

Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
AG Softwaretechnik

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

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science

vorgelegt von
Alexander Lauer
Matrikel-Nr: 2658100

26. Februar 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 26. 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 at some point be repaired. This problem is called *model repair*. In particular, we will consider *rule-based graph repair* which 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 transform G into a graph that satisfies c .

Known notions of consistency have either viewed consistency as a binary property, either a graph is consistent w.r.t. a constraint c or not, or only viewed the number of violations of the first graph of a constraint. In this thesis, we introduce new notions of consistency, which we call *consistency-maintaining* and *consistency-increasing* transformations and rules, respectively. This is based on the possibility that a constraint can be satisfied up to a certain nesting level. Our notion considers the graphs of the first unsatisfied nesting level and is more fine-grained than those already known. Finer in the sense that also 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-increasing* 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 at layer*, respectively. 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*. These application conditions for basic rules are less complex and restrictive than the ones for general rules. The notion of direct consistency-increasing at layer is a weaker version of direct consistency-increasing which allows the construction of less complex application conditions.

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*. Intuitively, a constraint c is *circular conflict free*, if there is an ordering C_0, \dots, C_n of all graphs of c such that there is no $j < i$ such that a repair of C_i leads to the insertion of a new violation of C_j . Analogously, a set of constraint \mathcal{C} is *circular conflict free*, if there is an ordering c_1, \dots, c_n of all constraints of \mathcal{C} such that there is no $j < i$ such that a repair of c_i at all graphs satisfying c_j leads to a graph not satisfying c_j .

Zusammenfassung

Model-driven software engineering ist eine geeignete Methode, um die ständig wachsende Komplexität von Softwareentwicklungsprozessen zu bewältigen. Graphen und Graph-

transformationen 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 Modells jedoch zu einem inkonsistenten Modell führen, welches wieder in ein konsistentes Modell überführt werden muss. Dieses Problem heißt *model repair*. Insbesondere betrachten wir das *rules-based graph repair* Problem, welches folgendermaßen definiert ist: 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.

Bereits bekannte Konzepte von Konsistenz haben Konsistenz entweder als binäre Eigenschaft, entweder ein Graph ist konsistent oder nicht, oder nur den ersten Graphen eines constraints betrachtet. In dieser Arbeit führen wir neue Begriffe von Konsistenz ein, die wir *consistency-maintaining* bzw. *consistency-increasing* Transformationen und Regeln nennen. Diese basieren auf der Möglichkeit, dass constraints bis zu einem bestimmten nesting level erfüllt sein können. Unsere Begriffe betrachten daher die Graphen des ersten nicht erfüllten nesting levels und sind feiner im Vergleich zu den 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 Anwendungsbedingungen zu konstruieren, die *direct consistency-maintaining* bzw. *direct consistency-increasing at layer* 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. Der Begriff von *direct consistency-increase rules at layer* ist eine schwächere Version von *direct consistency-increase rules* und führt zu weniger komplexen Anwendungsbedingungen.

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. Intuitiv ist ein constraint c *circular conflict free*, wenn es eine Ordnung C_0, \dots, C_n von allen Graphen von c gibt, sodass kein $j < i$ existiert, so dass eine Reparatur eines Vorkommens von C_i zur Einfügung einer neuen Verletzung von C_j führen kann. Analog ist eine Menge von constraints \mathcal{C} *circular conflict free*, wenn es eine Ordnung c_1, \dots, c_n von allen constraints von \mathcal{C} gibt, so dass kein $j < i$ existiert, so dass die Reparatur von c_i an einem Graphen, der bereits c_j erfüllt, nicht zu einem Graphen führen kann, der c_j nicht erfüllt.

Contents

1	Introduction	5
2	Preliminaries	7
2.1	Graphs and Graph morphisms	7
2.2	Nested Graph Conditions and Constraints	9
2.3	Rules and Graph Transformations	11
2.4	Concepts of Consistency	15
3	Consistency Increase and Maintainment	16
3.1	Universally quantified ANF	16
3.2	Satisfaction up to Layer	17
3.3	Consistency Increasing and Maintaining Transformations and Rules	23
3.4	Direct Consistency Maintaining and Increasing Transformations	27
3.5	Comparison with other concepts of Consistency	35
4	Consistency-Maintaining and Increasing Application Conditions	38
4.1	Application Condition for general Rules	38
4.2	Basic Consistency-increasing and Consistency-maintaining Rules	48
4.3	Application Conditions for Basic Rules	54
5	Rule-based Graph Repair	55
5.1	Conflicts within Conditions	56
5.2	Repairing rule Sets	61
5.3	Rule-based Graph Repair for one Constraint	65
5.4	Rule-based Graph Repair for multiple Constraints	69
6	Related Work	73
7	Conclusion	74

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 in 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 the consistency must at some point be restored.

