its transformations are.*

A rule ρ is called consistency maintaining w.r.t. *c at layer $-1 \leq k < \mathrm{nl}(c)$ if all transformations $t : G \Longrightarrow_{\rho,m} H$ with $\mathrm{k_{max}}(c, G) = k$ are consistency maintaining w.r.t. c. Analogously, a rule ρ is called* consistency increasing w.r.t. *c at layer $-1 \leq k < \mathrm{nl}(c)$ if all transformations $t : G \Longrightarrow_{\rho,m} H$ with $\mathrm{k_{max}}(c, G) = k$ are consistency-increasing w.r.t. c.*

Note that if $G \models c$, there is no consistency-increasing transformation $G \Longrightarrow H$ w.r.t c, since $\mathrm{nv}_j(c, G) = 0$ for all $0 \leq j < \mathrm{nl}(c)$. Also note that a consistency-maintaining rule at layer $\mathrm{nl}(c) - 1$ w.r.t. c is also a consistency-maintaining rule w.r.t. c, since all graphs to which this rule is applied satisfy c. For the same reason, there is no consistency-increasing rule at layer $\mathrm{nl}(c) - 1$ w.r.t. c. Also, no plain rule ρ is consistency-increasing w.r.t c, since a graph G satisfying c such that a transformation $t : G \Longrightarrow_{\rho,m} H$ exists can always be constructed. Therefore, every consistency-increasing rule must have at least one application condition.

As mentioned above, a transformation is considered to be consistency-increased if the largest satisfied layer is increasing. This property is already indirectly embedded in the definition of consistency-increasing transformations.

Theorem 3.1. *Given a rule ρ, a constraint c in UANF and a graph G with $G \not\models c$. A transformation $t : G \Longrightarrow_{\rho,m} H$ is consistency-increasing w.r.t. c if*

$$\mathrm{k}_{\max}(c, G) < \mathrm{k}_{\max}(c, H).$$

.

Proof. No $\ell > \mathrm{k}_{\max}(c, G)$ with $G \models_\ell c$ exists. Hence, $\mathrm{nv}_{\mathrm{k}_{\max}(c,G)+1}(c, G) > 0$. Since $\mathrm{k}_{\max}(c, H) > \mathrm{k}_{\max}(c, G)$, $\mathrm{nv}_{\mathrm{k}_{\max}(c,G)+1}(c, H) = 0$, which immediately implies that t is consistency-increasing w.r.t. c. □

Since there are no consistency-increasing transformations starting from consistent graphs, there are no infinitely long sequences of consistency-increasing transformations.

Theorem 3.2. *Let c be a constraint in UANF. Every sequence of consistency-increasing transformations w.r.t. c is finite.*

Proof. Let

$$G_0 \Longrightarrow_{\rho_0,m_0} G_1 \Longrightarrow_{\rho_1,m_1} G_2 \Longrightarrow_{\rho_2,m_2} \dots$$

be a sequence of consistency-increasing transformations w.r.t. c. We assume that $\mathrm{k}_{\max}(c, G_0) < \mathrm{nl}(c) - 1$, otherwise $\mathrm{nv}_j(c, G_0) = 0$ for all $0 \leq j < \mathrm{nl}(c)$ and there is no consistency-increasing transformation $G_0 \Longrightarrow H$ with respect to c.

We show that $G_x \models_{\mathrm{k}_{\max}(c,G_0)+2} c$ holds after a maximum of $x := \mathrm{nv}_{\mathrm{k}_{\max}(c,G_0)+1}(c, G_0)$ transformations. Note that x must be finite, since G_0 contains only a finite number of occurrences of $C_{\mathrm{k}_{\max}(c,G_0)+2}$. Since every transformation is consistency-increasing w.r.t. C, it follows that $\mathrm{nv}_{\mathrm{k}_{\max}(c,G_i)+1}(c, G_{i+1}) \leq \mathrm{nv}_{\mathrm{k}_{\max}(c,G_i)+1}(c, G_i) - 1$ after each transformation. Therefore, after at most x transformations, $\mathrm{nv}_{\mathrm{k}_{\max}(c,G_0)+1}(c, G_j) \leq \mathrm{nv}_{\mathrm{k}_{\max}(c,G_0)+1}(c, G_0) - x = 0$ and thus $G_x \models_{\mathrm{k}_{\max}(c,G_0)+2} c$. If this is applied iteratively, it follows that after a finite number of transformations there must exist a graph G_k with $G_k \models c$. Since there is no consistency increasing transformation $G_k \Longrightarrow_{\rho_k,m_k} G_{k+1}$, the sequence must be finite. □

3.4 Direct Consistency Maintaining and Increasing Transformations

We will now introduce stricter versions of consistency-increasing and consistency-maintaining transformations, called *direct consistency-maintaining* and *direct consistency-increasing* transformations and rules. These are consistency-maintaining and consistency-increasing transformations, which do not perform any unnecessary insertions and deletions. For example, given a constraint c in UANF and graphs G with $G \not\models c$ and H with $H \models c$, the transformation $t : G \Longrightarrow_{\rho,\mathrm{id}_G} H$ via the rule $\rho = G \xhookleftarrow{l} \emptyset \xhookrightarrow{r} H$ is a consistency-increasing transformation. Therefore, the notions of consistency-increasing

and consistency-maintaining transformations allow insertions or deletions that are unnecessary in order to increase or maintain consistency. That is, deleting occurrences of existentially bound graphs, deleting occurrences $p : C_k \hookrightarrow G$ of universally bound graphs C_k satisfying $\exists(C_{k+1}, \mathsf{true})$ or inserting occurrences of universally bound graphs and inserting occurrences p of intermediate graphs $C' \in \mathrm{IG}(C_{k-1}, C_k)$ such that each occurrence q of C_{k-1} with $q = p \circ a^r_{k-1}$ already satisfies $\exists(C', \mathsf{true})$.

Direct consistency-increasing and *maintaining* transformations are more restricted, in the sense that these unnecessary deletions and insertions cause a transformation not to be direct consistency-increasing or direct consistency maintaining, respectively. In addition, we can use second-order logic formulas to characterise these transformations. Furthermore, these formulas ensure that now new violations are inserted. Thus, the removal of one violation is sufficient to state that the transformation is (direct) consistency-increasing, which can also be expressed using a second-order logic formula. We start by introducing *direct consistency-maintaining* transformations, rules and the weaker notion of *direct consistency-maintaining rules at layer*. The definition of direct consistency maintaining transformations consists of the following formulas:

1. *No new violation by deletion:* This condition ensures that the consistency is not reduced by deleting intermediate graphs $C' \in \mathrm{IG}(C_{\mathrm{k_{max}}+2}, C_{\mathrm{k_{max}}+3})$. This leads to the insertion of new violations only if an occurrence of $C_{\mathrm{k_{max}}+2}$ which satisfies $\exists(C', \mathsf{true})$ in the originating graph does not satisfy $\exists(C', \mathsf{true})$ in the derived graph of the transformation. Therefore, this condition checks that this case does not occur.

2. *No new violation by insertion:* This condition ensures that the consistency is not decreased by inserting an occurrence of $C_{\mathrm{k_{max}}+2}$. Again, this only causes a new violation to be inserted if that occurrence does not satisfy $\exists(C_{\mathrm{k_{max}}+3}, \mathsf{true})$. The condition checks that this is not the case.

3. *No satisfied layer reduction by insertion:* This condition ensures that the largest satisfied layer is not reduced by inserting a universally bound graph C_j. This can only happen if $j \leq \mathrm{k_{max}}$, and the condition checks that no occurrences of such universally bound graphs are inserted.

4. *No satisfied layer reduction by deletion:* This condition ensures that the largest satisfied layer is not reduced by deleting an existentially bound graph C_j. Again, this can only happen if $j \leq \mathrm{k_{max}}$. The condition checks that no occurrences of such existentially bound graphs are deleted.

The *no new violation by deletion* and *no new violation by insertion* conditions ensure that the number of violations is not increased, and the *no satisfied layer reduction by insertion* and *no satisfied layer reduction deletion* conditions ensure that the largest satisfied layer is not reduced. Of course, the insertion of universal and deletion of existential graphs does not necessarily lead to a decrease of the largest satisfied layer, but it can also be considered as an unnecessary insertion or deletion.

Since a condition c in UANF is also allowed to end with $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$, *no new violation by deletion* and *no new violation by insertion* contain case discrimination. If the constraint c ends with $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$ and $\mathrm{k_{max}} = \mathrm{nl}(c) - 2$, there is no graph $C_{\mathrm{k_{max}}+3}$ and thus no intermediate graphs. Therefore, there is no new violation by deletion and this formula is set equal to true and the *no new violation by insertion* formula will check that no new occurrence of $C_{\mathrm{k_{max}}+2}$ are introduced at all.

For the rest of this paper, we will assume that the empty conjunction is always equal to true.

Definition 3.18 (**direct consistency maintaining transformations and rules**)**.** *Given a graph G, a rule ρ and a constraint c in UANF. If $G \models c$, a transformation $t : G \Longrightarrow_{\rho,m} H$ is called* direct consistency-maintaining w.r.t. *c if $H \models c$. Otherwise, if $G \not\models c$, let $\mathrm{k_{max}} = \mathrm{k_{max}}(c, G)$ and $e = \mathrm{sub}_{\mathrm{k_{max}}+2}(c)$. A transformation $t : G \Longrightarrow_{\rho,m} H$ is called* direct consistency maintaining w.r.t. *c if the following formulas are satisfied.*

1. No new violation by deletion: *If $e \neq \mathsf{false}$, then each occurrence of $C_{\mathrm{k_{max}}+2}$ in G which satisfies $\mathrm{IC}_0(e, C')$ for any $C' \in \mathrm{IG}(C_{\mathrm{k_{max}}+2}, C_{\mathrm{k_{max}}+3})$ still satisfies $\mathrm{IC}_0(e, C')$ in H:*

$$\forall p : C_{\mathrm{k_{max}}+2} \hookrightarrow G\Big(\bigwedge_{C' \in \mathrm{IG}(C_{\mathrm{k_{max}}+2}, C_{\mathrm{k_{max}}+3})} \big(p \models \mathrm{IC}_0(e, C') \wedge \mathrm{tr}_t \circ p \text{ is total}\big) \\ \Longrightarrow \mathrm{tr}_t \circ p \models \mathrm{IC}_0(e, C')\Big) \tag{3.1}$$

   Otherwise, if $e = \mathsf{false}$, this formula is equal to true.

2. No satisfied layer reduction by insertion: *Let $d = \mathrm{IC}_0(e, C_{\mathrm{k_{max}}+3})$ if $e \neq \mathsf{false}$ and $d = \mathsf{false}$ otherwise. Each newly inserted occurrence of $C_{\mathrm{k_{max}}+2}$ satisfies d.*

$$\forall p' : C_{\mathrm{k_{max}}+2} \hookrightarrow H\big(\neg\exists p : C_{\mathrm{k_{max}}+2} \hookrightarrow G(p' = \mathrm{tr}_t \circ p) \implies p' \models d\big) \tag{3.2}$$

3. No satisfied layer reduction by insertion: *No occurrence of a universally bound graph C_j with $j \leq \mathrm{k_{max}}$ is inserted.*

$$\bigwedge_{\substack{i < \mathrm{k_{max}} \\ C_i \text{ universal}}} \forall p : C_i \hookrightarrow H(\exists p' : C_i \hookrightarrow G(p = \mathrm{tr}_t \circ p')) \tag{3.3}$$

4. No satisfied layer reduction by deletion: *No occurrence of an existentially bound graph C_j with $j \leq \mathrm{k_{max}} + 1$ is deleted.*

$$\bigwedge_{\substack{i \leq \mathrm{k_{max}} \\ C_i \text{ existential}}} \forall p : C_i \hookrightarrow G(\mathrm{tr}_t \circ p \text{ is total}) \tag{3.4}$$

A rule ρ is called direct consistency-maintaining w.r.t. *c if all of its transformations are. A rule ρ is called* direct consistency maintaining w.r.t. *c* at layer *$-1 \leq k < \mathrm{nl}(c)$ if all transformations $t : G \Longrightarrow_{\rho,m}$ with $\mathrm{k_{max}}(c, G) = k$ are direct consistency-maintaining w.r.t. c.*

Before continuing with the definition of direct consistency-increasing transformations and rules, let us first show that every direct consistency-maintaining transformation is indeed consistency-maintaining. To do this, we first show that satisfying the *no satisfied layer reduction by insertion* and *no satisfied layer reduction by insertion* formulas guarantees that the largest satisfied layer is not decreased.

Lemma 3.19. *Given a transformation $t : G \Longrightarrow H$ and a constraint c in UANF such that the* no satisfied layer reduction by insertion *and* no satisfied layer reduction by insertion *formulas are satisfied. Then*

$$H \models_{\mathrm{k_{max}}(c,G)} c.$$

Proof. Let us assume that $H \not\models_{\mathrm{k_{max}}(c,G)} c$. Then either a new occurrence of a universally bound graph C_i with $i < \mathrm{k_{max}}(c, G)$ has been inserted, or an occurrence of an existentially bound graph C_j with $j \leq \mathrm{k_{max}}(c, G)$ has been destroyed. Therefore, the following applies:

$$\exists p : C_i \hookrightarrow H(\neg \exists p' : C_i \hookrightarrow G(p = \mathrm{tr}_t \circ p')) \vee \exists p : C_j \hookrightarrow G(\mathrm{tr}_t \circ p \textit{ is not total})$$

where $i, j \leq \mathrm{k_{max}}(c, G)$, i is even and j is odd, i.e. C_i is universally and C_j is existentially bound. It follows immediately that either the *no satisfied layer reduction by insertion* and *no satisfied layer reduction by insertion* formula is not satisfied. This is a contradiction. □

With this we are now going to show that a direct consistency-maintaining transformation is also a consistency-maintaining transformation.

Theorem 3.3. *Given a graph G, a constraint c in UANF, a rule ρ and a direct consistency-maintaining transformation $t : G \Longrightarrow_{\rho,m} H$ w.r.t. c. Then, t is also a consistency-maintaining transformation.*

Proof. Lemma 3.19 implies that $\mathrm{k_{max}}(c, G) \leq \mathrm{k_{max}}(c, H)$ and it immediately follows that $\mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, H) \neq \infty$. It remains to show that $\mathrm{nv}_k(c, H) \leq \mathrm{nv}_k(c, G)$ for all $0 \leq k < \mathrm{nl}(c)$. In particular, we only need to show that $\mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, H) \leq \mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, G)$ since for all $-1 \leq j < \mathrm{k_{max}}(c, G) + 1$ it holds that $\mathrm{nv}_j(c, H) = \mathrm{nv}_j(c, G) = 0$. And since $\mathrm{nv}_j(c, G) = \infty$ for all $\mathrm{k_{max}}(c, G) + 1 < j < \mathrm{nl}(c)$, it follows that $\mathrm{nv}_j(c, H) \leq \mathrm{nv}_j(c, G)$ for all $\mathrm{k_{max}}(c, G) + 1 < j < \mathrm{nl}(c)$.

Let $\mathrm{k_{max}} = \mathrm{k_{max}}(c, G)$ and $d = \mathrm{sub}_{\mathrm{k_{max}}+2}(c)$. We show that the satisfaction of the *no new violation by deletion* and *no new violation by insertion* formulas imply that $\mathrm{nv}_{\mathrm{k_{max}}+1}(c, H) \leq \mathrm{nv}_{\mathrm{k_{max}}+1}(c, G)$.

Let us assume that $\mathrm{nv}_{\mathrm{k_{max}}+1}(c, H) > \mathrm{nv}_{\mathrm{k_{max}}+1}(c, G)$. Therefore, there is a morphism $p : C_{\mathrm{k_{max}}+2} \hookrightarrow H$ with $p \not\models \mathrm{IC}_0(d, C')$ for some $C' \in \mathrm{IG}(C_{\mathrm{k_{max}}+2}, C_{\mathrm{k_{max}}+3})$ such that either 1. or 2. below is satisfied. Note that this is only the case if $d \neq \mathsf{false}$. Otherwise there must be a morphism p which satisfies 2.

1. There is a morphism $q' : C_{\mathrm{k_{max}}+2} \hookrightarrow G$ with $q' \models \mathrm{IC}_0(d, C')$ and $p = \mathrm{tr}_t \circ q'$.
2. There is no morphism $q : C_{\mathrm{k_{max}}+2} \hookrightarrow G$ with $p = \mathrm{tr}_t \circ q$.

This is a contradiction, if 1 is satisfied, q' does not satisfy the *no new violation by deletion* formula. If (2) is satisfied, p does not satisfy the *no new violation by insertion* formula since p only satisfies $\mathrm{IC}_0(d, C_{k+2})$ if p satisfies $\mathrm{IC}_0(d, C')$ for all $C' \in \mathrm{IG}(C_{k+1}, C_{k+2})$. It follows that

$$\mathrm{nv}_k(c, H) \leq \mathrm{nv}_k(c, G)$$

holds and t is a consistency-maintaining transformation.

□

The following corollary arises as a direct consequence of Theorem 3.3.

Corollary 3.20. *Given a constraint c in UANF and a rule ρ. If ρ is a direct consistency-maintaining rule w.r.t. c, ρ is also a consistency-maintaining rule w.r.t. c. If ρ is a direct consistency-maintaining rule w.r.t. c at layer $-1 \leq k \leq \mathrm{nl}(c)$, ρ is also a consistency-maintaining rule w.r.t. c at layer k.*

Let us now introduce the notions of *direct consistency-increasing* transformations, rules and *direct consistency-increasing rules at layer*. Similar to the definition of consistency-maintaining and consistency-increasing transformations, the notion of *direct consistency-increasing transformations* is based on the notion of direct consistency-maintaining transformations, in the sense that a direct consistency-increasing transformation is also a direct consistency-maintaining one. Since a direct consistency-maintaining transformation t does not introduce any new violations, it is sufficient that t removes at least one violation to say that t is direct consistency-increasing.

Again, we need case discrimination if the constraint ends with $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$ and $\mathrm{k}_{\max} = \mathrm{nl}(c) - 2$. So we will use two second-order logic formulas, one for the general case and one for this special case.

1. *General increasing formula*: This formula is satisfied if either an occurrence of $C_{\mathrm{k}_{\max}+2}$ that does not satisfy $\exists(\mathrm{k}_{\max}+3, \mathsf{true})$ is deleted, or an occurrence of $C_{\mathrm{k}_{\max}+2}$ which does not satisfy $\exists(C', \mathsf{true})$ in the first graph of the transformation satisfies $\exists(C', \mathsf{true})$ in the second graph of the transformation where $C' \in \mathrm{IG}(C_{\mathrm{k}_{\max}+2}, C_{\mathrm{k}_{\max}+3})$. Both cases result in the removal of a violation.

2. *Special increasing formula*: This formula is satisfied if an occurrence of $C_{\mathrm{k}_{\max}+2}$ is removed. In the special case this is the only way to remove a violation.

Definition 3.21 (**direct consistency-increasing transformations and rules**)**.** *Given a constraint c in UANF, a rule ρ, a graph G with $G \not\models c$ and let $e = \mathrm{sub}_{\mathrm{k}_{\max}+2}(c)$.*

A transformation $t : G \Longrightarrow_{\rho,m} H$ is called direct consistency-increasing w.r.t. *c if it is direct consistency-maintaining w.r.t. c and either the special increasing condition is satisfied if $\mathrm{sub}_{\mathrm{nl}(c)-1}(c) = \forall(C_{\mathrm{nl}(c)}, \mathsf{false})$ and $\mathrm{k}_{\max} = \mathrm{nl}(c) - 2$ or the general increasing condition is satisfied otherwise.*

1. *General increasing formula:*

$$\exists p : C_{\mathrm{k_{max}}+2} \hookrightarrow G \Big(\bigvee_{C' \in \mathrm{IG}(\mathrm{k_{max}}+2, \mathrm{k_{max}}+3)} (p \not\models \mathrm{IC}_0(e, C') \wedge (\mathrm{tr}_t \circ p \textit{ is not total } \vee \mathrm{tr}_t \circ p \models \mathrm{IC}_0(e, C'))) \Big) \tag{3.5}$$

2. *Special increasing formula:*

$$\exists p : C_{\mathrm{k_{max}}+2} \hookrightarrow G(\mathrm{tr}_t \circ p \textit{ is not total}) \tag{3.6}$$

A rule ρ is called direct consistency-increasing w.r.t. *c if all of its transformations are. A rule ρ is called* direct consistency-increasing w.r.t. *c at layer $-1 \leq k < \mathrm{nl}(c)$ if all transformations $t : G \Longrightarrow_{\rho,m} H$ with $\mathrm{k_{max}}(c, G) = k$ are direct consistency-increasing w.r.t. c.*

Note that the satisfaction of the *no satisfied layer reduction by insertion* and *no satisfied layer reduction by deletion* formulas not only ensure that the largest satisfied layer does not decrease, as shown in Lemma 3.19, but also prevent further unnecessary insertions and deletions, since inserting a universally bound graph and deleting an existentially bound graph will never lead to an increase in consistency.

Now, we will show the already indicated relation between direct consistency-increasing and consistency-increasing transformations, namely that a direct consistency-increasing transformation is also consistency-increasing transformation. Counterexamples in which the inversion of the implication does not hold can be easily constructed, showing that these notions are not identical but related.

Theorem 3.4. *Given a constraint c in UANF, a rule ρ, a graph G with $G \not\models c$ and a direct consistency-increasing transformation $t : G \Longrightarrow_{\rho,m} H$ w.r.t. c. Then, t is also a consistency-increasing transformation.*

Proof. Theorem 3.3 implies that t is a consistency maintaining transformation. Therefore, it is sufficient to show that $\mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, H) < \mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, G)$. Let $\mathrm{k_{max}} = \mathrm{k_{max}}(c, G)$ and $d = \mathrm{sub}_{\mathrm{k_{max}}+2}(c)$ with $d \neq \mathsf{false}$.

Then, the general increasing formula is satisfied, so there exists a intermediate graph $C' \in \mathrm{IG}(C_{\mathrm{k_{max}}+2}, C_{\mathrm{k_{max}}+3})$ and a morphism $p : C_{\mathrm{k_{max}}+2} \hookrightarrow G$ with $p \not\models \mathrm{IC}_0(d, C')$, such that either $\mathrm{tr} \circ p$ is total and $\mathrm{tr}_t \circ p \models \mathrm{IC}_0(d, C')$ or $\mathrm{tr} \circ p$ is not total. In both cases the following applies:

$$p \in \{q \mid q : C_{\mathrm{k_{max}}+2} \hookrightarrow G \text{ and } q \not\models \mathrm{IC}_0(d, C')\} \text{ and}$$
$$\mathrm{tr} \circ p \notin \{q \mid q : C_{\mathrm{k_{max}}+2} \hookrightarrow H \text{ and } q \not\models \mathrm{IC}_0(d, C')\}$$

Since t is direct consistency maintaining, it follows that

$$|\{q \mid q : C_{\mathrm{k_{max}}+2} \hookrightarrow G \text{ and } q \not\models \mathrm{IC}_0(d, C)\}| \leq |\{q \mid q : C_{\mathrm{k_{max}}+2} \hookrightarrow H \text{ and } q \not\models \mathrm{IC}_0(d, C)\}|.$$

for all $C \in \mathrm{IG}(C_{\mathrm{k_{max}}+2}, C_{\mathrm{k_{max}}+3})$. Furthermore, this inequality is strictly satisfied if $C = C'$. It immediately follows that $\mathrm{nv}_k(c, H) < \mathrm{nv}_k(c, G)$ and t is a consistency-increasing transformation.

If $d = \mathsf{false}$, i.e. $\mathrm{sub}_{\mathrm{nl}(c)-1}(c) = \forall(C_{\mathrm{nl}(c)}, \mathsf{true})$ and $\mathrm{k_{max}} = \mathrm{nl}(c) - 2$, the special increasing formula is satisfied. It holds that

$$|\{q \mid q : C_k \hookrightarrow G\}| \leq |\{q \mid q : C_k \hookrightarrow H\}|,$$

and since t is a direct consistency-maintaining transformation, it can be shown in a similar way as above that satisfying the special increasing formula implies that

$$|\{q \mid q : C_k \hookrightarrow G\}| < |\{q \mid q : C_k \hookrightarrow H\}|.$$

It follows that t is a consistency-increasing transformation. □

Again, the following corollary is a direct consequence of Theorem 3.4.

Corollary 3.22. *Given a constraint c in UANF and a rule ρ. If ρ is a direct consistency-increasing rule w.r.t. c, ρ is also a consistency-increasing rule w.r.t. c. If ρ is a direct consistency-increasing rule w.r.t. c at layer $-1 \leq k \leq \mathrm{nl}(c)$, ρ is also a consistency-increasing rule w.r.t. c at layer k.*

Example 3.7. *Consider constraint c_1 given in Figure 5, the transformations t_1, t_2 and the set $\mathrm{IG}(C_1^1, C_2^1)$ given in Figure 7. Then, t_1 is a consistency-maintaining transformation w.r.t. c_1. The number of violations in both graphs is 9. In the first graph, the occurrence* `c1` *does not satisfy $\exists(I_3, \mathsf{true})$, $\exists(I_4, \mathsf{true})$, $\exists(I_5, \mathsf{true})$ and $\exists(I_6, \mathsf{true})$, the occurrence* `c2` *does not satisfy $\exists(I_2, \mathsf{true})$, $\exists(I_3, \mathsf{true})$, $\exists(I_4, \mathsf{true})$, $\exists(I_5, \mathsf{true})$ and $\exists(I_6, \mathsf{true})$. In the second graph, these roles are swapped, i.e.* `c1` *satisfies exactly the intermediate conditions that* `c2` *satisfied in the first graph, and vice versa. But, t_1 is not a direct consistency-maintaining transformation, since the occurrence* `c1` *satisfies $\exists(I_2, \mathsf{true})$ in the first but not in the second graph. Therefore, the* no new violation by deletion *formula is not satisfied.*

The transformation t_2 is consistency increasing w.r.t. c_1. The number of violations in the first graph is equal to 14. The occurrence `c1` *does not satisfy $\exists(I_3, \mathsf{true})$, $\exists(I_4, \mathsf{true})$, $\exists(I_5, \mathsf{true})$ and $\exists(I_6, \mathsf{true})$. Both occurrences* `c2` *and* `c3` *do not satisfy $\exists(I_2, \mathsf{true})$, $\exists(I_3, \mathsf{true})$, $\exists(I_4, \mathsf{true})$, $\exists(I_5, \mathsf{true})$ and $\exists(I_6, \mathsf{true})$.*

In the second graph, `c1` *does not satisfy $\exists(I_2, \mathsf{true})$, $\exists(I_4, \mathsf{true})$, $\exists(I_5, \mathsf{true})$ and $\exists(I_6, \mathsf{true})$ and both* `c2` *and* `c3` *do not satisfy $\exists(I_6, \mathsf{true})$. Therefore, the number of violations in the second graph is 6.*

But t_2 is not a direct consistency increasing transformation, since `c1` *satisfies $\exists(I_3, \mathsf{true})$ in the first but not in the second graph, and the* no new violation by deletion *formula is not satisfied.*

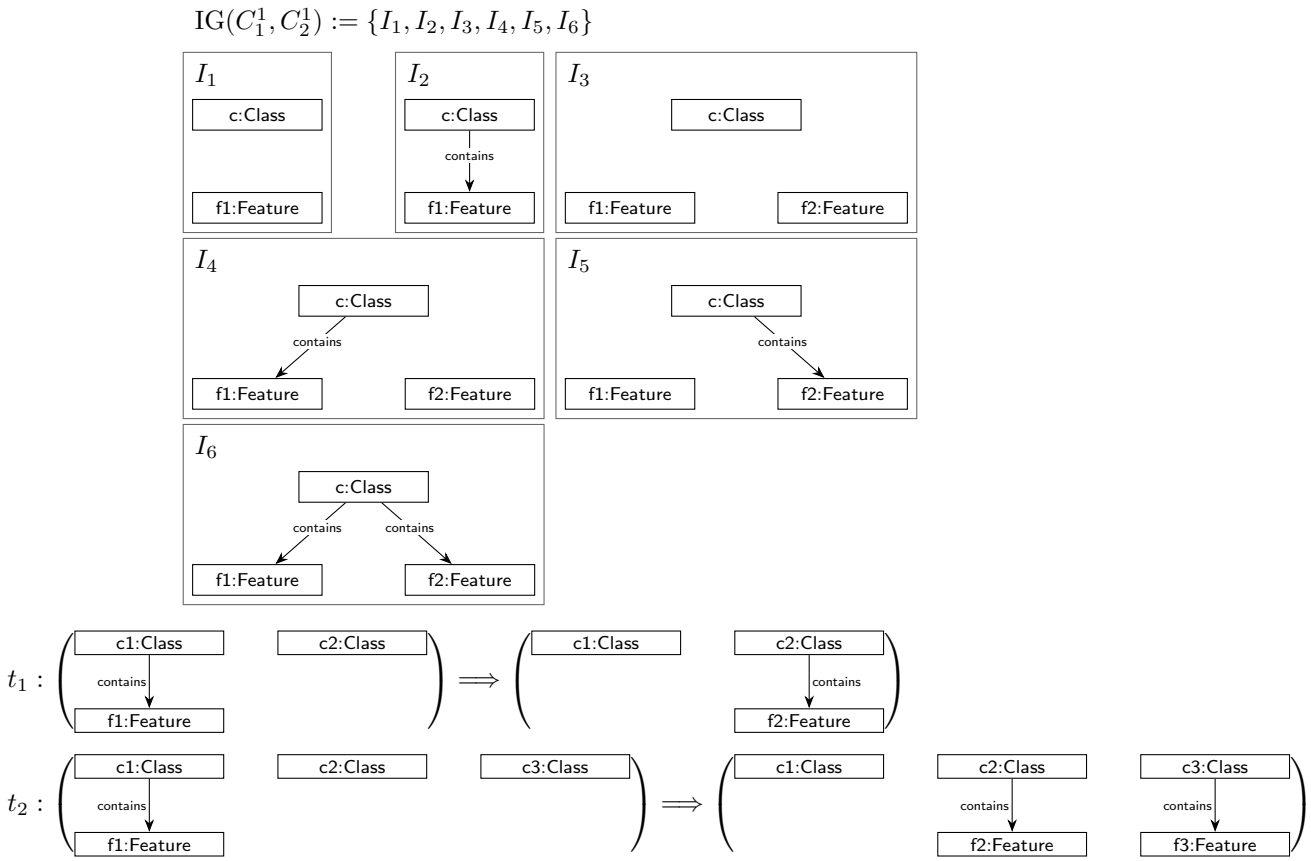

Figure 7: example

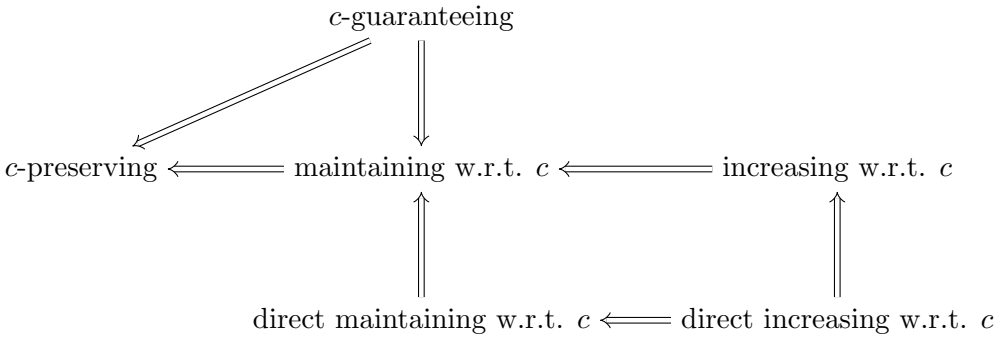

Figure 8: Relations of consistency notions.

3.5 Comparison with other concepts of Consistency

In this chapter, the notions of (direct) consistency increase and maintainment are compared to the already known notions of consistency guaranteeing, consistency preserving [10], (direct) consistency increasing and sustaining [12], in order to reveal relations between them and to ensure that (direct) consistency increase and maintainment are indeed new notions of consistency. These relationships are summarized in Figure 8.

First, we compare (direct) consistency increase and maintainment with the notions of consistency-guaranteeing, preserving, sustaining and improving in the general case and later on, for some special cases. We begin by examining the implications that can be drawn about a consistency-maintaining or consistency-increasing transformation.

Theorem 3.5 (**Implications of a consistency-maintaining or consistency-increasing transformation**). *Given a condition c in UANF and a transformation $t : G \Longrightarrow H$. Then,*

$$
\begin{aligned}
&t \text{ is consistency-maintaining w.r.t. } c && \Longrightarrow && t \text{ is } c\text{-preserving} && \text{and}\\
&t \text{ is consistency-maintaining w.r.t. } c && \not\Longrightarrow && t \text{ is } c\text{-guaranteeing} && \text{and}\\
&t \text{ is direct consistency-increasing w.r.t. } c && \not\Longrightarrow && t \text{ is consistency sustaining w.r.t. } c &&
\end{aligned}
$$

Proof. 1. t is consistency-maintaining w.r.t. $c \implies t$ is c-preserving: Let t be a consistency-maintaining transformation w.r.t. c. If $G \not\models c$, then t is a c-preserving transformation. If $G \models c$, then $\mathrm{nv}_j(c, G) = 0$ for all $0 \leq j < \mathrm{nl}(c)$. Since t is consistency maintaining it follows that $\mathrm{nv}_j(c, H) = 0$ for all $0 \leq j < \mathrm{nl}(c)$ and hence $H \models c$. It follows that t is a c-preserving transformation.

2. t is consistency maintaining w.r.t. $c \not\Longrightarrow t$ is c-guaranteeing: Consider the transformation $t_2 : G \Longrightarrow H$ shown in Figure 7 and constraint c_1 shown in Figure 5. As discussed in Example 3.7, t_2 is consistency-increasing and thus consistency-maintaining w.r.t. c_1. But t is not a c-guaranteeing transformation, since all occurrences of nodes of type `Class` do not satisfy $\exists(C_2^1, \mathtt{true})$.

3. t is direct consistency maintaining w.r.t. $c \not\Longrightarrow t$ is consistency sustaining w.r.t. c: Consider the constraint $c = \forall(C_1^1, \exists(C_2^1, \forall(C_4^2, d)))$, where d is an existentially bound constraint in ANF with $d \neq \mathsf{false}$ composed of the graphs given in Figure 5. And consider transformation t_2 given in Figure 9. Then, t is direct consistency increasing; the *no new violation by deletion*, *no new violation by insertion*, *no satisfied layer reduction by insertion* and *no satisfied layer reduction by deletion* formulas are satisfied and the *general increasing* formula is satisfied because an occurrence of C_1^1 that did not satisfy $\exists(C_2^1, \mathsf{true})$ in G satisfies $\exists(C_2^1, \mathsf{true})$ in H. But this transformation is not consistency-sustaining since the number of occurrences of C_1^1 not satisfying $\exists(C_2^1, \forall(C_4^2, d))$ in H is greater than the number of occurrences of C_1^1 in G not satisfying $\exists(C_2^1, \forall(C_4^2, d))$.

□

These results are not surprising, since consistency-maintaining and consistency-increasing are much stricter notions than guaranteeing and sustaining, in the sense that the notion of violation is more fine-grained. For example, for guaranteeing transformations, an arbitrary number of violations can be introduced as long as the derived graph satisfies the constraint, and thus guaranteeing does not imply direct increasing, since a direct increasing transformation is not allowed to introduce new violations. Let us now examine whether a concept of consistency implies the notions of consistency-maintaining and increasing.

Theorem 3.6 (**Implications of preserving, guaranteeing, sustaining and improving transformations**). *Given a condition c in UANF and a transformation $t : G \Longrightarrow H$. Then,*

t is c-guaranteeing	$\Longrightarrow$	*t is consistency-maintaining w.r.t. c*	*and*
t is c-guaranteeing	$\not\Longrightarrow$	*t is consistency-increasing w.r.t. c*	*and*
t is c-preserving	$\not\Longrightarrow$	*t is consistency-maintaining w.r.t. c*	*and*
t is direct consistency improving w.r.t c	$\not\Longrightarrow$	*t is consistency-maintaining w.r.t. c*	

Proof. 1. t is c-guaranteeing $\Longrightarrow$ t is consistency-maintaining w.r.t. c: Let t be a c-guaranteeing transformation. Then, t is also a consistency-maintaining transformation w.r.t. c since $H \models c$ and therefore $\mathrm{nv}_j(c, H) = 0$ for all $-1 \leq j < \mathrm{nl}(c)$.

2. t is c-guaranteeing $\not\Longrightarrow$ t is consistency-increasing w.r.t. c: Assume that $G \models c$ and $H \models c$. Then, t is a c-guaranteeing transformation, but not a consistency-increasing one.

3. t is c-preserving $\not\Longrightarrow$ t is consistency-maintaining w.r.t. c: Consider graphs C_1^1, C_2^2 and constraint c_1 given in Figure 5. The transformation $t : C_2^2 \Longrightarrow C_1^1$ is c_1-preserving, since $C_2^2 \not\models c_1$, but not consistency maintaining w.r.t. c_1 since $\mathrm{nv}_0(c_1, C_2^2) = 4$ and $\mathrm{nv}_0(c_1, C_1^1) = 6$.

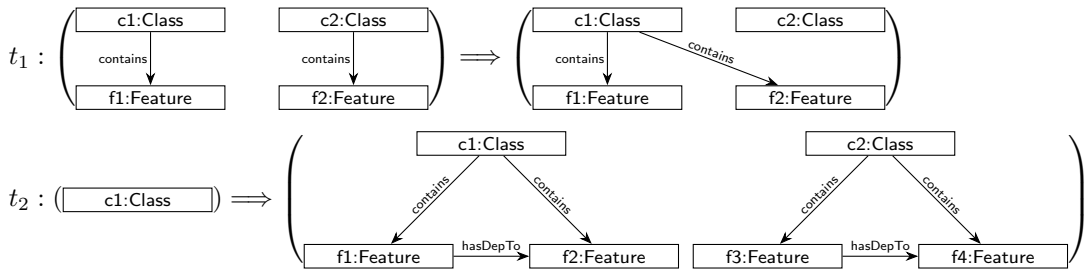

Figure 9: example

4. t is direct consistency-improving w.r.t. $c \not\Longrightarrow t$ is consistency-maintaining w.r.t. c: Consider transformation t_1 given in Figure 9 and constraint c_1 given in Figure 5. The transformation t_1 is direct consistency-improving since no occurrence of C_1^1 is inserted, no occurrence of C_1^1 satisfying $\exists(C_2^1, \mathsf{true})$ is deleted, and one occurrence of C_1 satisfies $\exists(C_2^1, \mathsf{true})$. But, this transformation is not consistency-maintaining w.r.t. c since the number of violations in the first graph is 2 and the number of violations in the second graph is 3.

□

This shows, that in general the notions of (direct) consistency increase and maintainment are not related to (direct) consistency improve- and sustainment. We have only shown some of these relations. Since by definition, (direct) consistency improvement implies (direct) consistency sustainment are related we can conclude results for all pairs of consistency types [12]. An overview of these is given in Table 3.

For some special cases, we can infer other types of relationships.

Theorem 3.7 (Relations of consistency concepts in special cases). *Given a constraint c in UANF and a transformation $t : G \Longrightarrow H$.*

1. *If $G \not\models c$, then*

$$t \text{ is } c\text{-guaranteeing} \implies t \text{ is consistency-increasing w.r.t. } c.$$

2. *If* $\mathrm{nl}(c) = 1$, *then*

$$t \text{ is consistency improving w.r.t } c \iff t \text{ is consistency increasing w.r.t } c$$

Proof. 1. Let t be a c-guaranteeing transformation with $G \not\models c$. Then, t is a consistency-increasing transformation w.r.t. c since $0 < \mathrm{nv}_{\mathrm{k_{max}}(c,G)+1}(c, G) < \infty$ and $\mathrm{nv}_j(c, H) = 0$ for all $-1 \leq j \leq \mathrm{nl}(c)$.

2. Let $\mathrm{nl}(c) = 1$. Since c is in UANF, $\mathrm{sub}_1(c) = \mathsf{false}$ and $\mathrm{nv}_0(c, G)$ is the number of occurrences of C in G. This is exactly the definition of the number of violations for consistency-improving transformations, and the statement immediately follows.