The problem of restoring this consistency is called *graph repair*: Given a constraint c and an inconsistent graph G , transform G into a graph that satisfies c . In particular, we will consider the problem of *rule-based graph repair*, which is defined as follows: Given a graph G , a constraint c and a set of rules \mathcal{R} , use the rules of \mathcal{R} to transform G into a graph that satisfies c . 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.

There are several rule-based graph repair approaches 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, these binary notions allow 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 them is consistency-preserving or consistency-guaranteeing 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, these notions allow an evaluation of the magnitude of the 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]. Nassar et al. have presented a repair approach to repair multiplicities for EMF models [16, 15].

None of these approaches is able to repair multiple constraints with a nesting level higher than 2, and the application conditions constructed are very complex.

In this thesis, we will introduce a rule-based graph repair process for certain constraints in alternating normal-form (ANF), so-called *circular conflict-free constraints*, and for sets of these constraints.

The main idea of our approach is to increase the consistency of a graph level by level. That is, given a constraint $c = \forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \dots)))$) and an inconsistent graph G , in the first step we repair the graph such that the derived graph satisfies $\forall(C_1, \exists(C_2, \text{true}))$.

In the next step, we repair the graph until it satisfies $\forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \text{true})))$)

and so on until finally, the derived graph satisfies c . Note that we always repair two nesting levels at a time, since $\forall(C_1, \text{true})$ is always satisfied, and the satisfaction of $\forall(C_1, \text{false})$ 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.

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 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 $\forall(C_1, \exists(C_2, \text{true}))$ but not $\forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \text{true}))))$ 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 $\forall(C_1, \exists(C_2, \forall(C_3, \exists(C_4, \text{true}))))$. 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 into the notions of *direct consistency-maintaining* and *direct consistency-increasing* transformations and rules, which can be expressed via second-order logic formulas. These notions completely prohibit the insertion of new violations. We will formally compare our newly introduced notions with those described above to ensure that they are indeed new notions of consistency, and to point out similarities and differences.

We will introduce weaker notions of (direct) consistency-maintaining and (direct) consistency-increasing rules, called (direct) consistency-maintaining rules at layer and (direct) consistency-increasing rules at layer. Intuitively, a rule is (direct) consistency-maintaining at layer or (direct) consistency-increasing at layer if all its applications to graphs satisfying the constraint up to that layer are (direct) consistency-maintaining or (direct) consistency-increasing with respect to that constraint. With these notions, we will be able to construct less complex application conditions. We will present direct consistency-maintaining and two types of direct consistency-increasing application conditions at layer. 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 direct consistency-increasing application conditions at layer that are less restrictive and less complex compared to the general ones. We will show that each of the constructed application conditions produces consistency-maintaining rules or consistency-increasing rules at layer.

We introduce the notion of *circular conflict-free constraints* and *circular conflict-free sets of constraints*. Intuitively, a constraint c is *circular conflict free*, if there is an ordering C_0, \dots, C_n of all graphs of c such that there is no $j < i$ such that a repair of C_i leads to the insertion of a new violation of C_j . Analogously, a set of constraint \mathcal{C} is *circular conflict free*, if there is an ordering c_1, \dots, c_n of all constraints of \mathcal{C} such that there is no $j < i$ such that a repair of c_i at all graphs satisfying c_j leads to a graph not satisfying c_j .

Finally, we present two rule-based graph repair approaches. The first is for a *circular conflict-free* constraint and the second for a *circular conflict-free set of constraints*, which uses the 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. We will use the notions of *consistency-maintainment* and *consistency-increasement* to characterise sequences of transformations, that are able to repair an occurrence of a graph C_k without introducing new violations of certain other graphs of the constraint, and in particular of C_k itself. 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} . In addition, we show the correctness and termination of our approach.