□

$\Longrightarrow$	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
maintaining (1)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
increasing (2)	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓
direct maintaining(3)	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
direct increasing (4)	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓
improving (5)	✗	✗	✗	✗	✓*	✓*	✗*	✗*	✗*	✓*
sustaining(6)	✗	✗	✗	✗	✗*	✓*	✗*	✗*	✗*	✓*
direct improving (7)	✗	✗	✗	✗	✓*	✓*	✓*	✓*	✗*	✓*
direct sustaining (8)	✗	✗	✗	✗	✗*	✓*	✗*	✓*	✗*	✓*
guaranteeing(9)	✓	✗	✗	✗	✓*	✓*	✓*	✓*	✓**	✓**
preserving (10)	✗	✗	✗	✗	✗*	✗*	✗*	✗*	✗**	✓**

Table 3: Overview of the relationships between consistency concepts, "✓" indicates that the notion in this row implies the notion in the column, and "✗" indicates that this implication does not hold. All results marked with "*" are from [12] and those marked with "**" are from [10].

4 Consistency-Maintaining and Increasing Application Conditions

In the following, we present application conditions which ensure that any rule equipped with this application condition is (direct) consistency-increasing at layer or (direct) consistency-maintaining at layer or (direct) consistency-maintaining. In particular, we present application conditions in the general case and for specific rules, called *basic maintaining* and *basic increasing* rules. For basic rules, less complex application conditions can be constructed. Similar to the notions of consistency-maintaining and consistency-increasing, these application conditions will only consider graphs of the constraint up to a certain layer and we will show that these rules are direct consistency-maintaining and direct consistency-increasing rules at layer.

4.1 Application Condition for general Rules

We will now present consistency-maintaining and consistency-increasing application conditions at layer for general rules. That means, a rule equipped with these application conditions is consistency-maintaining or consistency-increasing at layer respectively. We will also introduce an application condition such that any rule equipped with it is indeed a consistency-maintaining rule.

Let us start with the consistency-maintaining application condition. The maintaining application condition has a odd parameter $-1 \leq k < \mathrm{nl}(c)$ which specifies which graphs of a constraint are to be considered. In particular, only the graphs C_j with $0 \leq j \leq k+3$ are considered. Note that there is no graph with $\mathrm{k_{max}}$ odd and $\mathrm{k_{max}} \neq \mathrm{nl}(c) - 1$. So it is not a restriction that k must be odd. The maintaining application condition consists of the following three parts, which also use the k parameter:

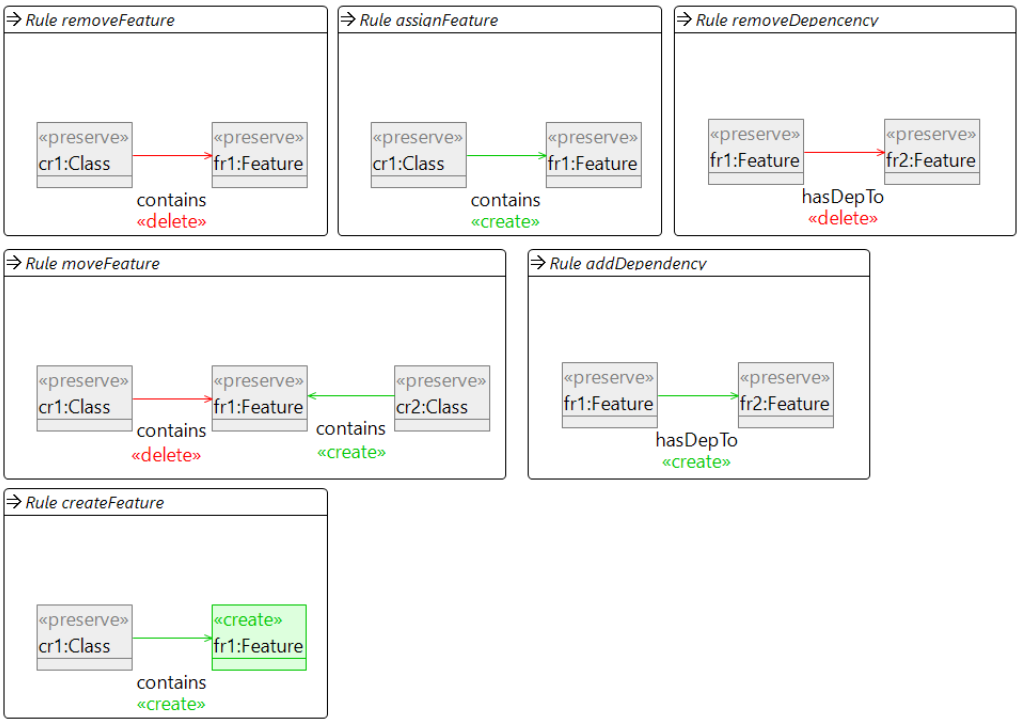

Figure 10: rules

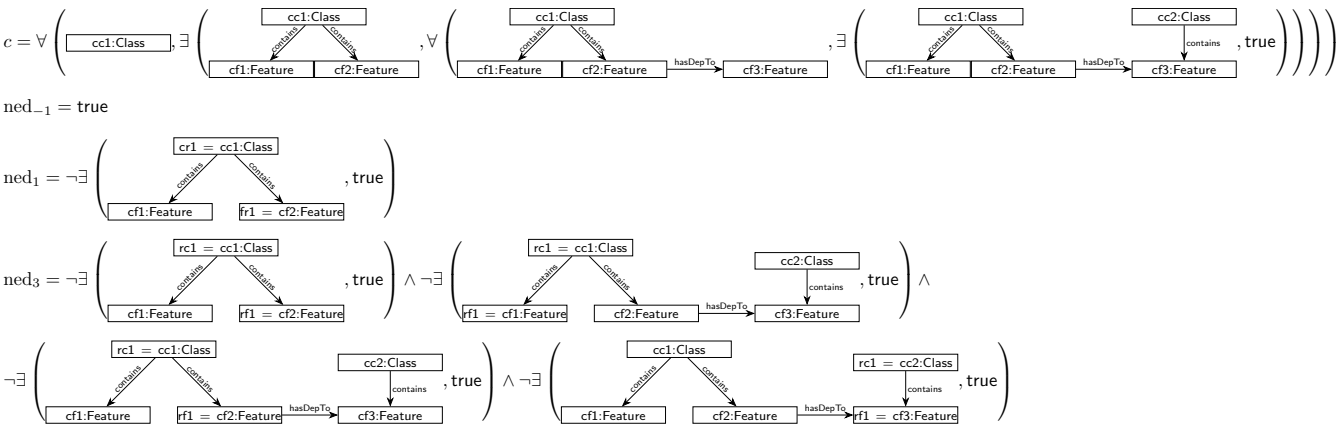

Figure 11: no existentially deleted conditions for the constraint and rule `removeFeature`.

1. *No violation inserted (*$\mathrm{vio}_k()$*):* This application condition checks that no new violations are introduced by removing occurrences of intermediate graphs from the set $\mathrm{IG}(C_{k+2}, C_{k+3})$. It corresponds to the *no new violation by deletion* formula in the sense that a rule that satisfies this application condition also satisfies the *no new violation by deletion* formula when applied to a graph with $\mathrm{k_{max}} = k$. There are several cases for this application condition. If $k = \mathrm{nl}(c) - 2$, the constraint ends with $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$. Therefore no violations can be introduced by removing occurrences of intermediate graphs. In particular, there is no graph C_{k+3}. If $k = \mathrm{nl}(c) - 1$, every transformation $t : G \Longrightarrow H$ with $G \models c$ is direct consistency maintaining w.r.t. c if $H \models c$. This is ensured by $\mathrm{ned}_k()$ and $\mathrm{nui}_k()$. Therefore, if $k \geq \mathrm{nl}(c) - 2$, we set the application condition to true.

2. *No universally inserted (*$\mathrm{nui}_k()$*):* This application condition checks that no occurrences of universally bound graphs C_j with $1 \leq j \leq k + 2$ are inserted. It corresponds to the *no satisfied layer reduction by insertion* and *no new violations by insertion* formulas, in the sense that a rule that satisfies this application condition also satisfies the *no satisfied layer reduction by insertion* and *no new violations by insertion* formulas when applied to a graph with $\mathrm{k_{max}} = k$.

3. *No existentially destroyed (*$\mathrm{ned}_k()$*):* This application condition checks that no occurrences of existentially bound graphs C_j with $2 \leq j \leq k + 1$ are removed. It corresponds to the *no satisfied layer reduction by deletion* formula, since a rule that satisfies this application condition also satisfies the *no satisfied layer reduction by deletion* formula when applied to a graph with $\mathrm{k_{max}} = k$.

Recall that given a constraint c in UANF, each subcondition $\mathrm{sub}_k(c)$ is a condition over the graph C_k and the morphism with domain C_k is denoted by a_k.

Definition 4.1 (**consistency maintaining application condition**)**.** *Given a rule* $\rho = (\mathrm{ac}, \rho')$ *with* $\rho' = L \hookleftarrow K \hookrightarrow R$*, a constraint* c *in UANF and an odd* $-1 \leq k < \mathrm{nl}(c)$*. The* maintaining application condition *of* c *for* ρ *at layer* k *is defined as* $\mathrm{ac} \wedge \mathrm{main}_k(\rho')$ *with*

$$\mathrm{main}_k(\rho') := \mathrm{ned}_k(\rho') \wedge \mathrm{nui}_k(\rho') \wedge \mathrm{vio}_k(\rho')$$

where $\mathrm{ned}_k(\rho')$*,* $\mathrm{nui}_k(\rho')$ *and* $\mathrm{vio}_k()$ *are defined as*

1. *No violation inserted: Let* $\mathbf{P}_{C'}$ *be the set of all overlaps* P *of* L *and* C' *with* $i^P_L(L \setminus K) \cap i^P_{C'}(C' \setminus C_{k+2}) \neq \emptyset$ *for* $C' \in \mathrm{IG}(C_{k+2}, C_{k+3})$*:*

$$\mathrm{vio}_k(\rho') := \begin{cases} \mathsf{true} & \text{if } k \geq \mathrm{nl}(c) - 2 \\ \bigwedge_{C' \in \mathrm{IG}(C_{k+2}, C_{k+3})} \bigwedge_{P \in \mathbf{P}_{C'}} \neg\exists(i^P_L : L \hookrightarrow P', \mathsf{true}) & \text{otherwise} \end{cases}$$

2. *No universally inserted:*

Let $\mathbf{U}$ *be the set of all universally bound graphs* C_j *with* $1 \leq j \leq k+2$*, and* $\mathbf{P}_{C_j}$ *be the set of all overlaps* P' *of* R *and* C_j *with* $i_R^{P'}(R \setminus K) \cap i_{C_j}^{P'}(C_j \setminus C_{j-1}) \neq \emptyset$*:*

$$\mathrm{nui}_k(\rho') := \bigwedge_{C \in \mathbf{U}} \bigwedge_{P' \in \mathbf{P}_C} \mathrm{Left}(\neg\exists(i_R^{P'} : R \hookrightarrow P', \mathit{true}), \rho')$$

3. *No existentially destroyed: If* $k = -1$*, we set* $\mathrm{ned}_k(\rho') := \mathit{true}$*. Otherwise, let* $\mathbf{E}$ *be the set of all existentially bound graphs* C_j *with* $2 \leq j \leq k+1$ *and* $\mathbf{P}_{C_j}$ *be the set all overlaps* P' *of* L *and* C_j *with* $i_L^{P'}(L \setminus K) \cap i_{C_j}^{P'}(C_j \setminus C_{j-1}) \neq \emptyset$*:*

$$\mathrm{ned}_k(\rho') := \bigwedge_{C \in \mathbf{E}} \bigwedge_{P' \in \mathbf{P}_C} \neg\exists(i_L^{P'} : L \hookrightarrow P', \mathit{true})$$

We are aware that there are optimisations for $\mathrm{nui}_k()$ and $\mathrm{ned}_k(\rho')$ since not all overlaps need to be considered which [4]. But for the simplicity of our definition we did not implement these optimisations.

Example 4.1. *1. Consider the constraint* c *given in Figure 11 and the rule* `removeFeature`*. The conditions* ned_{-1}(`removeFeature`)*,* ned_1(`removeFeature`) *and* ned_3(`removeFeature`) *are also given in Figure 11;* ned_{-1}(`removeFeature`) *is equal to* *true*. ned_1(`removeFeature`) *checks that no occurrences of* C_2 *are inserted, while* ned_3(`removeFeature`) *checks that no occurrences of* C_2 *and* C_4 *are inserted. Obviously,* ned_1(`removeFeature`) *is contained in* ned_3(`removeFeature`)*. Note that there are conditions in* ned_3(`removeFeature`) *that imply each other. For example, the first condition implies the second and third. Using the optimisations according to [4], these can be removed.*

2. *Again consider constraint* c *given in Figure 11 and the rule* `assignFeature`*. The condition* nui_{-1}(`assignFeature`) *is equal to* *true* *since* `assignFeature` *cannot create any occurrences of the first universally bound graph of the constraint. The condition* nui_1(`assignFeature`) *is given in Figure 12.*

   The condition nui_3(`assignFeature`) *is equal to* nui_1(`assignFeature`) *since there are only two universally bound graphs and* nui_1(`assignFeature`) *already considers both.*

3. *Consider the constraint* c *given in Figure 13 and the rule* `addDependency`*. Then,* vio_{-1}(`addDependency`) *is given in Figure 13. The condition* vio_1(`addDependency`) *is equal to* *true* *since* $1 > \mathrm{nl}(c) - 2 = 2 - 2$*.*

Let us now show that every rule equipped with $\mathrm{main}_k()$ is a consistency-maintaining rule at layer k.

Theorem 4.1. *Given a constraint* c *in UANF. Every rule* $\rho = (\mathrm{ac} \wedge \mathrm{main}_k(\rho'), \rho')$ *where* $\rho' = L \xhookleftarrow{l} K \xhookrightarrow{r} R$ *and* $-1 \leq k < \mathrm{nl}(c)$ *is odd is a direct consistency maintaining rule w.r.t.* c *at layer* k*.*

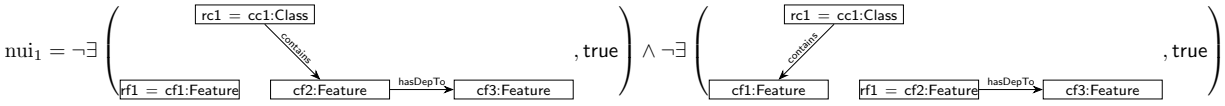

Figure 12: No universally inserted conditions for the constraint given in Figure 11 and rule `assignFeature`.

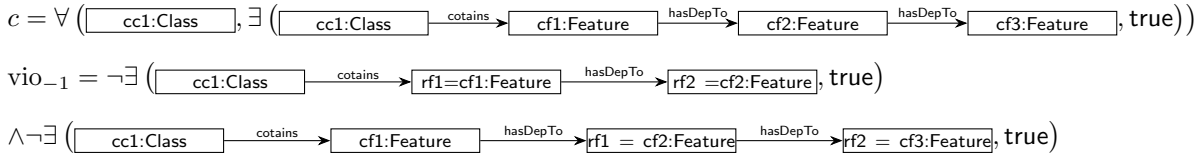

Figure 13: No violation inserted conditions for the constraint and rule `removeDependency`.

Proof. Given a graph G with $\mathrm{k}_{\max} = k$ and a transformation $t : G \Longrightarrow_{\rho,m} H$. We show that t is a direct consistency maintaining transformation w.r.t. c.

We show that t satisfies the *no new violation by deletion*, *no new violation by insertion*, *no satisfied layer reduction by insertion* and *no satisfied layer reduction by deletion* formulas.

1. Assume that t does not satisfy the *no new violation by deletion* formula. Then $\mathrm{k}_{\max} < \mathrm{nl}(c) - 1$, $e = \mathrm{sub}_{k+2}(c) \neq \mathsf{false}$ and there is a morphism $p : C_{k+2} \hookrightarrow G$ such that $p \models \mathrm{IC}_0(e, C')$, $\mathrm{tr}_t \circ p$ is total and $\mathrm{tr}_t \circ p \not\models \mathrm{IC}_0(e, C')$ for a graph $C' \in \mathrm{IG}(C_{k+2}, C_{k+3})$. Therefore, there is an overlap P of L and C' with $i^P_L(L \setminus K) \cap i^P_{C'}(C' \setminus C_{k+2}) \neq \emptyset$ such that $i^P_{C'} \circ a^r_{k+2} \models \exists(a^r_{k+2} : C_{k+2} \hookrightarrow C', \mathsf{true})$ and $m \models \exists(i^P_L : L \hookrightarrow P, \mathsf{true})$. Thus, $\mathrm{vio}_k(\rho')$ and consequently also $\mathrm{main}_k(\rho')$ cannot be satisfied.

2. Assume that t does not satisfy the *no new violation by insertion* formula. Let

$$d := \begin{cases} \mathrm{IC}_0(\mathrm{sub}_{k+2}(c), C_{k+3}) & \text{if } \mathrm{sub}_{k+2}(c) \neq \mathsf{false} \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

   Then, there is a morphism $p' : C_{k+2} \hookrightarrow H$ with $p' \not\models d$ such that no morphism $p : C_{k+2} \hookrightarrow G$ with $\mathrm{tr}_t \circ p = p'$ exists. Therefore, there is an overlap P of R and C_{k+2} with $i^P_R(R \setminus K) \cap i^P_{C_{k+2}}(C_{k+2} \setminus C_{k+1}) \neq \emptyset$ such that $m \models \mathrm{Left}(\exists(i^P_R : R \hookrightarrow P, \mathsf{true}), \rho')$. Hence, m does not satisfy $\mathrm{nui}_k(\rho')$.

3. Assume that t does not satisfy the *no satisfied layer reduction by insertion* formula. Then, there is a morphism $p : C_j \hookrightarrow H$ with $0 \leq j < k$ and C_j universally bound such that no morphism $p' : C_j \hookrightarrow G$ with $\mathrm{tr}_t \circ p' = p$ exists. Then, there is an overlap P of C_j and R with $i^P_R(R \setminus K) \cap i^P_{C_j}(C_j \setminus C_{j-1}) \neq \emptyset$ such that $m \models \mathrm{Left}(\exists(i^P_R : R \hookrightarrow P, \mathsf{true}), \rho)$. Hence, $m \not\models \mathrm{nui}_k(\rho')$.

4. Assume that t does not satisfy the *no satisfied layer reduction by deletion* formula. There is a morphism $p : C_j \hookrightarrow G$ with $j \leq k+1$ and C_j existentially bound and such that $\mathrm{tr}_t \circ p$ is not total. Then, there is an overlap P of C_j and L with $i_L^P(L \setminus K) \cap i_{C_j}^P(C_j \setminus C_{j-1}) \neq \emptyset$, such that $m \models \exists(i_L^P : L \hookrightarrow P, \mathsf{true})$. Hence, $m \not\models \mathrm{ned}_k(\rho')$.

It follows that ρ is a direct consistency-maintaining rule at layer k w.r.t. c. □

With the constructions described in Definition 4.1 we are also able to construct direct consistency-maintaining application conditions.

Theorem 4.2. *Given a constraint c in UANF. Every rule ρ equipped with the application condition*

$$\Big(\bigwedge_{\substack{-1 \leq i < \mathrm{nl}(c) \\ i \ odd}} \mathrm{vio}_i(\rho) \Big) \wedge \ \mathrm{nui}_{\mathrm{nl}(c)-1}(\rho)$$

is a direct consistency-maintaining rule w.r.t. c.

Proof. Let $\rho = R \xleftarrow{r} K \xhookrightarrow{l} L$ be a rule equipped with this application condition. We show that ρ is a consistency-maintaining rule at layer k w.r.t. c for all $-1 \leq k \leq \mathrm{nl}(c)-1$. Obviously, $\mathrm{nui}_{\mathrm{nl}(c)-1}(\rho)$ contains $\mathrm{nui}_j(\rho)$ for all $-1 < j \leq \mathrm{nl}(c)-1$. The set of intermediate graphs always contains second graphs on which this set was built, so $\mathrm{vio}_i(\rho)$ contains the condition

$$\bigwedge_{P' \in \mathbf{P}_{C_{i+3}}} \neg\exists(i_L^{P'} : L \hookrightarrow P', \mathsf{true})$$

which also checks that no occurrence of C_{i+3} is deleted. Therefore

$$\Big(\bigwedge_{\substack{-1 \leq i < \mathrm{nl}(c) \\ i \ odd}} \mathrm{vio}_i(\rho) \Big)$$

must contain $\mathrm{ned}_{\mathrm{nl}(c)-1}(\rho)$ and therefore it contains $\mathrm{ned}_j(\rho)$ for all $-1 \leq j \leq \mathrm{nl}(c)-1$. So we can rewrite this application condition into the equivalent condition

$$\Big(\bigwedge_{\substack{-1 \leq i < \mathrm{nl}(c) \\ i \ odd}} \mathrm{vio}_i(\rho) \wedge \mathrm{nui}_i(\rho) \wedge \mathrm{ned}_i(\rho) \Big) = \Big(\bigwedge_{\substack{-1 \leq i < \mathrm{nl}(c) \\ i \ odd}} \mathrm{main}_i(\rho) \Big).$$

It follows that ρ is a direct consistency-maintaining rule at layer k for all $-1 \leq k < \mathrm{nl}(c)$. Since $-1 \leq \mathrm{k_{max}} < \mathrm{nl}(c)$ for each graph G, all transformations of ρ are direct consistency-maintaining w.r.t. c. Hence, ρ is a consistency-maintaining rule w.r.t. c. □

The *extended overlap* will be a useful tool for our consistency-increasing application conditions. Intuitively, given an overlap and a morphism a, the overlap is extended such that one overlap morphism satisfies $\exists(a, \mathsf{true})$.

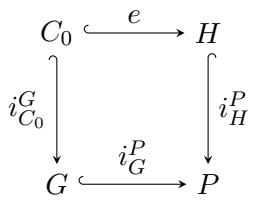

Figure 14: Diagram for the alternative definition of extended overlaps.

Definition 4.2 (extended overlaps). *Given an overlap* $(G, i^G_{C_0}, i^G_{C_1})$ *of* C_0 *and* C_1 *and a morphism* $e : C_0 \hookrightarrow H$*. The set of* extended overlaps *of* G *at* $i^G_{C_0}$ *with* e*, denoted by* $\mathrm{eol}(G, i^G_{C_0}, e)$ *is defined as*

$$\mathrm{eol}(G, i^G_{C_0}, e) := \{P \in \mathrm{ol}(G, H) \mid i^P_G \circ i^G_{C_0} \models \exists(e : C_0 \hookrightarrow H, \mathit{true})\}.$$

In other words, $\mathrm{eol}(G, i^G_{C_0}, e)$ is the set of all overlaps of G and H such that the square in Figure 14 is commutative, i.e. $i^P_H \circ e = i^P_G \circ i^G_{C_0}$. Using extended overlaps, we will be able to check whether a violation has been removed. Let us now consider consistency-increasing application conditions. For these we will use the maintaining application conditions described above. All that remains is to ensure that at least one violation is removed. To do this, we must first check that there is a violation in the match. This means that the match and a violation are overlapping. Finally, we need to check that this violation is removed.

Again, the increasing application condition has the odd parameter $-1 \leq k < \mathrm{nl}(c)-1$, which specifies which constraint graphs are to be considered. Note that k must not be $\mathrm{nl}(c) - 1$, since all graphs with $\mathrm{k_{max}} = \mathrm{nl}(c)$ satisfy the constraint. Therefore for these graphs there can be no consistency increasing transformations. It also has a second parameter C' which is an intermediate graph of C_{k+2} and C_{k+3} if c contains a graph C_{k+3}, i.e. $k < \mathrm{nl}(c) - 2$ and C' is set to C_{k+2} otherwise. Again, a rule equipped with this application condition is a consistency-increasing rule at layer k. It consists of the following parts:

1. The maintaining application condition ($\mathrm{main}_k()$): As already discussed, this application condition ensures that a rule equipped with this application condition is a consistency-maintaining rule at layer k.

2. *Violation exists* (exv()): This condition checks that there is a violation at the match, i.e. there is an occurrence p of C_{k+2} with $m(L) \cap p(C_{k+2}) \neq \emptyset$ not satisfying $\exists(C', \mathsf{true})$. If $k = \mathrm{nl}(c) - 2$, then c ends with $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$ and thus there is no graph C_{k+3} in c. In this case it is sufficient to check only that $m(L) \cap p(C_{k+2}) \neq \emptyset$.

3. *Violation removed* (remv()): This condition checks that the violation is removed. This can be done in several ways, either by deleting the occurrence p or by inserting elements such that $p \models \exists(C', \mathsf{true})$. This leads to case discrimination. The first case is easy to check, if $m(L \setminus K) \cap p(C_{k+2} \setminus C_{k+1}) \neq \emptyset$, p is removed and this

condition can be set to true. Otherwise, we need to check that the violation has been removed by an additional condition that checks whether p satisfies $\exists(C', \mathsf{true})$ after the transformation. The last case is the special case where the constraint ends with $\forall(C_{\mathrm{nl}(c)}, \mathsf{true})$ and $k = \mathrm{nl}(c) - 2$. Then there is only one way to remove a violation, by removing the occurrence p. So the condition is set to true if $m(L \setminus K) \cap p(C_{k+2} \setminus C_{k+1}) \neq \emptyset$ and to false otherwise.

Definition 4.3 (consistency increasing application condition). *Given a rule* $\rho = (\mathrm{ac}, \rho')$ *with* $\rho' = L \xleftarrow{l} K \xrightarrow{r} R$ *and a constraint* c *in UANF. Let* $0 \leq k < \mathrm{nl}(c) - 1$ *be odd and* $C' = C_{k+2}$ *if* $k = \mathrm{nl}(c) - 2$ *and* $C' \in \mathrm{IG}(C_{k+2}, C_{k+3})$ *otherwise. The* increasing application condition of c for ρ at layer k with C' *is defined as*

$$\mathrm{incr}_k(C', \rho) := \mathrm{ac} \wedge \mathrm{main}_k(\rho') \wedge \Big(\bigvee_{P \in \mathrm{ol}(L, C_{k+2})} \mathrm{exv}(P, C') \wedge \mathrm{remv}(P, C') \Big) \tag{4.1}$$

with

1. *Violation exists: Let* $a^r_{k+2} : C_{k+2} \hookrightarrow C'$ *be the restricted morphism of* a_{k+2} *and* i^P_L *and* i^Q_P *be overlap morphisms of* P *and* Q*, respectively:*

$$\mathrm{exv}(P, C') := \begin{cases} \exists(i^P_L : L \hookrightarrow P, \mathsf{true}) & \text{if } \mathrm{sub}_{k+2}(c) = \mathsf{false} \\ \exists(i^P_L : L \hookrightarrow P, \bigwedge_{Q \in \mathrm{eol}(P, i^P_{C_{k+2}}, a^r_{k+2})} \neg\exists(i^Q_P : P \hookrightarrow Q, \mathsf{true})) & \text{otherwise} \end{cases}$$

2. *Violation removed:*

   (a) *If* $i^P_L(L \setminus K) \cap i^P_{C_{k+2}}(C_{k+2} \setminus C_{k+1}) \neq \emptyset$, $\mathrm{remv}(P, C') :=$ *true.*

   (b) *If* $\mathrm{sub}_{k+2}(c) =$ *false, i.e.* $k = \mathrm{nl}(c) - 2$,

$$\mathrm{remv}(P, C') := \begin{cases} \mathsf{true} & \text{if } i^P_L(L \setminus K) \cap i^P_{C_{k+2}}(C_{k+2} \setminus C_{k+1}) \neq \emptyset \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

   (c) *Otherwise, let* P' *be the overlap of* R *and* C_{k+2} *such that there is a transformation* $P' \Longrightarrow_{\rho^{-1}, i^{P'}_R} P$*. If this overlap or transformation does not exist, we set* $\mathrm{remv}(P, C') :=$ *false and otherwise*

$$\mathrm{remv}(P, C') := \mathrm{Left}(\forall(i^Q_R : R \hookrightarrow P', \bigvee_{Q \in \mathrm{eol}(P', i^{P'}_{C_{k+2}}, a^r_{k+2})} \exists(i^Q_{P'} : P' \hookrightarrow Q, \mathsf{true})), \rho).$$

Example 4.2. *1. Consider constraint* c_1 *given in Figure 5 and the rule* `assignFeature`*. There is only one overlap* P *of* L *and* C^1_1 *which is shown in Figure 15. The* $\mathrm{exv}(P, C')$ *and* $\mathrm{exv}(P, C')$ *parts of* $\mathrm{incr}_{-1}(C', \texttt{assignFeature})$ *with* $C' = C^2_2$ *and* $C' = C^1_2$*, respectively, are also given in Figure 15.*

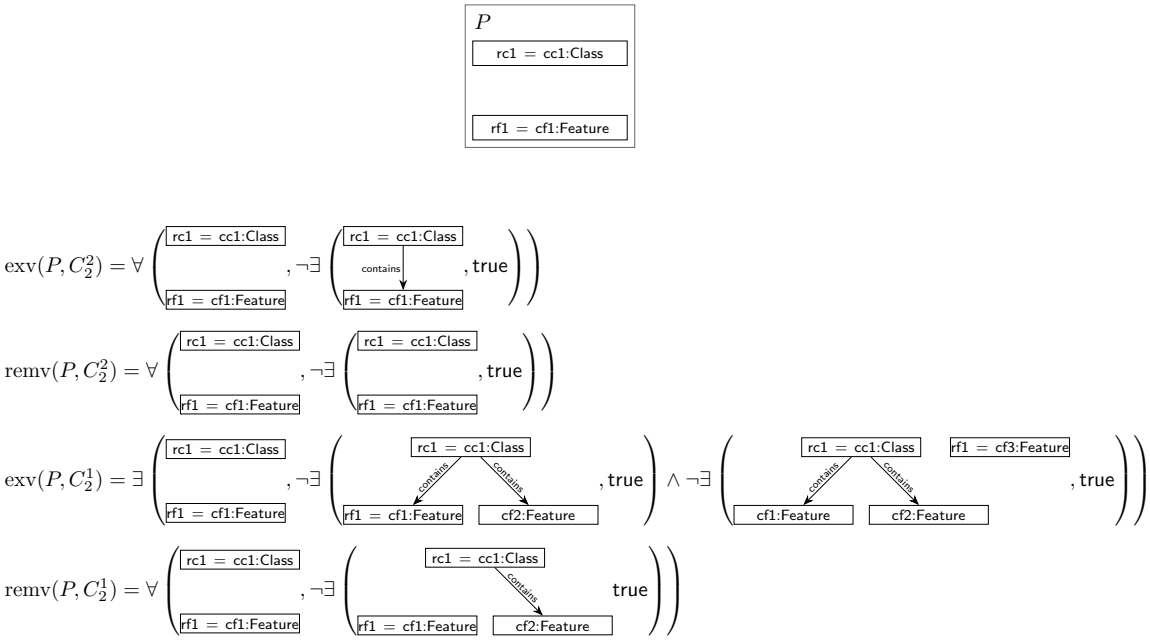

Figure 15: Examples for $\mathrm{exv}(P, C_2^2), \mathrm{exv}(P, C_2^1), \mathrm{remv}(P, C_2^2)$ and $\mathrm{remv}(P, C_2^1)$ using the rule `assignFeature` and constraint c_1 given in Figure 5.

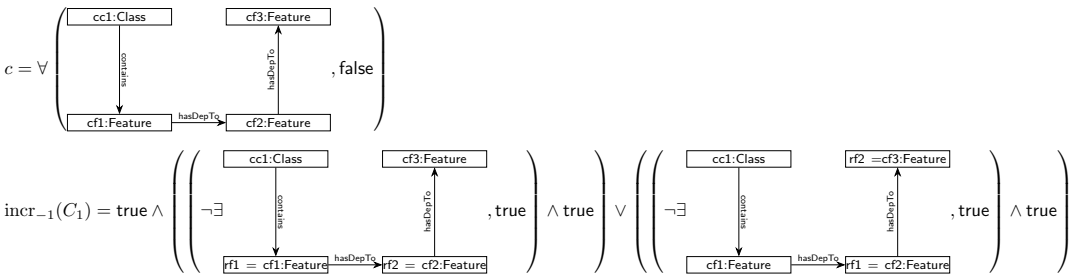

Figure 16: Example of $\mathrm{incr}_{-1}(C_1, \rho)$ using this constraint and the rule `removeDependency`.

2. *Consider the constraint and rule given in Figure 16. The consistency-increasing application condition of this rule and at layer -1 with C_1 of the constraint is also given in this Figure.*

Let us now show that a rule equipped with the consistency-increasing transformation condition at layer k is indeed a consistency-increasing rule at layer k.

Theorem 4.3. *Given a constraint c in UANF. Every rule $\rho = (\mathrm{ac} \wedge \mathrm{incr}_k(C', \rho'), \rho')$ with $-1 \leq k < \mathrm{nl}(c) - 1$ odd and $C' \in \mathrm{IG}(C_{k+2}, C_{k+3})$ if $k < \mathrm{nl}(c) - 2$ and $C' = C_{k+2}$ otherwise is a direct consistency-increasing rule w.r.t. c at layer k.*

Proof. Let a transformation $t : G \Longrightarrow_\rho H$ with $\mathrm{k}_{\max}(c, G) = k$ be given. Since $\mathrm{incr}_k(C', \rho)$ contains $\mathrm{main}_k(\rho)$, t is a direct consistency maintaining transformation at layer k according to Theorem 4.1. It remains to show that t satisfies the general or special increasing formula respectively.

1. If $k = \mathrm{nl}(c) - 2$, i.e. c ends with a condition of the form $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$, assume that t does not satisfy the special increasing formula. Then there is no morphism $p : C_{k+2} \hookrightarrow G$ such that $\mathrm{tr}_t \circ p$ is not total. So there is no overlap P of L and C_{k+2} with $i^P_L(L \setminus K) \cap i^P_{C_{k+2}}(C_{k+2} \setminus C_{k+1}) \neq \emptyset$ such that $m \models \exists(i^P_L : L \hookrightarrow P, \mathsf{true})$. Since, $k = \mathrm{nl}(c) - 2$ it follows that either $\mathrm{remv}(P', C') = \mathsf{false}$ or $m \not\models \mathrm{exv}(P', C')$ for all $P \in \mathrm{ol}(L, P)$. Therefore $m \not\models \mathrm{incr}_k(C, \rho)$, this is a contradiction.

2. Otherwise let $P \in \mathrm{ol}(L, C_{k+2})$. We show that $m \models \mathrm{exv}(P, C') \wedge \mathrm{remv}(P, C')$ implies that t satisfies the general increasing formula. If $m \models \mathrm{exv}(P, C')$, there is a morphism $p : P \hookrightarrow G$ with $m = p \circ i^P_L$ and $p \models \neg\exists(i^Q_P : P \hookrightarrow Q, \mathsf{true})$ for all $Q \in \mathrm{eol}(P, i^P_{C_{k+2}}, a^r_{k+2})$. Therefore, $q := p \circ i^P_{C_{k+2}} \not\models \exists(a^r_{k+2} : C_{k+2} \hookrightarrow C', \mathsf{true})$.

   For $\mathrm{remv}(P, C')$ there are two cases. Either $\mathrm{remv}(P, C') = \mathsf{true}$ or $\mathrm{remv}(P, C')$ is the condition described in Definition 4.3 and $m \models \mathrm{remv}(P, C')$. In the first case, if $\mathrm{remv}(P, C') = \mathsf{true}$ it follows that $i^P_L(L \setminus K) \cap i^P_{C_{k+2}}(C_{k+2} \setminus C_{k+1}) \neq \emptyset$. Therefore $\mathrm{tr}_t \circ q$ is not total and the general increasing formula is satisfied.

   Otherwise, if $\mathrm{remv}(P, C')$ is the condition described in Definition 4.3 and $m \models \mathrm{remv}(P, C')$, $\mathrm{tr}_t \circ q$ is total and there is a morphism $p' : P' \hookrightarrow H$ with $\mathrm{tr}_t \circ q = p' \circ i^{P'}_{C_{k+2}}$. Since $m \models \mathrm{remv}(P, C')$, all morphisms $p'' : P' \hookrightarrow H$ with $n = p'' \circ i^Q_R$ satisfy $\bigvee_{Q \in \mathrm{eol}(P', i^{P'}_{C_{k+2}}, a^r_{k+2})} \exists(i^Q_{P'} : P' \hookrightarrow Q, \mathsf{true})$. It follows that $p'' \circ i^P_{C_{k+2}} \models \exists(C', \mathsf{true})$ and in particular that $\mathrm{tr}_t \circ q = p' \circ i^{P'}_{C_{k+2}} \models \exists(C', \mathsf{true})$. Therefore, the general increasing formula is satisfied.

In summary, ρ is a direct consistency increasing rule at layer k. □

Note that $\mathrm{incr}_k(C', \rho)$ is only evaluated to true if an occurrence $p : C_{k+2} \hookrightarrow G$ that does not satisfy $\exists(C', \mathsf{true})$ is either removed or satisfies $\exists(C', \mathsf{true})$ in the derived graph. For all smaller improvements, i.e. a similar improvement for a subgraph

$C'' \in \mathrm{IG}(C_{k+2}, C')$ of C', $\mathrm{incr}_k(C', \rho)$ would be evaluated to false. For any larger improvements, i.e. the same improvement for a supergraph $C'' \in \mathrm{IG}(C', C_{k+3})$ of C', $\mathrm{incr}_k(C', \rho)$ will also be evaluated to false if the repaired occurrence of C_{k+2} satisfies $\exists(C', \mathsf{true})$. In both cases, the application condition would prohibit the transformation, even though it would be direct consistency-increasing. To solve this problem, several application conditions could be combined by

$$\bigvee_{C' \in \mathrm{IG}(C_{k+2}, C_{k+3})} \mathrm{incr}_k(C', \rho).$$

This application condition will be evaluated to true if the cases described above occur, with the drawback that this results in a huge condition, even if duplicate conditions are removed. At least all duplicates of main() can be removed, since they are identical for each $\mathrm{incr}_k(C', \rho)$ and only need to be constructed once.

In general, these application conditions are a compromise between condition size and restrictiveness. They are very restrictive because they do not allow deletions of occurrences of existentially bound graphs and insertions of universally bound graphs. For example, any of these application conditions with the rule `moveFeature` and the constraint c_1 will be equivalent to false; the maintaining part of the condition will always be evaluated to false, since `moveFeature` always removes occurrences of the existentially bound graph C_2^1. Changing the conditions constructed by main() to check whether two nodes of type `Feature` are connected to a node of type `Class` will give application conditions that can be satisfied with `moveFeature`. However, for a similar rule moving two nodes of type `Feature`, this newly constructed main() would still be evaluated to false. So this would only lead to a slight decrease in restrictiveness.

The conditions constructed by ned() and nui() could be modified in a similar way. For ned() and an occurrence p of the universally bound graph C_j, by checking whether there are two occurrences p_1, p_2 of C_j with $p = p_1 \circ a_j = p_2 \circ a_j$, and for nui(), by checking whether an introduced occurrence p of C_j satisfies $\exists(C_{j+1}, \mathsf{true})$. As above, this only leads to a small decrease of restrictiveness. Also, the consistency increasing application condition becomes more and more restrictive as k increases, since the number of conditions and in particular the number of negative application conditions also increases.

4.2 Basic Consistency-increasing and Consistency-maintaining Rules

The construction of the application conditions introduced in the previous section, as well as the constructed application conditions themselves, are very complex. For a certain set of rules, which we will call *basic consistency-increasing rules*, we are able to construct application conditions with the same property, namely that a rule equipped with this application condition is consistency-increasing at a layer k, in a less complex way. The main idea is that these rules (a) are not able to delete occurrences of existentially bound graphs or insert occurrences of universally bound graphs and (b) are able to increase consistency at a certain layer. That is, given a basic increasing rule ρ, there exists a

transformation $t : G \Longrightarrow_\rho H$ such that t is a consistency increasing transformation with respect to a constraint c.

To ensure that (a) is satisfied, we first introduce *basic consistency-maintaining rules at layer*, which means that, given a constraint, a plain rule is not able to delete existentially bound and insert universally bound graphs up to a certain layer. For the definition, we use the notion of direct consistency maintaining rules at layer. The set of basic consistency-maintaining rules up to layer is actually a subset of the set of consistency-maintaining rules at layer, since these rules must be plain rules, whereas consistency-maintaining rules at layer are allowed to have application conditions, i.e. $\mathrm{main}(\cdot, \cdot)$.