This thesis 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. 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, [7]). A graph $G = (V, E, \text{src}, \text{tar})$ consists of a set of vertices (or nodes) V , a set of edges E and two mappings $\text{src}, \text{tar} : E \rightarrow V$ that assigns the source and target vertices to an edge. The edge $e \in E$ connects the vertices $\text{tar}(e)$ and $\text{src}(e)$.

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 thesis, 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, [7]). Let graphs G and H be given. A graph morphism $f : G \rightarrow H$ consists of two mappings $f_V : V_G \rightarrow V_H$ and $f_E : E_G \rightarrow E_H$ such that the source and target functions are preserved. This means

$$\begin{aligned} f_V \circ \text{src}_G &= \text{src}_H \circ f_E \text{ and} \\ f_V \circ \text{tar}_G &= \text{tar}_H \circ f_E \end{aligned}$$

holds. A graph morphism f is called injective (surjective) if f_E and f_V are injective (surjective) mappings. An injective morphism $f : G \rightarrow H$ 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 $f_1 : G_1 \rightarrow H$ and $f_2 : G_2 \rightarrow H$ 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 maintenance, 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 morphism 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 $\text{IG}(G, H)$.

Definition 2.5 (overlap). Let the graphs G_1 and 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 $\text{ol}(G_1, G_2)$. If a tuple as above is not given, then G_P , $i_{G_1}^P$ and $i_{G_2}^P$ denote the graph and morphisms of a given overlap $P \in \text{ol}(G_1, G_2)$.

Note that (H, i_{G_1}, i_{G_2}) where i_{G_1} and i_{G_2} are 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 thesis 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 *restrictions of morphisms*, which intuitively is the restriction of the domain and codomain 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 $f : G \rightarrow H$ be given. Then, a morphism $f' : G' \rightarrow H'$ is called a restriction of f if inclusions $i : G' \hookrightarrow G$ and $i' : H' \hookrightarrow H$ exist, i.e. G' is a subgraph of G and H' is a subgraph of H , such that

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

A restriction of f is denoted by f^r .

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

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,[10]). 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 an 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 *false* = $\neg \text{true}$.

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

Definition 2.8 (semantic of graph conditions,[10]). 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. $c = \text{true}$.
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 empty 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),[18]). 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 a universal condition over C_1 in ANF or $d = \text{true}$.
3. $\forall(a_0 : C_0 \hookrightarrow C_1, d)$ is a condition in ANF if either d is an existential condition over C_1 in ANF or $d = \text{false}$.

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 subconditions 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 $\text{nl}(c)$ of a condition c is recursively defined as $\text{nl}(\text{true}) = \text{nl}(\text{false}) = 0$ and $\text{nl}(\exists(a : P \hookrightarrow Q, d)) = \text{nl}(\forall(a : P \hookrightarrow Q, d)) := \text{nl}(d) + 1$.

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

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,[10]). 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 $\text{Shift}(c, i)$, is given by

1. If $c = \text{true}$, $\text{Shift}(c, i) = \text{true}$.
2. If $c = \exists(a_1 : C_0 \hookrightarrow C_1, d)$, $\text{Shift}(c, i) = \bigvee_{(a', i') \in \mathcal{F}} \exists(a', \text{Shift}(d, i'))$ 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 commutes.
3. If $c = \neg d$, $\text{Shift}(c, i) = \neg \text{Shift}(d, i)$

$$\begin{array}{ccc}
C_0 & \xhookrightarrow{i} & C'_0 \\
\downarrow a_0 & & \downarrow a'_0 \\
C_1 & \xhookrightarrow{i'} & C'_1
\end{array}$$

Figure 1: Diagram for the Shift operator.

- 4. If $c = d_1 \wedge d_2$, $\text{Shift}(c, i) = \text{Shift}(d_1, i) \wedge \text{Shift}(d_2, i)$
- 5. If $c = d_1 \vee d_2$, $\text{Shift}(c, i) = \text{Shift}(d_1, i) \vee \text{Shift}(d_2, i)$

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 $m : C'_0 \hookrightarrow G$,*

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

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 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. Intuitively, for every edge $e \in E_H$ there are vertices $u, v \in V_H$ such that $\text{tar}(e) = u$ and $\text{src}(e) = v$ 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,[7]). A plain rule $\rho = L \xleftarrow{l} K \xrightarrow{r} R$ consists of graphs L, K, R and inclusions $l : K \hookrightarrow L$ and $r : K \hookrightarrow R$. The rule $\rho^{-1} = R \xleftarrow{r} K \xrightarrow{l} L$ is called the inverse rule of ρ .