Definition 4.4 (**basic consistency maintaining rule at layer**). *Given a plain rule ρ and a constraint c in UANF. Then, ρ is called* basic consistency maintaining rule at layer $-1 \leq k < \mathrm{nl}(c)$ w.r.t. c *if it is a direct consistency-maintaining rule at layer k w.r.t. c.*

Example 4.3. *Consider the rules* `moveFeature`, `assignFeature` *and* `addDependency` *given in Figure 10 and constraints c_1 and c_2 given in Figure 5. The rule* `assignFeature` *is a basic consistency maintaining rule at layer 1 w.r.t. c_1, whereas* `moveFeature` *is not a basic consistency maintaining rule w.r.t. c_1. The rule* `addDependency` *is a basic consistency-maintaining rule at layer -1 w.r.t. c_2, but is not a basic consistency-maintaining rule at layer 1 w.r.t. c_2 since it can insert occurrences of C_3^2.*

Since there are infinitely many transformations via a plain rule ρ, it is impossible to check whether ρ is a basic consistency maintaining rule at a layer based on the definition above. Therefore, we present a characterisation of basic consistency-maintaining rules that relies only on ρ itself.

First, let us assume that ρ is able to create occurrences of a universally bound graph C_j. This is possible if (a) ρ inserts an edge of $C_j \setminus C_{j-1}$ connecting pre-existing nodes of C_j, since it is unclear whether this would create a new occurrence of C_j, or (b) if ρ inserts a node v of C_j, so that all edges $e \in E_{C_j}$ with $\mathrm{src}(e) = v$ or $\mathrm{tar}(e) = v$ are also inserted. If at least one of these edges is not inserted, it is guaranteed that this insertion will not create an occurrence of C_j, since v is only connected to edges that have also been inserted by ρ.

Second, suppose ρ is able to delete occurrences of an existentially bound graph C_j. This is possible if (a) ρ deletes an edge of $C_j \setminus C_{j-1}$ or (b) ρ deletes a node v of $C_j \setminus C_{j-1}$ such that all edges $e \in E_{C_j}$ with $\mathrm{src}(e) = v$ or $\mathrm{tar}(e) = v$ are also deleted. If ρ deletes a node c of $C_j \setminus C_{j-1}$ without all its connected edges in C_j, there is no transformation via ρ such that an occurrence of C_j is deleted by deleting that node, since the dangling edge condition would not be satisfied. A rule that satisfies these properties does not reduce the largest satisfied layer.

We also need to ensure that the number of violations is not increased. To do this, we have to check that ρ is not able to insert occurrences of the corresponding universally bound graph, as described above, and that ρ is not able to remove occurrences of any

intermediate graph. This is only ensured if ρ does not remove any elements of $C_{k+1} \setminus C_k$ when the set of intermediate graphs is given by $\mathrm{IG}(C_k, C_{k+1})$.

To check that a plain rule satisfies these properties, we use the dangling edge condition, or in other words, we check that the rule does not apply to certain overlaps of L and an existentially bound graph, or that the inverse rule does not apply to certain overlaps of R and a universally bound graph.

Lemma 4.5. *Given a plain rule $\rho = L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R$ and a constraint c in UANF. Let $-1 \leq k < \mathrm{nl}(c)$ be odd, then ρ is a basic consistency-maintaining rule up to layer k w.r.t. c if 1 and 2 hold for all k, and 3 holds if $k < \mathrm{nl}(c) - 2$.*

1. *For each existentially bound graph C_j with $2 \leq j \leq k+1$ and each overlap $P \in \mathrm{ol}(L, C_j)$ with $i_L^P(L \setminus K) \cap i_{C_j}^P(C_j \setminus C_{j-1}) \neq \emptyset$, the rule ρ is not applicable at match i_L^P.*

2. *For each universally bound graph C_j with $1 \leq j \leq k+2$ and each overlap $P \in \mathrm{ol}(R, C_j)$ with $i_R^P(R \setminus K) \cap i_{C_j}^P(C_j) \neq \emptyset$, the rule ρ^{-1} is not applicable at match i_r^P.*

3. *For all graphs $P \in \mathrm{ol}(L, C_{k+3})$ it holds that*

$$i_L^P(L \setminus K) \cap i_{C_{k+3}}^P(C_{k+3} \setminus C_{k+2}) = \emptyset.$$

Proof. Let $\rho = L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R$ be a rule that satisfies the characterisations listed in Lemma 4.5 with $-1 \leq k < \mathrm{nl}(c)$ odd. Suppose ρ is not a direct consistency-maintaining rule up to layer k w.r.t. c. Then, there is a transformation $t : G \Longrightarrow_{\rho,m} H$ with $\mathrm{k_{max}}(c, G) = k$ such that t is not direct consistency-maintaining w.r.t. c. Therefore, either the no new violation by deletion, no new violation by insertion, no satisfied layer reduction by insertion or no satisfied layer reduction by deletion formula is not satisfied.

1. If the no new violation by deletion formula is not satisfied, then $k < \mathrm{nl}(c) - 2$. There is an occurrence $p : C_{k+2} \hookrightarrow G$ such that $p \models \mathrm{IC}_0(\mathrm{sub}_{k+2}(c), C')$ and $\mathrm{tr}_t \circ p \not\models \mathrm{IC}_0(\mathrm{sub}_{\mathrm{k_{max}}+2}(c), C')$ with $C' \in \mathrm{IG}(C_{k+2}, C_{k+3})$. So an overlap $P \in \mathrm{ol}(L, C_{k+3})$ with $i_L^P(L \setminus K) \cap i_{C_{k+3}}^P(C_{k+3} \setminus C_{k+2}) = \emptyset$ must exist. This is a contradiction.

2. If the no new violation by insertion formula is not satisfied, there is an occurrence $p : C_{k+2} \hookrightarrow H$ such that no morphism $q : C_{k+2} \hookrightarrow G$ with $p = \mathrm{tr}_t \circ q$ exists and $p \not\models \mathsf{false}$ if $\mathrm{sub}_{k+2}(c) = \mathsf{false}$ and $p \not\models \mathrm{IC}_0(\mathrm{sub}_{k+2}(c), C_{j+3})$ otherwise. So there is an overlap $P \in \mathrm{ol}(R, C_{k+2})$ with $i_R^P(R \setminus K) \cap i_{C_{k+2}}^P(C_{k+2}) \neq \emptyset$ such that ρ^{-1} is applicable at match i_R^P. This is a contradiction.

3. If the no satisfied layer reduction by insertion formula is not satisfied, there is an occurrence $p : C_j \hookrightarrow H$ of an universally bound graph C_j with $1 \leq j \leq k+2$ such that no morphism $q : C_j \hookrightarrow G$ with $\mathrm{tr}_t \circ q = p$ exists. So there is an overlap $P \in \mathrm{ol}(R, C_j)$ with $i_R^P(R \setminus K) \cap i_{C_j}^P(C_j) \neq \emptyset$ such that ρ^{-1} is applicable at match i_R^P. This is a contradiction.

4. If the no satisfied layer reduction by deletion formula is not satisfied, there is an occurrence $p : C_j \hookrightarrow H$ of an existentially bound graph C_j with $2 \leq j \leq k+1$ such that $\mathrm{tr}_t \circ p$ is not total. So there is an overlap $P \in \mathrm{ol}(L, C_j)$ with $i_L^P(L \setminus K) \cap i_{C_j}^P(C_j \setminus C_{j-1}) \neq \emptyset$ such that the rule ρ is applicable at match i_L^P. This is a contradiction.

In summary, ρ is a basic consistency-maintaining rule up to layer k. □

Now we are ready to introduce *basic increasing rules at layer k*, where k is odd. The set of basic increasing rules is a subset of the set of consistency-maintaining rules at layer k, which ensures that the largest satisfied layer as well as the number of violations will not increase. In addition, the left-hand side of this rule contains an occurrence p of the universally bound graph C_{k+2}, such that either this occurrence is removed, i.e. elements of $C_{k+2} \setminus C_{k+1}$ are deleted, or an intermediate graph $C \in \mathrm{IG}(C_{k+2}, C_{k+3})$ is inserted. Of course, this second case only occurs if $k < \mathrm{nl}(c) - 2$, where c is the corresponding constraint. This property has the advantage that the application conditions for basic increasing rules are less complex and smaller, since it can be determined exactly how this rule removes a violation, and therefore no overlaps need to be considered.

This, at first sight, seems to be a restriction of the set of basic increasing rules, but the context of any rule ρ that satisfies all the properties of a basic increasing rule except that C_{k+2} is a subgraph of the left-hand side can be extended so that this new rule ρ' is a basic increasing rule and the semantic of ρ' is a subset of the semantic of ρ. A characterisation for these derived rules will be presented later. In particular, derived rules are all amalgamated rules [5] of $L' \overset{l'}{\hookleftarrow} K \overset{r'}{\hookrightarrow} R$, $C_{k+2} \overset{\mathrm{id}}{\hookleftarrow} C_{k+2} \overset{id}{\hookrightarrow} C_{k+2}$ and $\rho = L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R$ such that injective morphisms $l_1 : L' \hookrightarrow C_{k+2}$, $l_2 : L' \hookrightarrow L$, $k_1 : K' \hookrightarrow C_{k+2}$, $k_2 : K' \hookrightarrow K$, $r_1 : R' \hookrightarrow C_{k+2}$ and $r_1 : R' \hookrightarrow R$ exists.

Basic increasing rules at layer k are called *deleting basic increasing rules* when p is removed and *inserting basic increasing rules* when an intermediate graph is inserted. For our repair process, we will introduce the restriction that deleting basic increasing rules may only delete edges but not nodes of C_{k+2}, since otherwise it is not possible to decide, given a rule set and a constraint, whether this rule set is able to repair an arbitrary graph based only on deleting basic increasing rules. For example, consider a rule that deletes a node from C_{k+2}. Then it is unknown whether this node is connected to nodes that do not belong to C_{k+2}, and it is unclear whether all occurrences of C_{k+2} could be destroyed by ρ, since the dangling edge condition might not be satisfied.

Definition 4.6 (**basic increasing rule**). *Given a constraint c in UANF and a direct consistency-maintaining rule $\rho = (\mathrm{ac}, L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$ at layer $-1 \leq k \leq \mathrm{nl}(c) - 2$, where k is odd. Then, ρ is a* basic increasing rule w.r.t *c* at layer *k if a morphism $p : C_{k+2} \hookrightarrow L$, called the* increasing morphism*, exists such that either 1 or 2 holds.*

1. Universally deleting*: $r \circ l^{-1} \circ p$ is not total. Then, ρ is called a* deleting basic increasing rule*.*

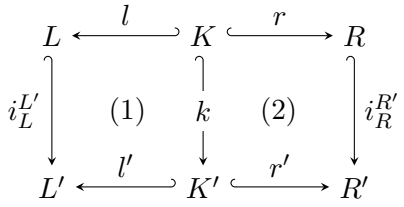

Figure 17: Pushout diagram for Lemma 4.8

2. Intermediate inserting: *If* $k < \mathrm{nl}(c) - 2$, *there is an intermediate graph* $C' \in \mathrm{IG}(C_{k+2}, C_{k+3})$ *such that* $p \not\models \exists(a^r_{k+2} : C_{k+2} \hookrightarrow C', \mathsf{true})$, $r \circ l^{-1} \circ p$ *is total and* $r \circ l^{-1} \circ p \models \exists(a^r_{k+2} : C_{k+2} \hookrightarrow C', \mathsf{true})$. *Then,* ρ *is called a* inserting basic increasing rule with C.

Example 4.4. *Consider the rule* `assignFeature` *given in Figure 10 and constraint* c_1 *given in Figure 5. Then,* `assignFeature` *is an inserting basic rule with* $C^2_2 \in \mathrm{IG}(C^1_1, C^1_2)$ *w.r.t.* c_1 *but is not an inserting basic rule with respect to the constraint* $\forall(C^2_2, \exists(C^1_2, \mathsf{true}))$ *since the left-hand side of* `assignFeature` *does not contain an occurrence of* C^2_2.

As mentioned above, given a direct consistency-maintaining rule ρ, we can derive basic increasing rules that are applicable when ρ is applicable by extending the context of that rule so that it contains an occurrence of the graph C_{k+2}.

Definition 4.7 (derived rules). *Given a constraint* c *in UANF and a rule* $\rho = (\mathrm{ac}, L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$. *The set of* derived rules from ρ *at layer* $-1 \leq k \leq \mathrm{nl}(c) - 2$, *where* k *is odd, contains rules characterised as follows: Let*

$$\mathbf{G} := \begin{cases} \{C_{k+2}\} & \text{if } k = \mathrm{nl}(c) - 2 \text{ is existentially bound} \\ \mathrm{IG}(C_{k+2}, C_{k+3}) & \text{otherwise.} \end{cases}$$

For $P \in \mathbf{G}$ *and* $(L', i_L^{L'}, i_P^{L'}) \in \mathrm{ol}(L, P)$: *If the diagram shown in Figure 17 is a transformation, i.e. (1) and (2) are pushouts, and*

$$L' \overset{l'}{\hookleftarrow} K' \overset{r'}{\hookrightarrow} R' \text{ is universally deleting or intermediate inserting}$$

the rule

$$\rho' = (\mathrm{Shift}(\mathrm{ac}, i_L^{L'}), L' \overset{l'}{\hookleftarrow} K' \overset{r'}{\hookrightarrow} R')$$

is a derived rule of ρ *at layer* k.

Example 4.5. *Consider the rule* `assignFeature` *given in Figure 10 and constraint* c_1 *given in Figure 5. The set of derived rules from* ρ *at layer* 1 *is given in Figure 18.*

The following lemma shows that a derived rule ρ' of ρ is only applicable at a graph G if ρ is also applicable at G and the derived graphs of these transformations are identical.

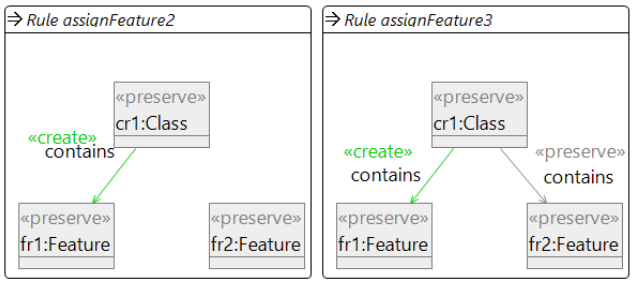

Figure 18: Derived rules of `assignFeature` and c_1.

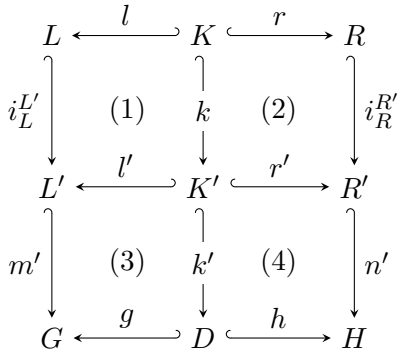

Figure 19: Pushout diagram for the construction of basic increasing rules.

Lemma 4.8. *Given a graph G and rules $\rho = L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R$ and $\rho' = L' \overset{l'}{\hookleftarrow} K' \overset{r'}{\hookrightarrow} R'$ such that ρ' is a derived rule of ρ. Then, ρ' is only applicable at match $m' : L' \hookrightarrow G$ if ρ is applicable at match $m' \circ i_L^{L'}$ where $i_L^{L'}$ is the morphism shown in Figure 17.*

Proof. Consider the transformation composed of the pushouts (3) and (4) given in Figure 19. Since ρ' is a derived rule of ρ, the squares (1) and (2) are pushouts. Therefore, the squares (1) + (3) and (2) + (4) are also pushouts [7]. It follows that ρ is applicable at match $m' \circ i_L^{L'}$. □

Therefore extending a rule set $\mathcal{R}$ by the set of all derived rules for each rule of $\mathcal{R}$ does not extend the semantic of $\mathcal{R}$. The main idea of the concept of derived rules is to extend a given set of rules by as many basic increasing rules as possible without extending its semantic.

In transformations via a rule ρ such that the match intersects an occurrence of a universally bound graph C_{k+2}, ρ can be replaced by a derived rule of ρ at layer k.

Lemma 4.9. *Given a constraint c in UANF and a rule $\rho = (\mathrm{ac}, L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$. Then, for each transformation*

$$t : G \Longrightarrow_{\rho,m} H$$

such that an occurrence $p : C_{k+2} \hookrightarrow G$ *of a universally bound graph* C_{k+2} *with* $p(C_{k+2}) \cap m(L) \neq \emptyset$ *exists, there is a transformation*

$$t' : G \Longrightarrow_{\rho',m'} H$$

where ρ' *is a derived rule of* ρ *at layer* k*.*

Proof. Since $p(C_{k+2}) \cap m(L) \neq \emptyset$ there is an overlap $P \in \mathrm{ol}(C_{k+2}, L)$ such that there exists a morphism $q : P \hookrightarrow G$ with $m = q \circ i_L^P$ and $p = q \circ i_{C_{k+2}}^P$. Since t exists, there is a derived rule $\rho' = (\mathrm{ac}', L' \overset{l'}{\hookleftarrow} K' \overset{r'}{\hookrightarrow} R)$ of ρ at layer k, where $L' = P$. We set $m' = q$, since $m = m' \circ i_L^{L'} \models \mathrm{ac}$, it follows that $m' = \mathrm{ac}'$, and since ρ removes and inserts the same elements as ρ, there is the transformation $t' : G \Longrightarrow_{\rho',m'} H$. □

This allows us to replace consistency-increasing transformations via a direct consistency-maintaining rule ρ at layer k by a rule derived from ρ at layer k, i.e. a basic increasing rule at layer k.

4.3 Application Conditions for Basic Rules

Let us now introduce the application conditions for basic increasing rules. Since basic rules are consistency-maintaining at a certain layer k it suffices to check whether $m \circ i \models \exists(C_{k+3}, \mathsf{true})$ if ρ is a deleting rule, and whether $m \circ i \models \exists(C', \mathsf{true})$ if ρ is an inserting rule, where m is the match of the transformation and i is the increasing morphism of ρ.

Definition 4.10 (application conditions for basic increasing rules). *Given a constraint* c *in UANF and a basic increasing rule* $\rho = (\mathrm{ac}, L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$ *w.r.t.* c *at layer* $-1 \leq k \leq \mathrm{nl}(c) - 2$*, where* k *is odd. The* basic application condition of ρ w.r.t. c at layer $-1 \leq j \leq \mathrm{nl}(c) - 2$ *is given by*

$$\mathrm{ac}' = \mathrm{ac} \wedge \mathrm{basic}_j(\rho)$$

with

$$\mathrm{basic}_j(\rho) := \begin{cases} \bigwedge_{P \in \mathrm{eol}(L,a,i)} \neg\exists(i_L^P : L \hookrightarrow P, \mathsf{true}) & \text{if } j = k \text{ and } k < \mathrm{nl}(c) - 2 \\ \mathsf{true} & \text{if } k = \mathrm{nl}(c) - 2 \\ \mathsf{false} & \text{otherwise} \end{cases}$$

where $a = a_{k+2}$*, if* ρ *is a deleting rule,* $a = a_{k+2}^r : C_{k+2} \hookrightarrow C'$ *if* ρ *is an inserting rule with* C' *and* i *is the increasing morphism of* ρ*.*

These application conditions are much easier to construct and smaller than those constructed by Definition 4.3. Note that in the case of an inserting rule ρ which inserts an intermediate graph C, the application condition only checks whether the increasing morphism does not satisfy $\exists(C, \mathsf{true})$. But an application of this rule could also lead to a consistency increasing transformation w.r.t. c if the increasing morphism satisfies $\exists(C, \mathsf{true})$ and another intermediate graph C' is inserted. To check this, conditions

similar to those constructed via Definition 4.3 must be constructed. At first sight this seems like a restriction, but via the notion of derived rules we are able to dissolve this restriction, since the set of derived rules of ρ will contain an inserting basic increasing rule with C', so that this rule, equipped with the corresponding basic application condition, can be used to perform this consistency-increasing transformation. For example, consider the rule `assignFeature` and constraint c_1 given in Figure 5, there is a consistency increasing transformation $t : C_2^2 \Longrightarrow_{\texttt{assignFeature},m} C_2^1$ such that $m \not\models \text{basic}_{-1}(\texttt{assignFeature})$, but there is also a transformation $t : C_2^2 \Longrightarrow_{\texttt{assignFeature3},m'} C_2^1$ with $m' \models \text{basic}_{-1}(\texttt{assignFeature3})$.

Let us now show that basic increasing rules equipped with the application condition constructed by Definition 4.10 are indeed direct consistency increasing rules at layer.

Theorem 4.4. *Given a constraint c in UANF and a basic increasing rule $\rho = (\text{ac}, L \xhookleftarrow{l} K \xhookrightarrow{r} R)$ w.r.t c at layer $-1 \leq k \leq \text{nl}(c) - 2$, where k is odd.*

Then, $\rho' = (\text{ac} \wedge \text{basic}_k(\rho), L \xhookleftarrow{l} K \xhookrightarrow{r} R)$ is a direct consistency increasing rule at layer k.

Proof. Given a graph G with $\text{k}_{\max} = k$. We show that each transformation $t : G \Longrightarrow_{\rho',m} H$ is direct consistency increasing w.r.t. c. Since, ρ' is a basic increasing rule at layer k, ρ' is also a consistency maintaining transformation at layer k and t satisfies the no new violation by deletion, no new violation by insertion, no satisfied layer reduction by insertion and no satisfied layer reduction by deletion formulas. Therefore, we only need to show that t satisfies the special increasing or general increasing formula respectively.

1. If ρ' is a deleting rule, $r \circ l^{-1} \circ i$ is not total, where i is the increasing morphism of ρ'. If $k = \text{nl}(c) - 2$, the transformation satisfies the special increasing formula, since one occurrence of C_{k+2} is removed. If $k < \text{nl}(c) - 2$, since $m \models \text{basic}_k(\rho)$, the morphism $m \circ i$ does not satisfy $\exists(C_{k+3}, \mathsf{true})$. Since this occurrence is destroyed, t satisfies the general increasing formula.

2. If ρ' is an inserting rule with $C' \in \text{IG}(C_{k+2}, C_{k+3})$, then $k \leq \text{nl}(c) - 2$. The morphism $m \circ i$ does not satisfy $\exists(C', \mathsf{true})$, since $m \models \text{basic}_k(\rho)$. Since ρ' is a inserting rule it holds that $\text{tr}_t \circ m \circ i = n \circ r \circ l^{-1} \circ i \models \exists(C', \mathsf{true})$ and therefore t satisfies the general increasing formula.

In summary, ρ' is a basic direct consistency increasing rule at layer k w.r.t. c. □

Example 4.6. *Again, consider the rule* `assignFeature`*, its derived rule* `assignFeatur3` *and c_1. The application condition for these rules at layer -1 w.r.t. c_1 is given in Figure 20.*

5 Rule-based Graph Repair

In the following, we present our rule-based graph repair approach. First, we propose a graph repair process for a constraint in UANF, and second, a repair process for a set of

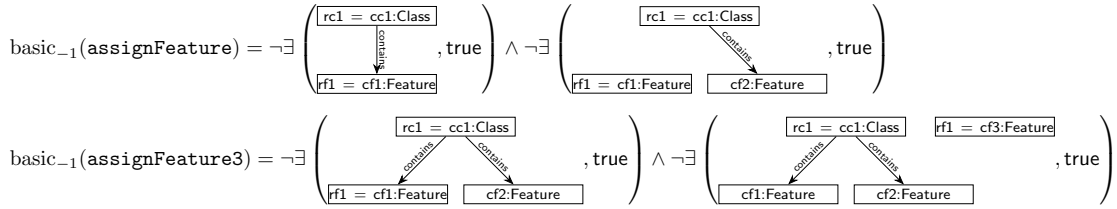

Figure 20: Application condition for `assignFeature` and `assignFeature3` with c_1 at layer 1.

constraints in UANF, both based on a given set of rules $\mathcal{R}$. In addition, we need to make further assumptions for these constraints and sets of constraints, namely that they are *circular conflict free*, in order to guarantee that our approach terminates. Intuitively, a constraint is *circular conflict free* if during a repair of an occurrence of a universally bound graph C_k, no new occurrences of C_k not satisfying $\exists(C_{k+1}, \mathsf{true})$ will be inserted and every occurrences of C_k that satisfied $\exists(C_{k+1}, \mathsf{true})$ also satisfies $\exists(C_{k+1}, \mathsf{true})$ after the repair. A set of constraints $\mathcal{C}$ is *circular conflict free* if there is a sequence $c_1, \ldots, c_n$ with $c_i \in \mathcal{C}$ such that a repair of a constraint c_i will not destroy the satisfaction of c_j for all $j < i$.

5.1 Conflicts within Conditions

During a repair process, inserting elements of an existentially bound constraint C_j could also insert new occurrences of universally bound graphs C_i. This insertion is unproblematic if $i > \mathrm{k_{max}} + 2$, but if $i \leq \mathrm{k_{max}} + 2$ it could lead either to the insertion of new violations or to a reduction of the largest satisfied layer. Additionally, removing elements of a universally bound graph C_j may destroy occurrences of an existentially bound graph C_i. Again, this can lead to the insertion of new violations or a reduction of the largest satisfied layer.

We will now introduce the notion of *conflicts within conditions*, which states that C_j has a conflict with C_i if and only if one of the cases described above can occur. Note that conflicts can only occur between existentially and universally bound graphs, and vice versa. There cannot be a conflict between two existentially bound or two universally bound graphs, since the insertion of elements cannot destroy occurrences of existentially bound graphs, and the removal of elements cannot insert new occurrences of universally bound graphs.

Definition 5.1 (conflicts within conditions). *Given a condition c in UANF. A existentially bound graph C_k* causes a conflict for *an universally bound graph C_j if there is a transformation $t : G \Longrightarrow_\rho H$ with $\rho = C_{k-1} \xhookleftarrow{\mathrm{id}} C_{k-1} \xhookrightarrow{a_{k-1}} C_k$ such that*

$$\exists p : C_j \hookrightarrow H(\neg\exists q : C_j \hookrightarrow G(\mathrm{tr}_t \circ q = p)).$$

A universally bound graph C_j causes a conflict for *an existentially bound graph C_k if there is a transformation $t : G \Longrightarrow_\rho H$ with $\rho = C_j \xhookleftarrow{a^r_{j-1}} C \xhookrightarrow{\mathrm{id}} C$ for any $C \in \mathrm{IG}(C_{j-1}, C_j)$*

such that

$$\exists p : C_k \hookrightarrow G(\mathrm{tr}_t \circ p \text{ is not total}).$$

Additionally, we introduce *conflict graphs*, which represent the conflicts within a condition via a graph. With these we are able to define *transitive conflicts*, *circular conflicts* and their absence, which will be a necessary property for the termination of our repair process. Intuitively, as the name suggests, a condition c contains a circular conflict if a graph C_k has a conflict with itself or if there exists a sequence $C_k = C_{j_1}, \ldots, C_{j_n} = C_k$ of graphs such that C_{j_i} has a conflict with $C_{j_{i+1}}$. We can check this property by checking whether the conflict graph contains cycles. Note that conflict graphs contain additional edges that do not correspond to the conflicts within the constraint. These edges ensure that during repair it can be chosen, whether a violation will be removed by deletion or insertion. Otherwise, this needs to be alternating. That means, after a violation has been removed by deletion, all violations introduced by this deletion must be removed by insertion and vice versa. Therefore the definition of repairing set also becomes more restrictive.

Definition 5.2 (**conflict graph, circular conflicts**). *Let a condition c in UANF be given. The* conflict graph *of c is constructed in the following way. For every $0 \leq k < \mathrm{nl}(c)$ there is a node labelled k. If C_k causes a conflict for C_j, there is an edge e with $\mathrm{src}(e) = k'$ and $\mathrm{tar}(e) = j'$ if either $k = k'$ or $k = k' + 1$, either $j = j'$ or $j = j' + 1$ and $j' \neq k'$.*

A graph C_k causes a transitive conflict *with C_j if there exists a path from k to j in the conflict graph of c. A graph C_k has a* circular conflict *if C_k has a transitive conflict with itself. A condition c is called* circular conflict free *if c does not contain a circular conflict.*

In other words, a condition c is *circular conflict free* if its conflict graph is acyclic.

Example 5.1. *Consider constraint c_3 and the transformations t_1 and t_2 shown in Figure 21. Transformation t_1 shows that C_1 has a conflict with C_2 because the rule $\rho = C_1 \xleftarrow{\mathrm{id}} C_1 \xhookrightarrow{a_1} C_2$ has been applied and there is a newly inserted occurrence of C_1. Transformation t_2 shows that C_2 has a conflict with C_1, since the rule $C_2 \xleftarrow{a_1} C_1 \xhookrightarrow{\mathrm{id}} C_1$ has been applied and one occurrence of C_1 has been destroyed. So c_3 contains a circular conflict, the conflict graph of c_3 is shown in Figure 22.*

In general, the statement "C_j has a conflict with C_k" does not imply that "C_k has a conflict with C_j" as shown by constraint c_4 given in Figure 21. The conflict graph of c_4 is also shown in Figure 22. It can be seen that c_4 is a circular conflict free constraint.

We will now present a characterisations of conflicts. For C_k, which is existentially bound and C_j, which is universally bound, the characterisation checks whether for each overlap of C_k and C_j, such that the overlap morphisms restricted to $C_k \setminus C_{k+1}$ and C_j overlap, the rule that only deletes $C_k \setminus C_{k-1}$ is applicable. If this is not possible, there does not exist a transformation as described in Definition 5.2. If C_k is universally bound and C_j is existentially bound, the characterisation checks whether for each overlap of C_k

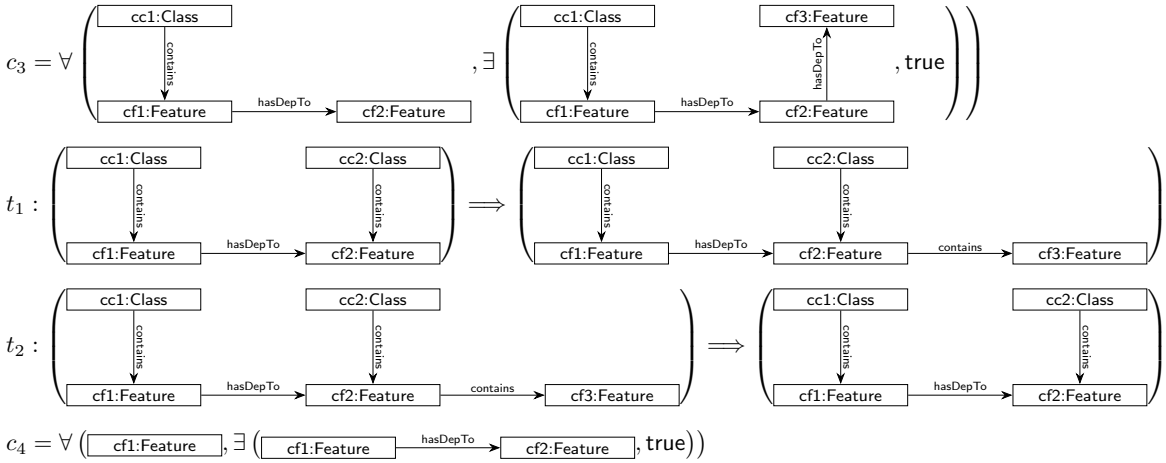

Figure 21: Constraint c_3 and the transformation that show the existence of conflicts between C_1 and C_2 and C_2 and C_1.

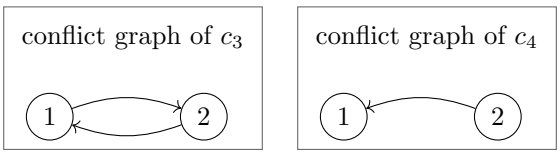

Figure 22: Conflict graphs of c_3 and c_4.

and C_j such that the elements of $C_k \setminus C_{k-1}$ and $C_j \setminus C_{j-1}$ overlap, a rule is applicable that only removes elements of $C_k \setminus C_{k-1}$. Again, if this is not possible, there is no transformation as described in Definition 5.2.

Lemma 5.3. *Given a constraint c in UANF.*

1. *Let C_k be an existentially and C_j a universally bound graph of c. Then, C_k causes a conflict for C_j, if and only if there is an overlap $P \in \mathrm{ol}(C_k, C_j)$ with*

$$i^P_{C_k}(C_k \setminus C_{k-1}) \cap i^P_{C_j}(C_j) \neq \emptyset$$

and the rule $\rho = C_k \xleftarrow{a_{k-1}} C_{k-1} \xhookrightarrow{\mathrm{id}} C_{k-1}$ is applicable at match $i^P_{C_k}$.

2. *Let C_j be a universally and C_k be an existentially bound graph of c. Then, C_j causes a conflict with C_k if an only if there is an overlap $P \in \mathrm{IG}(C_j, C_k)$ with*

$$i^P_{C_j}(C_j \setminus C_{j-1}) \cap i^P_{C_k}(C_k) \neq \emptyset$$

and a rule $\rho = C_j \xleftarrow{a^r_{j-1}} C \xhookrightarrow{\mathrm{id}} C$ with $C \in \mathrm{IG}(C_{j-1}, C_j)$ and $i^P_{C_j}(C_k \setminus C) \cap i^P_{C_k}(C_k) \neq \emptyset$ is applicable at match $i^P_{C_j}$.

Proof. Given a condition c in UANF.

1. "$\Longrightarrow$": Let C_k be an existentially bound graph that causes a conflict for an universally bound graph C_j. Then, there is a transformation $t : G \Longrightarrow_\rho H$ with $\rho = C_{k-1} \xleftarrow{\mathrm{id}} C_{k-1} \xhookrightarrow{a_{k-1}} C_k$ such that a new occurrence p of C_j is inserted. Since only elements of $C_k \setminus C_{k-1}$ are inserted, it holds that $p(C_j) \cap n(C_k \setminus C_{k-1}) \neq \emptyset$, with n being the co-match of t. The graph $P = p(C_j) \cup n(C_k)$ is the overlap we are looking for, and the rule $\rho^{-1} = C_k \xleftarrow{a_{k-1}} C_{k-1} \xhookrightarrow{\mathrm{id}} C_{k-1}$ must be applicable at the match $i^P_{C_k}$.
"$\Longleftarrow$": Let C_k be an existentially and C_j an universally bound graph such that there exists an overlap $P \in \mathrm{ol}(C_k, C_j)$ with $i^P_{C_k}(C_k \setminus C_{k-1}) \cap i^P_{C_j}(C_j \setminus C_{j-1}) \neq \emptyset$ so that the rule $\rho = C_k \xleftarrow{a_{k-1}} C_{k-1} \xhookrightarrow{\mathrm{id}} C_{k-1}$ is applicable at match $i^P_{C_k}$. Then the inverse transformation of $t : P \Longrightarrow_{\rho, i^P_{C_k}} H$ is the transformation we are looking for and C_k causes a conflict for C_j.

2. "$\Longrightarrow$": Let C_j be a universally bound graph that causes a conflict for an existentially bound graph C_k. Then, there is a transformation $t : G \Longrightarrow_\rho H$ with $\rho = C_j \xleftarrow{a^r_{j-1}} C \xhookrightarrow{\mathrm{id}} C$ and $C \in \mathrm{IG}(C_{j-1}, C_j)$ such that $\mathrm{tr}_t \circ p$ is no total for an occurrence $p : C_k \hookrightarrow G$. The graph $p(C_k) \cup m(C_j)$ is the overlap we are looking for and $i^P_{C_j}(C_j \setminus C) \cap i^P_{C_k}(C_k) \neq \emptyset$ must hold since ρ only deletes elements of $C_j \setminus C$.
"$\Longleftarrow$": Let C_j be universally and C_k existentially bound such that there is an overlap $P \in \mathrm{ol}(C_j, C_k)$ with $i^P_{C_j}(C_j \setminus C_{j-1}) \cap i^P_{C_k}(C_k) \neq \emptyset$ so that a rule $\rho = C_j \xleftarrow{a^r_{j-1}}$

$C \overset{\text{id}}{\hookrightarrow} C$ with $C \in \text{IG}(C_{j-1}, C_j)$ is applicable at match $i^P_{C_j}$. Then, the transformation of $t : P \Longrightarrow_{\rho, i^P_{C_j}} H$ is the transformation we are looking for and C_j causes a conflict for C_k.

□

Note that this characterization can also be expressed via the notions of *conflicts between rules* and *parallel independency* [13]. Using these notions, the first part of Lemma 5.3 can be expressed as: An existentially bound graph C_k causes a conflict for a universally bound graph C_j if and only if the rules $\rho = C_k \overset{a_{k-1}}{\hookleftarrow} C_{k-1} \overset{\text{id}}{\hookrightarrow} C_{k-1}$ and $C_j \overset{\text{id}}{\hookleftarrow} C_j \overset{\text{id}}{\hookrightarrow} C_j$ are parallel independent. The second part can be expressed as: A universally bound graph C_j causes a conflict for an existentially bound graph C_j if and only if the rules $C_j \overset{a^r_{j-1}}{\hookleftarrow} C \overset{\text{id}}{\hookrightarrow} C$ and $C_k \overset{\text{id}}{\hookleftarrow} C_k \overset{\text{id}}{\hookrightarrow} C_k$ are parallel independent for all $C \in \text{IG}(C_{j-1}, C_j)$.

5.2 Repairing rule Sets

Given a set of rules and a constraint, it is unclear whether or not it is possible to repair a graph using the rules of that set. Therefore, we introduce the notion of *repairing rule sets*, which is a characterisation of rule sets that are able to repair a graph w.r.t. a circular conflict free constraint. First, we introduce the notion of *repairing sequences*. A repairing sequence is a sequence of rule applications that either destroys an occurrence of a universal or inserts an occurrence of an existentially bound graph, and is applicable to each occurrence of these respective graphs. To ensure that these sequences are applicable to every occurrence, it is necessary to ensure that no nodes of these occurrences are removed and that the left-hand side of the first rule of the repairing sequence is contained in that occurrence. In other words, every repairing sequence of C_k starts with a transformation originating in C_k if C_k is universally bound and C_{k-1} if C_k is existentially bound.

Definition 5.4 (**repairing sequence**). *Let a constraint c in UANF and a set of rules $\mathcal{R}$ be given.*

1. *If C_k is existentially bound, a sequence of transformations*

$$C_{k-1} = G_0 \overset{t_1}{\Longrightarrow}_{\rho_1, m_1} G_1 \overset{t_2}{\Longrightarrow}_{\rho_2, m_2} \ldots \overset{t_n}{\Longrightarrow}_{\rho_n, m_n} G_n$$

with $\rho_i \in \mathcal{R}$ is called a repairing sequence for C_k *if $G_n \models_k c$, $\text{tr}_{t_n} \circ \ldots \circ \text{tr}_{t_1} \circ \text{id}_{C_{k-1}}$ is total. And for each universally bound graph C_j such that C_k does not cause a conflict for C_j, the concurrent rule of this sequence is a basic consistency maintaining rule w.r.t. $\forall(C_j, \mathsf{false})$.*

2. *If C_k is universally bound, a sequence of transformations*

$$C_k = G_0 \overset{t_1}{\Longrightarrow}_{\rho_1, m_1} G_1 \overset{t_2}{\Longrightarrow}_{\rho_2, m_2} \ldots \overset{t_n}{\Longrightarrow}_{\rho_n, m_n} G_n$$

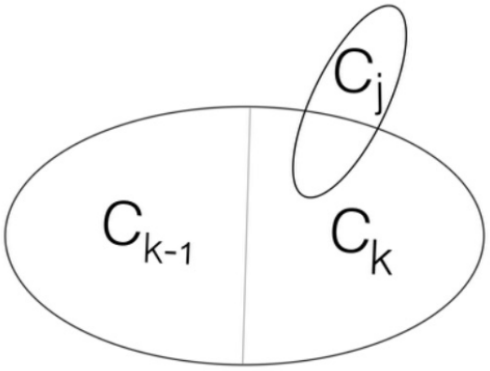

Figure 23: Scheme of a overlap of existentially bound graph C_k and universally bound graph C_j that could lead to an insertion of C_j via repairing sequences.

with $\rho_i \in \mathcal{R}$ is called a repairing sequence for C_k *if $G_n \models_k c$, for each node $v \in V_{G_0}$ there is a node $v' \in V_{G_n}$ with $v' = \mathrm{tr}_{t_n}(\ldots \mathrm{tr}_{t_1}(v))$ and the concurrent rule of this sequence is a basic consistency maintaining rule w.r.t. $\forall(C_j, \mathsf{false})$ for all universally bound graphs C_j.*

Note that this definition prohibits the deletion of nodes of V_G. If nodes are deleted, there is no guarantee that the repairing sequence is applicable at every occurrence of the corresponding graph since the dangling edge condition can be violated. Also note that a repairing sequence for a universally bound graph C_j can only insert occurrences of an existentially bound graph C_k if C_j causes a conflict for C_j.