An application condition is a nested condition over L or R respectively. A rule $(\text{ap}_L, \rho, \text{ap}_R)$ 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,[7]). Let a rule $\rho = (\text{ap}_L, \rho', \text{ap}_R)$, a graph G and a morphism $m : L \hookrightarrow G$, called the match, be given. Then, a graph transformation, denoted by $t : G \Longrightarrow_{\rho, m} H$, is given in Figure 2 if the squares (1) and (2) are pushouts in the sense of category theory, $m \models \text{ap}_L$ and the morphism $n : L \hookrightarrow H$, called the

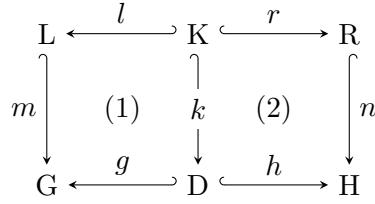


Figure 2: Diagram of a transformation in the double-pushout approach.

co-match of t , satisfies ap_R . The morphisms $g : D \rightarrow G$ and $h : D \rightarrow H$ are called the transformations morphisms of t .

The presence of right application 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,[10]). Let a rule $\rho = L \xleftarrow{l} K \xleftarrow{r} R$ with the right application condition ap be given. The shift of ap over ρ , denoted with $\text{Left}(\text{ap}, \rho)$, is defined as

1. If $\text{ap} = \text{true}$, $\text{Left}(\text{ap}, \rho) := \text{true}$.
2. If $\text{ap} = \neg d$, $\text{Left}(\text{ap}, \rho) := \neg \text{Left}(d, \rho)$.
3. If $\text{ap} = d_1 \wedge d_2$, $\text{Left}(\text{ap}, \rho) := \text{Left}(d_1, \rho) \wedge \text{Left}(d_2, \rho)$.
4. If $\text{ap} = d_1 \vee d_2$, $\text{Left}(\text{ap}, \rho) := \text{Left}(d_1, \rho) \vee \text{Left}(d_2, \rho)$.
5. If $\text{ap} = \exists(a_0 : R \rightarrow C_1, d)$, $\text{Left}(\text{ap}, \rho) := \exists(a'_0 : L \rightarrow C'_0, \text{Left}(d, \rho'))$ where $\rho' = C_0 \xleftarrow{g} D \xleftarrow{h} C'_0$ is the rule shown in Figure 3 which is derived by applying ρ^{-1} at match a_0 . If this transformation does not exist, we set $\text{Left}(\text{ap}, \rho) := \text{false}$.

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 $\text{Left}(\text{ap}, \rho)$ if and only if the co-match satisfies ap [10].

Lemma 2.15 ([10]). Let a plain rule $\rho = L \xleftarrow{l} K \xleftarrow{r} R$, a right application condition ap for ρ and a transformation $t : G \Rightarrow_{\rho, m} H$ be given. Then,

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

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) . The following Theorem shows, under which conditions a rule ρ is applicable at a match m , i.e. there is transformation via ρ at match m .

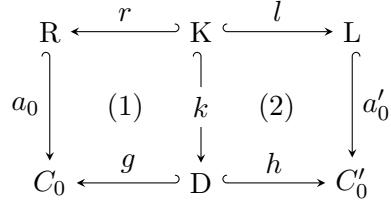


Figure 3: Transformation for the shift over rule operator.

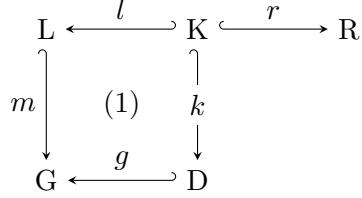


Figure 4: Applicability of a rule $L \xleftarrow{l} K \xleftarrow{r} R$ at match m .