In both cases of Definition 5.4 the insertion of additional elements, i.e. $G_n \neq C_{k+1}$ if C_k is existentially bound and $G_n \neq C$ for all $C \in \mathrm{IG}(C_{k-1}, C_k)$ if C_k is universally bound, could lead to the insertion of universally bound graphs. For an existentially bound graph, this can happen if there is an overlap with a universally bound graph in a similar way as shown in Figure 23. To ensure that this does not happen, we need the additional condition that the concurrent rule is a basic consistency maintaining rule with respect to certain constraints. If $G_n = C_{k+1}$ if C_k is existentially or $G_n = C$ with $C \in \mathrm{IG}(C_{k-1}, C_k)$ if C_k is universally bound, this condition is not needed as the following Theorem shows.

Theorem 5.1. *Let a constraint c in UANF and a set of rules $\mathcal{R}$ be given.*

1. *If C_k is existentially bound and there is sequence*

$$C_{k-1} \Longrightarrow_{\rho_1,m_1} \ldots \Longrightarrow_{\rho_n,m_n} C_k$$

with $\rho_i \in \mathcal{R}$ such that $\mathrm{tr}_{t_n} \circ \ldots \mathrm{tr}_{t_1} \circ \mathrm{id}_{C_{k-1}}$ is total and $C_k \models_k c$. Then, this is a repairing sequence for C_k.

2. *If C_k is universally bound and there is a sequence*

$$C_k \Longrightarrow_{\rho_1,m_1} \ldots \Longrightarrow_{\rho_n,m_n} C$$

with $\rho_i \in \mathcal{R}$, $C \in \mathrm{IG}(C_{k-1}, C_k)$, $C \models_k c$ *and for each node* $v \in V_{G_0}$ *there is a node* $v' \in V_{G_n}$ *with* $v' = \mathrm{tr}_{t_n}(\dots \mathrm{tr}_{t_1}(v))$. *Then, this is a repairing sequence for* C_k.

Proof. 1. If C_k is existentially bound, we need to show that the concurrent $\rho = C_{k-1} \xhookleftarrow{\mathrm{id}} C_{k-1} \xhookrightarrow{a_k} C_k$ is a basic consistency maintaining rule w.r.t. $\forall(C_j, \mathsf{false})$ for all universally bound C_j such that C_k does not cause a conflict with C_j. Assume that C_j is a universally bound graph such that C_k does not cause a conflict for C_j and ρ is not a basic consistency maintaining rule w.r.t. $\forall(C_j, \mathsf{false})$. With Lemma **??** follows immediately that C_k causes a conflict for C_j, this is a contradiction.

2. If C_k is universally bound, the concurrent rule is given by $\rho = C_k \xhookleftarrow{a^r_{k-1}} C \xhookrightarrow{\mathrm{id}} C$. Then, ρ is a basic consistency maintaining rule w.r.t. $\forall(C_j, \mathsf{false})$ for all universally bound graphs since ρ does not insert any elements.

□

Note that this characterisation is similar to the construction of repairing rules in [18].

Definition 5.5 (repairing rule set). *Let a set of rules* $\mathcal{R}$ *and a circular conflict free constraint* c *in UANF be given. Then,* $\mathcal{R}$ *is called a* repairing rule set of c *if there does exist a repairing sequence for each existentially bound graph of* c *and, if* $\mathrm{nl}(c)$ *is odd, i.e.* c *ends with a condition of the form* $\forall(C_{\mathrm{nl}(c)}, \mathsf{false})$, $\mathcal{R}$ *contains a repairing sequence for* $C_{\mathrm{nl}(c)}$.

Note that there cannot exist a repairing sequence for a universally bound graphs C_k such that $C_k \setminus C_{k-1}$ does not contain any edges. Therefore, there is no repairing set for all constraints of the form $\forall(C_1, \mathsf{false})$ such that $E_{C_1} = \emptyset$.

Theorem 5.2. *Let a circular conflict free constraint* c *in UANF and a repairing set* $\mathcal{R}$ *of* c *be given. Then, for each graph* G *with* $G \not\models c$, *there is a sequence of transformations*

$$G = G_0 \Longrightarrow_{\rho_1, m_1} \dots \Longrightarrow_{\rho_n, m_n} G_n$$

with $\rho_i \in \mathcal{R}$ *such that* $G_n \models c$.

We will postpone the proof of this Theorem, as it follows immediately from the termination of our repair process.

Example 5.2. *Consider the constraints* c_1, c_4 *and the sequences shown in Figure 24. The first sequence is not a repairing sequence for the existentially bound graph of* c_4, *since* $G_1 \not\models_1 c_4$ *and therefore a rule set containing only this rule is not a repairing set w.r.t* c_4. *The second sequence is a repairing sequence for the existentially bound graph of* c_4, *since the last graph satisfies* c_4 *and the existentially bound graph has a conflict with the universally bound graph. Therefore, the condition for the concurrent rule is also satisfied, and a rule set containing this rule is a repairing set w.r.t.* c_4.

The third sequence is a repairing sequence for c_1 *since the last graph satisfies* c_1 *and the sequence satisfies the criteria given in Theorem 5.1. Note that this sequence consists of two applications of the same rule. A set of rules containing this rule is a repairing set w.r.t.* c_1.

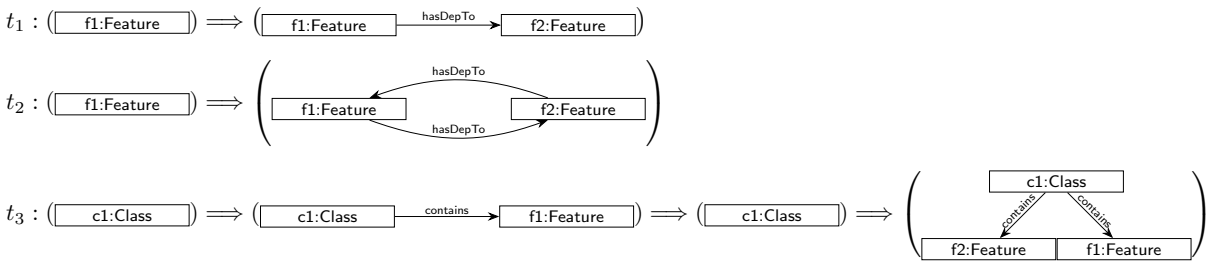

Figure 24: Repairing sequences for c_1 and c_4.

Algorithm 1: Determine $\mathrm{k}_{\max}$.

Data: A graph G, a constraint c in UANF.
Result: $\mathrm{k}_{\max}$.

```
if G ⊨ c then
    return nl(c) − 1;
end
for i ← −1 to nl(c) − 1 by 2 do
    if G ⊭ cut_i(c) then
        return i − 2;
    end
end
```

5.3 Rule-based Graph Repair for one Constraint

In the following, we present our graph repair process for a circular conflict-free constraint in UANF. We first start with an algorithm which calculated $\mathrm{k}_{\max}$ given a graph G and a constraint c in UANF, it is shown in Algorithm 1.

Theorem 5.3. *Given a graph G and a constraint c in UANF, Algorithm 1 returns* $\mathrm{k}_{\max}(c, G)$.

Proof. If $G \models c$, then $\mathrm{k}_{\max} = \mathrm{nl}(c) - 1$ which will be returned by the algorithm. Otherwise, if $G \not\models c$, Corollary 3.10 implies that $G \not\models_k c$ for all even $-1 \leq k < \mathrm{nl}(c)$. Therefore, $\mathrm{k}_{\max}$ must be odd. If the algorithm returns k, which is odd, it holds that $G \not\models_{k+2} c$ and Lemma 3.11 implies that k must be equal to $\mathrm{k}_{\max}$.

□

The repair process is shown in Algorithm 2 and proceeds as follows. The algorithm starts by finding all potentially increasing occurrences p of $C_{\mathrm{k}_{\max}+2}$ at layer $\mathrm{k}_{\max}$ w.r.t. c. Recall, these occurrences satisfy the following:

1. $p \not\models d$.
2. $p = a_{\mathrm{k}_{\max}+1} \circ \ldots \circ a_0 \circ q$ where $a_i \circ \ldots \circ q \models \mathrm{sub}_{i+1}(\mathrm{cut}_{\mathrm{k}_{\max}}(c))$ for all $0 \leq i \leq \mathrm{k}_{\max}$ and $q : \emptyset \hookrightarrow G$ is the empty morphism.

The condition d is equal to false if $\mathrm{k}_{\max}+2 = \mathrm{nl}(c) - 2$ and equal to $\exists(C_{\mathrm{k}_{\max}+2}, \mathsf{true})$ otherwise. All these occurrences are contained in the set P (line 3). If P is empty, Lemma 3.16 implies $G \models_{\mathrm{k}_{\max}+2} c$, and so we only will apply repairing sequences at occurrences contained in this set. It may be sufficient to repair only some of these occurrences. Since we do not know which of these are likely to increase the consistency, we choose one at random (line 4). For example, for existentially bound constraints d, i.e. their equivalent constraint in UANF is $\forall(\emptyset, d)$, there may exist occurrences of $C_{\mathrm{k}_{\max}+2}$ whose repair will never lead to an increase of the largest satisfied layer.

There are two ways to repair the selected occurrence, either by destroying it, or by inserting elements such that the occurrence satisfies $\mathrm{cut}_0(\mathrm{sub}_{\mathrm{k}_{\max}+2}(c))$. The algorithm chooses one of these options (line 5) and applies the appropriate repair sequence (lines 6–12). Note that there may be no repair sequence for $C_{\mathrm{k}_{\max}+2}$, since this graph is universally bound. If this is the case, we use the repairing sequence for $C_{\mathrm{k}_{\max}+3}$. This must exist because $C_{\mathrm{k}_{\max}+3}$ is existentially bound and $\mathcal{R}$ is a repairing set for c.

If the repairing sequence for $C_{\mathrm{k}_{\max}+2}$ was applied, occurrences of existentially bound graphs may have been destroyed. Note that these can only be occurrences of graphs C_i such that $C_{\mathrm{k}_{\max}+2}$ has a conflict with C_i. This could lead to a reduction of the largest satisfied layer. Therefore the algorithm finds all these destroyed occurrences, in particular it finds all occurrences p of universally bound graphs C_i such that an occurrence q of C_{i+1} with $p = q \circ a_j$ has been removed (line 8). If the repairing sequence for $C_{\mathrm{k}_{\max}+3}$ has been applied, occurrences of universally bound graphs may have been inserted. Again, these can only be occurrences of graphs C_i such that $C_{\mathrm{k}_{\max}+2}$ has a conflict with C_i and this could lead to a decrease of the largest satisfied layer. Again, the algorithm finds all inserted occurrences of universally bound graphs (line 11). Again, in both cases, the algorithm only finds potentially increasing occurrences at the respective layer w.r.t. c. If the largest satisfied layer has not been reduced, the algorithm chooses the next occurrence in P.

Otherwise, the largest satisfied layer must be restored. To do this, all newly inserted potentially increasing occurrences of universally bound graphs and potentially increasing occurrences p of universally bound graphs C_j such that $p \models \exists(C_{j+1}, \mathsf{true})$ and $\mathrm{tr} \circ p \not\models \exists(C_{j+1}, \mathsf{true})$ where tr is the morphism of the repairing sequence application are collected in the set M and must be repaired. Again, it might be sufficient to only repair some of these occurrences in order to restore the largest satisfied layer. Since it is unclear for which of these occurrences a deletion or extensions leads to increase of the largest satisfied layer, we select one at random. Repairing these occurrences may again result in the insertion of existentially bound graphs or the removal of universally bound graphs. These occurrences are added to M, and this process is repeated until the largest satisfied layer is restored, i.e. $H \models_{\mathrm{k}_{\max}} c$ (line 13 – 26). The whole process is repeated until a graph satisfying c is derived.

From this it is clear why c must be circular conflict free. For a constraint with circular conflicts, during the restoring phase a new occurrence of $C_{\mathrm{k}_{\max}+2}$ can be inserted and an occurrence of $C_{\mathrm{k}_{\max}+3}$ can be removed. In certain cases this could lead to an infinite loop, so there is no guarantee that this algorithm will terminate. For example, consider

the constraint c_3 given in Figure 21. The set of rules used for the transformations t_1 and t_2 in figure 21 forms a repairing set. During a repair process using Algorithm 2, where the starting graph is the first graph of t_1, it is possible for the Algorithm 2 to enter an infinite loop by alternately applying t_1 and t_2.

For any circular conflict-free constraint, Algorithm 2 will always terminate according to the following Theorem.

Theorem 5.4. *Given a graph G, a circular conflict free condition c in UANF and a repairing set $\mathcal{R}$ of c. Then, Algorithm 2 with input G, c and $\mathcal{R}$ terminates and returns a graph H with $H \models c$.*

Proof. If Algorithm 2 terminates, it returns a graph that satisfies c. Therefore, it is sufficient to show that Algorithm 2 terminates. Since G is finite, the set P must also be finite. If a repairing sequence has been applied, the set M contains only occurrences of graphs C_j such that $C_{\mathrm{k_{max}}+2}$ causes a (transitive) conflict with C_j, since the repairing sequence is not able to destroy or insert occurrences of C_i such that $C_{\mathrm{k_{max}}+2}$ does not cause a conflict with C_i. Note that this is also the case for repairing sequence for universally bound graphs. Since G is finite, $|M|$ must also be finite.

If the derived graph does not satisfy $\mathrm{cut}_{\mathrm{k_{max}}(c,G)}(c)$, we need to restore the largest satisfied layer. Since the largest satisfied layer only decreases if an occurrence of an existentially bound graph C_j is destroyed such that an occurrence p of C_{j-1} satisfies $\exists(C_j, \mathsf{true})$ and $\mathrm{tr} \circ p$ does not satisfy $\exists(C_j, \mathsf{true})$ or an occurrence of universally bound graphs is inserted, and M contains all these occurrences that can also improve the satisfaction at layer (see Lemma 3.16) , we only need to consider the occurrences contained in M. Applying repairing sequences to occurrences $p : C_j \hookrightarrow H \in M$ could again lead to the insertion of universally bound graphs or the removal of existentially bound graphs. The set M' contains all these occurrences, and again these are only occurrences of C_i such that C_j causes a (transitive) conflict for C_i. Since c is free of circular conflict, M' cannot contain any occurrences of $C_{\mathrm{k_{max}}+2}$, otherwise C_j would have cause a (transitive) conflict for $C_{\mathrm{k_{max}}+2}$ and therefore $C_{\mathrm{k_{max}}+2}$ has a circular conflict. Therefore no occurrences of $C_{\mathrm{k_{max}}+3}$ are destroyed and no occurrences of $C_{\mathrm{k_{max}}+2}$ are inserted. In addition, $C_{\mathrm{k_{max}}+2}$ causes a (transitive) conflict for C_i, and repairing any $p \in M'$ will not lead to insertion of an occurrence of $C_{\mathrm{k_{max}}+2}$ or removal of an occurrence of $C_{\mathrm{k_{max}}+3}$.

Since c is circular conflict free, there must exist graphs C_i, such that C_i causes no conflict with any other graph $C_{i'}$ and $C_{\mathrm{k_{max}}+2}$ causes a (transitive) conflict for C_i. Therefore, the application of repairing sequences at occurrences of these graphs will not lead to the insertion or removal of any universally or existentially bound graph, respectively. Since c is finite, the number of graphs C_i such that $C_{\mathrm{k_{max}}+2}$ causes (transitive) conflict with C_i is finite. Since $|M'|$ is also finite, after a finite number of applications of repairing sequences, M' contains only occurrences of graphs that do not cause any conflicts. After a repairing sequence has been applied to all these occurrences, M' is empty and $H \models_{\mathrm{k_{max}}(c,G)} c$, since all occurrences p of C_j which have either been inserted or an occurrence q of C_{j+1} with $p = a_j \circ q$ has been removed satisfy $\exists(C_{j+1}, \mathsf{true})$ or have been removed.

Algorithm 2: Repair for one circular conflict free constraint

Data: A graph G, a circular conflict free constraint c in UANF and a repairing set $\mathcal{R}$ for c.

Result: A graph H with $H \models c$.

while $G \not\models c$ **do**
  Determine $\mathrm{k}_{\max}$ using Algorithm 1 ;
  $P \leftarrow \{p : C_{\mathrm{k}_{\max}+2} \hookrightarrow H \mid p \not\models \mathrm{cut}_0(\mathrm{sub}_{\mathrm{k}_{\max}+2}(c))$ and p is a potentially increasing occurrence at layer $\mathrm{k}_{\max}$ w.r.t. $c\}$;
  Choose $p \in P$ uniformly at random ;
  Choose $r \in \{0, 1\}$ uniformly at random;
  if $r = 0$ *and* $\mathcal{R}$ *contains a repairing sequence for* $C_{\mathrm{k}_{\max}+2}$ **then**
    Apply the repairing sequence for $C_{\mathrm{k}_{\max}+2}$ at match p and let H be the derived graph ;
    $M \leftarrow \{q : C_j \hookrightarrow H \mid j < \mathrm{k}_{\max}+2$ odd and $\neg\exists q' : C_j :\hookrightarrow G(\mathrm{tr} \circ q' = q)$ and q is a potentially increasing occurrence at layer $j-2$ w.r.t. $c\}$;
  else
    Apply the repairing sequence for $C_{\mathrm{k}_{\max}+3}$ at match p and let H be the derived graph and tr the track morphism ;
    $M \leftarrow \{q : C_j \hookrightarrow H \mid j < \mathrm{k}_{\max}+2$ odd and $\exists q' : C_j \hookrightarrow G(q = \mathrm{tr} \circ q'$ is total, $q' \models \exists(C_{j+1}, \mathsf{true})$ and $q \not\models \exists(C_{j+1}, \mathsf{true}))$ and q is a potentially increasing occurrence at layer $j-2$ w.r.t. $c\}$;
  end
  while $H \not\models_{\mathrm{k}_{\max}(c,G)} c$ **do**
    Choose $p : C_j \hookrightarrow H \in M$ uniformly at random ;
    Choose $r \in \{0, 1\}$ uniformly at random ;
    if $r = 0$ *and* $\mathcal{R}$ *contains a repairing sequence for* C_j **then**
      Apply the repairing sequence for C_j at match p and let H' be the derived graph ;
      $M' \leftarrow \{q : C_{j'} \hookrightarrow H' \mid j'$ odd and $\neg\exists q' : C_{j'} \hookrightarrow H(\mathrm{tr} \circ q' = q)$ and q is a potentially increasing occurrence at layer $j'-2$ w.r.t. $c\}$;
    else
      Apply the repairing sequence for C_{j+1} at match p and let H' be the derived graph and tr the track morphism;
      $M' \leftarrow \{q : C_{j'} \hookrightarrow H' \mid j < \mathrm{k}_{\max}+2$ odd and $\exists q' : C_{j'} \hookrightarrow H(q = \mathrm{tr} \circ q'$ is total, $q' \models \exists(C_{j'+1}, \mathsf{true})$ and $q \not\models \exists(C_{j'+1}, \mathsf{true}))$ and q is a potentially increasing occurrence at layer $j'-2$ w.r.t. $c\}$;
    end
    $M \leftarrow M' \cup \{\mathrm{tr} \circ q \mid q \in M \setminus \{p\}$ and $\mathrm{tr} \circ q$ is total$\}$;
    $H \leftarrow H'$;
  end
  $G \leftarrow H$;
end
return G;

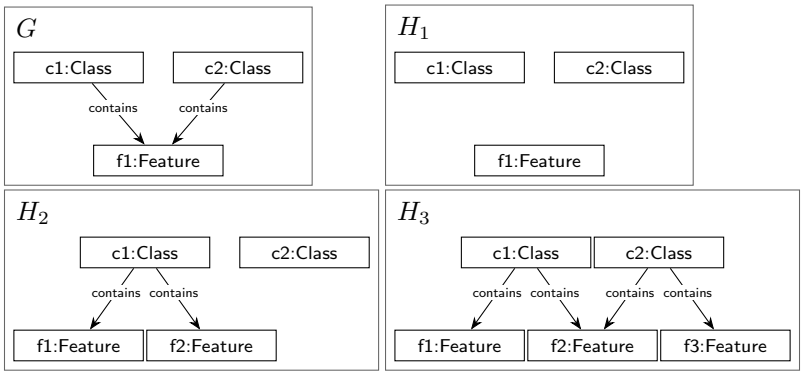

Figure 25: Possible outputs of the repairing process for G and $\forall(C_2^2, \exists(C_2^1, \mathsf{true}))$ using the rule set $\{\texttt{removeFeature}, \texttt{createFeature}\}$.

Therefore, after a finite number of iterations, the set P is empty and Lemma 3.16 implies that the largest satisfied layer has been increased by at least 1. It follows that after a finite number of iterations $G \models c$. Then Algorithm 2 terminates and returns G. □

Example 5.3. *Consider constraint $c = \forall(C_2^2, \exists(C_2^1, \mathsf{true}))$ which is composed of the graphs shown in Figure 5. This constraint is circular conflict free and a repairing set for c is given in Figure 25. There is a repairing sequence for C_2^2 via the rule* `remove` *and a repairing sequence for C_2^1 via the rule* `insert`*. Using the rule set* $\{\texttt{remove}, \texttt{insert}\}$*, Algorithm 2 could return one of the graphs G_1, G_2 or G_3 given in Figure 25, depending on the repairing sequences used.*

5.4 Rule-based Graph Repair for multiple Constraints

We will now present our rule-based repair approach for a set of constraints in UANF.

Definition 5.6 (**satisfaction of constraint sets**). *Let $\mathcal{C}$ be a set of constraints. A graph G satisfies $\mathcal{C}$, denoted by $G \models \mathcal{C}$, if $G \models \bigwedge_{c \in \mathcal{C}} c$. The set $\mathcal{C}$ is called* satisfiable *if there exists a graph G with $G \models \mathcal{C}$.*

To guarantee that a set of constraints can be repaired by a set of rules, we need to extend the notion of repairing sets such that a set of rules is called a *repairing set* for a set of constraints if it is a repairing set for every constraint in the constraint set.

Definition 5.7 (**repairing set for a set of constraints**). *Given a set of constraints $\mathcal{C}$ and a set of rules $\mathcal{R}$. Then $\mathcal{R}$ is called a* repairing set for $\mathcal{C}$ *if $\mathcal{R}$ is a repairing set for all constraints $c \in \mathcal{C}$.*

We also extend the notion of conflicts to *conflicts between constraints*. Intuitively, a constraint c has a conflict with another constraint c' if one of its graphs has a conflict with a graph of c'.

Algorithm 3: Repair for a circular constraint-conflict free set of constraints

Data: A graph G, circular constraint-conflict free set of constraints $\mathcal{C}$ and a repairing set $\mathcal{R}$ for $\mathcal{C}$.
Result: A graph H with $H \models \bigwedge_{c \in \mathcal{C}} c$.

```
(c_1, ..., c_n) ← topological ordering of C ;
for i ← 1 to n do
    Repair c_i in G with Algorithm 2, let H be the returned graph ;
    G ← H ;
end
return G;
```

Definition 5.8 (**conflict between constraints**). *Let the constraints c, c' in UANF and a set of rules $\mathcal{R}$ be given. Then c has a conflict with c' w.r.t. $\mathcal{R}$ if a repairing sequence*

$$C_k = G_0 \Longrightarrow_{\rho_1,m_1} \cdots \Longrightarrow_{\rho_n,m_n} G_n$$

exists for a graph C_k of c such that the concurrent rule of that sequence is not a basic consistency maintaining rule w.r.t. $\forall(C_j, \mathsf{false})$ or $\exists(C_j, \mathsf{true})$ for any universally or existentially bounded graph C_j of c'.

The following lemma is a useful statement for proving the correctness of our repair approach. It states that applying a repairing sequence to a constraint c cannot destroy the satisfaction of c' if c has no conflict with c'.

Lemma 5.9. *Let two constraints c and c' in UANF be given such that c has no conflict with c' w.r.t. to a set of rules $\mathcal{R}$. Then the concurrent rule ρ of any repairing sequence for c is a c'-preserving rule.*

Proof. Suppose ρ is not a c'-preserving rule. Then there exists a transformation $t : G \Longrightarrow_{\rho,m} H$ such that $G \models c'$ and $H \not\models c'$. Therefore, either a universally bound graph of c' has been inserted or an existentially bound graph of c' has been removed. It follows that ρ is not a basic maintaining rule w.r.t. $\forall(C_j, \mathsf{false})$ for all universally bound graphs C_j of C' or ρ is not a basic maintaining rule w.r.t. $\exists(C_j, \mathsf{true})$ for all existentially bound graphs C_j of C', which is a contradiction. □

The *conflict graph* for a set of constraints and *circular conflicts* for it are defined in a similar way to the conflict graph and circular conflicts for a constraint. A set of constraints is called *circular conflict free* if each of its constraints is circular conflict free and there is no sequence $c = c_0, \ldots, c_n = c$ such that c_i has a conflict with c_{i+1} for all $0 \leq i < n$. In other words, the conflict graph of this set is acyclic.

Definition 5.10 (**conflict graphs, circular conflicts**). *Given a set of constraints $\mathcal{C}$ in UANF. The* conflict graph *of $\mathcal{C}$ is constructed in the following way. For each constraint $c \in \mathcal{C}$ there is a node. If there is a conflict between c and c', there exists an edge e with* $\mathrm{src}(e) = c$ *and* $\mathrm{tar}(e) = c'$.

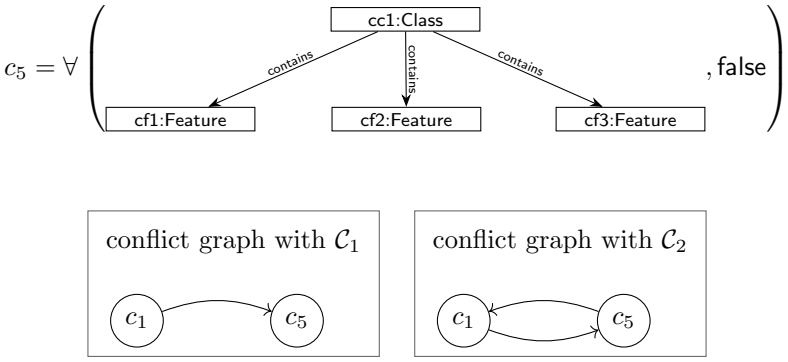

Figure 26: Constraints c_5 and conflicts graphs of the constraint set $\{c_1, c_5\}$ with the rule sets $\mathcal{C}_1 = \{\texttt{resolve}, \texttt{createFeatures}\}$ and $mathcal C_2 = \{\texttt{resolve2}, \texttt{createFeatures}\}$.

A constraint c has a transitive conflict *with c' if the conflict graph of $\mathcal{C}$ contains a path from c to c'. A constraint c has a* circular conflict *if c has a transitive conflict with itself. A set of constraints $\mathcal{C}$ is called* circular conflict free *if every constraint in $\mathcal{C}$ is circular conflict free and $\mathcal{C}$ contains no circular conflicts.*

Example 5.4. *Consider the rules* `resolve, resolve2, createFeatures` *and constraints c_1 and c_5 given in Figures 26 and 5. The constraint set $\mathcal{C} = \{c_1, c_5\}$ is a multiplicity which states that "Each node of type* `Class` *is connected to exactly two nodes of type* `Feature`*". With the rule set $\mathcal{R}_1 = \{\texttt{resolve}, \texttt{createFeatures}\}$, there is only one conflict in $\mathcal{C}$; c_1 has a conflict with c_5, since applying* `createFeatures` *could lead to inserting the universally bound graph of c_5. With the rule set $\mathcal{R}_2 = \{\texttt{resolve2}, \texttt{createFeatures}\}$ there are two conflicts. Again, there is a conflict of c_1 with c_5, but also a conflict of c_5 with c_1, since applying* `resolve` *can destroy an occurrence of the existentially bound graph of c_1.*

Therefore, our approach can repair with the rule set $\mathcal{R}_1$ but not with $\mathcal{R}_2$ because in this case, $\mathcal{C}$ is not circular conflict free.

Our repair process exploits the fact that the conflict graph of a circular conflict-free set of constraints in UANF is acyclic. In particular, our approach uses the *topological ordering* of this conflict graph.

Definition 5.11 (topological ordering of a graph). *Given is a graph G. A sequence $(v_1, \ldots, v_n)$ of nodes of G is called a* topological ordering *of G if no edge $e \in E_G$ exists with $\mathrm{src}(e) = v_i$, $\mathrm{tar}(e) = v_j$ and $i \geq j$. The topological ordering of a circular conflict-free set of constraints $\mathcal{C}$ is the topological order of its conflict graph.*

It is well known that every directed acyclic graph has a topological ordering that can be computed in $\Theta(|V| + |E|)$ where V and E are the set of nodes and edges of the respective graph [6]. Therefore every conflict graph of a circular conflict-free set of constraints also has a topological ordering .

The repair process is given in Algorithm 3 and proceeds as follows. First, the topological ordering of the constraint set is determined (line 1). Then Algorithm 2 is used to repair each constraint of $\mathcal{C}$ in the order of the topological ordering (lines 2 – 4). This ensures that the satisfaction of a constraint that has already been repaired is not destroyed by the repair of another constraint.

Theorem 5.5. *Given a graph G, a satisfiable, circular conflict-free set of constraints in UANF, $\mathcal{C}$, and a set of rules $\mathcal{R}$. Then Algorithm 3 terminates and returns a graph H with $H \models \mathcal{C}$.*

Proof. Since $\mathcal{C}$ is finite and every $c \in \mathcal{C}$ is circular conflict free, Algorithm 2 terminates for every $c \in \mathcal{C}$. Therefore Algorithm 3 will also terminate. It remains to show that the returned graph satisfies $\mathcal{C}$. Let $(c_1, \ldots, c_n)$ be a topological ordering of $\mathcal{C}$. Then no constraint c_j with $j \neq 1$ has a conflict with c_1, and by Lemma 5.9 it follows that the concurrent rule of every repair sequence for every c_i with $2 \leq i \leq n$ is a c_1-preserving rule. In general, the concurrent rule of each repairing sequence for c_j is a c_i-preserving rule if $i < j$. Note that in Algorithm 3 each repairing sequence can be replaced by its concurrent rule. After one iteration it holds that $G \models c_1$. Suppose that after m iterations it holds that $G \models c_i$ for all $1 \leq i \leq m$. In iteration $m+1$, c_{m+1} is repaired by Algorithm 3. Since each concurrent rule of each repairing sequence of c_{m+1} is a c_i-preserving rule for all $1 \leq i \leq m$, the application of repairing sequence can be replaced by its concurrent rule, and Algorithm 3 applies only these concurrent rules, it follows that $H \models c_i$ for all $1 \leq i \leq m+1$. Therefore, after n iterations, $H \models c_i$ for all $1 \leq i \leq n$ and the returned graph G satisfies $\mathcal{C}$. □

6 Related Work

In this section, we summarise other concepts for rule-based graph repair.

Iterative Development of Consistency-Preserving Rule-Based Refactorings: Becker et al. [3] introduced an interactive approach to construct consistency-preserving transformation based on their invariant checker introduced in [2] for so-called *well-formedness constraints*. These are constraints of the form $\neg\exists(c_1, \mathsf{true})$ or $\neg\exists(C_1, \neg\exists C_2, \mathsf{true})$. Given a consistent graph, a well-formedness constraint and a refactoring specification, which is a set of rules in the single-pushout approach [7], the invariant checker constructs all minimal counterexamples that lead to a non-consistency preserving transformation via rules of the refactoring specification. If there are no such counterexamples, then any transformation is consistency-preserving. This approach is designed to be fully interactive, requiring the user to revise the refactoring specification until no counterexamples are returned.

Ensuring Consistency of Conditional Graph Grammars: Heckel and Wagner [11] have presented an approach to construct consistency-preserving application conditions for rules in the single-pushout approach and a constraint of the form $\forall(C_1, \exists(C_1', \mathsf{true})) \wedge$

$\ldots \wedge \forall(C_n, \exists(C'_n, \mathsf{true}))$. Although the constructed application conditions are not presented as nested-conditions, they can be transformed into nested conditions of the form $\forall(C_1, \exists(C_1^1, \mathsf{true}) \vee \ldots \vee \exists(C_1^{k_1}, \mathsf{true})) \wedge \ldots \wedge \forall(C_n, \exists(C_n^1, \mathsf{true}) \vee \ldots \vee \exists(C_1^{k_n}, \mathsf{true}))$.

Sustaining and Improving graduated Graph Consistency: Kosiol et al. [12] have introduced the notions of (direct) consistency-sustaining and (direct) consistency-improving transformations as already introduced in section 2.4. This approach is designed for rules in the double-pushout approach and nested-conditions in ANF. They have introduced a method to construct consistency-sustaining application conditions, a sufficient criterion for consistency-sustaining transformations and a necessary criterion for consistency-improving transformations, which have been implemented and evaluated.

Constructing optimized constraint-preserving application conditions for model transformation rules: Nassar et al. [14] have introduced a method to construct consistency-sustaining and consistency-preserving application conditions in the framework of $\mathcal{M}-$adhesive categories. Due to some optimisations, these application conditions are less restrictive and less complex than those described in [10] and [12]. They have introduced the notion of *weakest application conditions*. As the name suggests, a weakest application condition is implied by any other application condition with the same property. For example, a weakest consistency-preserving application condition is implied by every other consistency-preserving application condition. The construction of the application conditions has been implemented as an eclipse plug-in called *OCL2AC*, which is able to construct consistency-guaranteeing, weakest consistency-preserving or consistency-sustaining application conditions.

Rule-based Graph Repair: Sandmann and Habel [18] have introduced a repair process for so-called *proper constraints* based on so-called *repair programs*. A constraint in ANF is called *proper* if it ends with $\exists(C, \mathsf{true})$ or is of the form $\exists(C_1, \forall(C_2, \mathsf{false}))$ or $\forall(C_1, \mathsf{false})$. The authors describe a method for inductively constructing a repair program consisting of rules in the double-pushout approach that, when applied to a non-consistent graph, returns a consistent graph. They also introduce graph repair given a set of rules $\mathcal{R}$. The approach can be used to repair a graph with rules from $\mathcal{R}$ if there is a repair program such that for every rule in the repair program there is an equivalent rule in $\mathcal{R}$.

Rule-based Repair of EMF Models: Nassar et al. [16, 15] introduced a repair approach for models of the *eclipse modeling framework* (EMF) [19]. In particular, this approach is able to repair multiplicities of a given EMF metamodel. Multiplicities can be described as nested conditions of the form $\forall(C_1, \exists(C_2, \mathsf{true}))$ and $\forall(C_1, \mathsf{false})$. The approach was implemented using two Eclipse plugins in Henshin [1]. One plugin derives rules for the repair process from a metamodel. The other plugin is an implementation of the repair process.

7 Conclusion

In summary, we have introduced a new concept of consistency, which leads to the new notions of consistency-maintaining and consistency-increasing transformations and rules, which is more fine-grained, compared to the already existing notions of consistency preserving, consistency guaranteeing, consistency sustaining and consistency improving. Finer-grained, in the sense that the smallest increases and decreases of consistency can be detected, namely the insertion or removal of one graph element. We have compared our notions with the already existing notions mentioned above. The notions of consistency-maintaining and consistency-improving are related to the notions of consistency-preserving and consistency-guaranteeing, and are generally not related to the notions of consistency-sustaining and consistency-improving. Only in special cases, for specific constraints, are these notions related or even identical.

Furthermore, we have introduced application conditions such that a rule equipped with this condition is consistency-maintaining or consistency-increasing. In particular, we introduce two types of application conditions. First, application conditions for general rules, and second, application conditions for a special set of rules, which we call basic-maintaining and basic-increasing rules, respectively, where the application conditions for basic rules are less complex and smaller in size compared to the general application conditions.

We have introduced a rule-based graph repair process based on our new notions of consistency-maintaining and consistency-increasing for a particular type of constraints in ANF, which we call circular conflict-free constraints, using a given rule set. We present characterisations of these circular conflict free constraints and a characterisation of a rule set that is able to repair a constraint using our repair process. In addition, we extend the notion of conflicts in constraints to the notion of conflicts between constraints, and present a rule-based graph repair process for a satisfiable circular conflict free set of constraints, which is based on the repair process for one constraint.

Future work is to extend the notions of consistency-maintaining and consistency-increasing transformations for all types of nested constraints, and a rule-based repair process for all satisfiable nested constraints, i.e. constraints that also use Boolean operators.

There are further optimisations for the application conditions for general rules. First, not all overlaps in the set $\mathbf{P}_{C'}$ need to be considered in order to construct the no violation inserted part of the maintaining application condition. Some of these constructed conditions are implied by other conditions and can therefore be removed. Second, the application condition constructed from the violation removed part of the consistency increasing application condition is way too restrictive since all occurrences of P' that extend R must satisfy $\exists(C', \mathsf{true})$ after the transformation, since we are unable to transform information about occurrences of C_{k+2} from $\mathrm{exv}(P, C')$ to $\mathrm{remv}(P, C')$. Obviously, only one of these occurrences of P' would be sufficient to claim that the transformation is consistency-increasing. We are confident that, with some additional theory, the $\mathrm{exv}(P, C')$ and $\mathrm{remv}(P, C')$ parts of the application can be combined in order to construct

less restrictive application conditions.

Although we have presented characterisations for circular conflict-free constraints, it remains unclear for which practical applications our approach is suitable. This may require implementation and further evaluation of the repair process, characterisations and construction of application conditions. In addition, the notion of conflict between constraints is very strict, and we are confident that there is a repair process that uses the repair process for one constraint to repair a set of constraints in parallel. To do this, the conflict graph for all graphs of all constraints must be acyclic, i.e. there is no circular conflict in set of all graphs of all constraints. Then, we are able repair all constraints using Algorithm 2. The challenge of this approach is to decide which occurrences of which graphs need to be repaired in order to increase consistency.

References

[1] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. Henshin: advanced concepts and tools for in-place EMF model transformations. In *International Conference on Model Driven Engineering Languages and Systems*, pages 121–135. Springer, 2010.

[2] B. Becker, D. Beyer, H. Giese, F. Klein, and D. Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In *Proceedings of the 28th international conference on Software engineering*, pages 72–81, 2006.

[3] B. Becker, L. Lambers, J. Dyck, S. Birth, and H. Giese. Iterative development of consistency-preserving rule-based refactorings. In *International Conference on Theory and Practice of Model Transformations*, pages 123–137. Springer, 2011.

[4] N. Behr, R. Heckel, and M. G. Saadat. Efficient computation of graph overlaps for rule composition: Theory and z3 prototyping. *arXiv preprint arXiv:2003.11010*, 2020.

[5] E. Biermann, H. Ehrig, C. Ermel, U. Golas, and G. Taentzer. Parallel independence of amalgamated graph transformations applied to model transformation. *Graph Transformations and Model-Driven Engineering: Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, pages 121–140, 2010.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.

[7] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Fundamentals of algebraic graph transformation. *Monographs in theoretical computer science. An EATCS series. Springer*, 2006.

[8] H. Ehrig, A. Habel, and L. Lambers. Parallelism and concurrency theorems for rules with nested application conditions. *Electronic Communications of the EASST*, 26, 2010.

[9] A. Habel and K.-H. Pennemann. Nested constraints and application conditions for high-level structures. In *Formal methods in software and systems modeling*, pages 293–308. Springer, 2005.

[10] A. Habel and K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.

[11] R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars-a constructive approach. *Electronic Notes in Theoretical Computer Science*, 2:118–126, 1995.

[12] J. Kosiol, D. Strüber, G. Taentzer, and S. Zschaler. Sustaining and improving graduated graph consistency: A static analysis of graph transformations. *Science of Computer Programming*, 214:102729, 2022.

[13] L. Lambers, H. Ehrig, and F. Orejas. Conflict detection for graph transformation with negative application conditions. In *Graph Transformations: Third International Conference, ICGT 2006 Natal, Rio Grande do Norte, Brazil, September 17-23, 2006 Proceedings 3*, pages 61–76. Springer, 2006.

[14] N. Nassar, J. Kosiol, T. Arendt, and G. Taentzer. Constructing optimized constraint-preserving application conditions for model transformation rules. *Journal of Logical and Algebraic Methods in Programming*, 114:100564, 2020.

[15] N. Nassar, J. Kosiol, and H. Radke. Rule-based repair of EMF models: formalization and correctness proof. In *Electronic Pre-Proc. Intl. Workshop on Graph Computation Models*, 2017.

[16] N. Nassar, H. Radke, and T. Arendt. Rule-based repair of EMF models: An automated interactive approach. In *International Conference on Theory and Practice of Model Transformations*, pages 171–181. Springer, 2017.

[17] D. Plump. Confluence of graph transformation revisited. In *Processes, Terms and Cycles: Steps on the Road to Infinity*, pages 280–308. Springer, 2005.

[18] C. Sandmann and A. Habel. Rule-based graph repair. *arXiv preprint arXiv:1912.09610*, 2019.

[19] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.