Theorem 2.1 ([7]). *Given a rule $\rho = (\text{ap}, L \xleftarrow{l} K \xleftarrow{r} R)$, a graph G and a match $m : L \hookrightarrow G$ such that $m \models \text{ap}$. The rule ρ is applicable at match m , i.e. there is a transformation $t : G \Rightarrow_{\rho, m} H$ if and only if there is a graph D such that the square (1) in Figure 4 is a pushout. This is equivalent to ρ and m satisfying the dangling-edge condition, i.e. $m \models \text{ap}$ and*

$$IP_{\rho, m} \cup DP_{\rho, m} \subseteq GP_{\rho, m}$$

where GP is the set of all nodes and edges in L that are not deleted by ρ , i.e.

$$GP_{\rho, m} = l(K),$$

IP is the set of nodes and edges in L that are identified by m , i.e.

$$\begin{aligned}
IP_{\rho, m} = & \{v \in V_L \mid \exists v' \in V_L (v' \neq v \text{ and } m_v(v) = m_V(v'))\} \cup \\
& \{e \in E_L \mid \exists e' \in E_L (e \neq e' \text{ and } m_v(e) = m_V(e'))\},
\end{aligned}$$

and DP is the set of nodes in L whose images under m are the source or target of an edge in G that does not belong to $m(L)$, i.e.

$$DP_{\rho, m} = \{v \in V_L \mid \exists e \in E_G \setminus m_E(E_L) (\text{src}(e) = m_V(v) \text{ or } \text{tar}(e) = m_V(v))\}.$$

Since we will only consider injective matches, the set $IP_{\rho, m}$ is always empty, and therefore $DP_{\rho, m} \subseteq GP_{\rho, m}$ is sufficient to state that the *dangling-edge condition* is satisfied.

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

Definition 2.16 (track morphism,[17]). Consider the transformation t shown in figure 2. The track morphism of t , denoted by $\text{tr}_t : G \dashrightarrow H$, is a partial morphism defined as

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

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

Lemma 2.17 ([12]). Let a transformation $t : G \Rightarrow H$ 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. $\text{tr}_t \circ p$ 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 $\text{tr}_t(G)$ 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 $t : G \Rightarrow H$ 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 $\text{tr}_t : G \dashrightarrow H$ is total when restricted to $p_i(C_i)$. Then, $p'_{i-1} = p'_i \circ a_i$ where $p'_{i-1} = \text{tr}_t \circ p_{i-1}$ and $p'_i = \text{tr}_t \circ p_i$. 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 $\text{tr}_t(G)$, then $p_{i-1} = p_i \circ a_i$ where $p_{i-1} = \text{tr}_t^{-1} \circ p'_{i-1}$ and $p_i = \text{tr}_t^{-1} \circ p'_i$.

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 2.19 (concurrent rule,[8]). Let the rules $\rho_1 = L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1$, $\rho_2 = L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2$ and a sequence of transformations

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

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

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

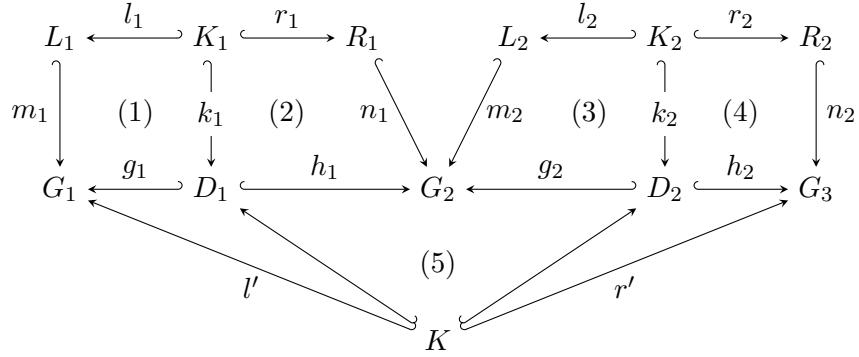


Figure 5: Pushout diagram of the transformation sequence $G_1 \xrightarrow{\rho_1, m_1} G_2 \xrightarrow{\rho_2, m_2} G_3$ using the rules $\rho_1 = L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1$ and $\rho_2 = L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2$.

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 2.20 (consistency preserving and guaranteeing transformations,[10]). Let a constraint c and a transformation $t : G \Rightarrow H$ be given. Then, t is called c -preserving if

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

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

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 2.21 (consistency sustaining and improving transformations,[12]). Let a constraint c in ANF and a transformation $t : G \Rightarrow H$ 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 universal, 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 is 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 2.22 (direct sustaining and improving transformations,[12]). Let a constraint c in ANF and a transformation $t : G \Rightarrow_{\rho} H$ be given. If c is existential, 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

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

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

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

3 Consistency Increase and Maintenance

In the following we will introduce the notions of *satisfaction up to 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 $\text{lay}(d) := \text{nl}(c) - \text{nl}(d)$.

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 maintenance 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 condition 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 $0 \leq k \leq \text{nl}(c)$ 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 $\exists(\text{id}_{C_0} : C_0 \hookrightarrow C_0, d)$ [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 universal, then c is already in UANF. If $c = \exists(a_0 : C_0 \hookrightarrow C_1, d)$, we show that c is equivalent to $c' := \forall(\text{id}_{C_0} : C_0 \hookrightarrow C_0, c)$.

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 $p = p \circ \text{id}_{C_0}$ 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 $p = q \circ \text{id}_{C_0}$ satisfy c . Since $p = p \circ \text{id}_{C_0}$, it immediately follows that $p \models c$. \square

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 $\exists(\text{id}_{C_0} : C_0 \hookrightarrow C_0, \forall(a_1 : C_0 \hookrightarrow C_2, d))$ is equivalent to $\forall(a_1 : C_0 \hookrightarrow C_2, d)$ and that $\forall(\text{id}_{C_0} : C_0 \hookrightarrow C_0, \exists(a_1 : C_0 \hookrightarrow C_2, d))$ 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 $\forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1 : C_1 \hookrightarrow C_2, \dots))$.

3.2 Satisfaction up to 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 up to 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* $-1 \leq k \leq \text{nl}(c)$ of a condition c . As the name suggests, the subcondition at layer $0 \leq k \leq \text{nl}(c)$ denotes the subcondition of c with layer k . We also define the subcondition at layer -1 , which is *true*. This will be useful for evaluating the satisfaction up to 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 $-1 \leq k \leq \text{nl}(c)$, denoted by $\text{sub}_k(c)$, is the subcondition d of c with $\text{lay}(d) = k$ if $0 \leq k \leq \text{nl}(c)$ and *true* if $k = -1$.*

$$c_1 = \forall(C_1^1, \exists(C_2^1, \text{true}))$$

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

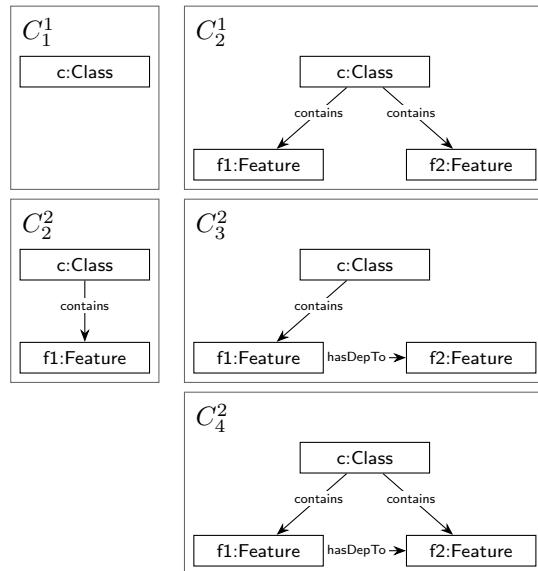


Figure 6: Constraints used for examples throughout the thesis.

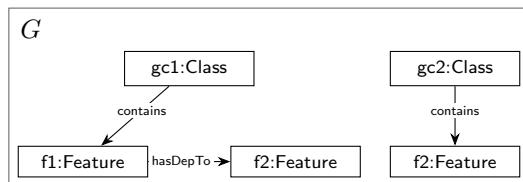


Figure 7: The graph used in example 3.4.

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 $c = \forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1 : C_1 \hookrightarrow C_2, \forall(a_2 : C_2 \hookrightarrow C_3, \text{false})))$. Then,

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

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

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

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

Example 3.2. Consider the conditions $c := \forall(a_0 : C_0 \hookrightarrow C_1, \exists(a_1 : C_1 \hookrightarrow C_2, \text{true}))$ and $e = \exists(a'_1 : C_1 \hookrightarrow C_3, d)$. The replacement starting from layer 1 in c by e is given by

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

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 $-1 \leq k < \text{nl}(c)$, denoted by $\text{cut}_k(c)$, is defined as

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

Example 3.3. Consider constraint c_2 given in Figure 6. The truncated conditions of c_2 after layer $-1 \leq k < 3$ are given by

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

Table 1: Overview of the inferences made about satisfaction up to layer, where “✓” indicates that $p \models_j c$ or $p \models c$ if $p \models_k c$ and “?” indicates that it cannot be inferred from $p \models_k c$ whether $p \models_j c$ or $p \not\models_j c$.

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

Note that the truncated condition of a condition c at layer $\text{nl}(c) - 1$ is c itself. With these prerequisites, we can now introduce *satisfaction up to 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 up to a layer if it satisfies the truncated condition after that layer.

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

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

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

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

Example 3.4. Consider the graph G given in Figure 7 and the constraint c_2 given in Figure 6. This graph does not satisfy c_2 because the occurrence of **Class** denoted with **gc2** does not satisfy $\text{sub}_1(c_2) = \exists(C_2^2, (\forall C_3^2, (\exists C_4^2, \text{true})))$, but it satisfies $\text{cut}_1(c_2) = \forall(C_1^1, \exists(C_2^2, \text{true}))$ and therefore

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

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

We start by examining the consequences for the satisfaction up to layer $\text{nl}(c) > j > k$. Our first lemma shows that replacing the subcondition $\text{sub}_{k+1}(c)$ 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 $p \models_k c$ where $-1 \leq k < \text{nl}(c)$ is even. Then, for any condition f over C_{k+1} it holds that*

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

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

Let $q : C_j \hookrightarrow G$ be a morphism such that $q \models \forall(a_j : C_j \hookrightarrow C_{j+1}, \text{false})$. This morphism must exist, since j is the smallest even number with $p \models_j c$. Therefore, there is no 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 $p \models \text{rep}_{j+1}(c, f)$.

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

As a direct consequence of the previous lemma, a morphism which satisfies a condition up to 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 $0 \leq k < \text{nl}(c)$ is even, i.e. $\text{sub}_k(c)$ is a universal condition, then for all $k < j < \text{nl}(c)$ 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 $\text{sub}_{k+1}(\text{cut}_j(c))$. \square

Since a morphism p satisfies a condition c in UANF if and only if p satisfies c up to layer $\text{nl}(c) - 1$, 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 $0 \leq k < \text{nl}(c)$ is even, it holds that*

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

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

Table 2: Overview of the satisfaction up to layer k with respect to 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 $p \not\models_j c$.

In the following, we will mostly assume that k_{\max} is odd. This is because an even 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 $-1 \leq k < \text{nl}(c)$. It follows that $p \models c$ and in particular $p \models c'$ for each condition c' with $\text{cut}_k(c) = \text{cut}_k(c')$.

Let us now examine the satisfaction up to layer j with $-1 < j < k$. If j is odd, i.e. $\text{sub}_j(c)$ is an existential condition, we can conclude that $p \models_j c$ as shown in the next lemma. If j is even, i.e. $\text{sub}_j(c)$ is universally bound, we can only make statements that depend on k_{\max} . If $k_{\max} < \text{nl}(c) - 1$, then $p \not\models_j c$. Otherwise, Corollary 3.10 implies that $p \models c$ and therefore $k_{\max} = \text{nl}(c) - 1$. If $k_{\max} = \text{nl}(c) - 1$, we can say that there is at least one even $j \leq k_{\max}$ with $p \models_j c$ if c ends with $\forall(C_{\text{nl}(c)}, \text{false})$. 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 $-1 \leq k \leq k_{\max}$, i.e. $\text{sub}_k(c)$ is an existential condition, we have*

$$p \models_k c.$$

Proof. If there is an even $0 \leq j < k_{\max}$, i.e. $\text{sub}_j(c)$ 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 $j' \leq \ell < \text{nl}(c)$. Otherwise we set $j' = k_{\max}$.

Let $\ell < j'$, such that $\text{sub}_\ell(c)$ is a existential condition and let $d = \text{sub}_\ell(\text{cut}_{j'}(c)) = \exists(a_\ell : C_\ell \hookrightarrow C_{\ell+1}, e)$ 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 $q \models \exists(a_\ell : C_\ell \hookrightarrow C_{\ell+1}, \text{true})$ and thus $p \models_\ell c$. \square

Example 3.5. *We will show counterexamples for all “?” in Table 1 and Table 2. Consider constraint $c_2 = \forall(C_1^1, \exists(C_2^2, \forall(C_3^2, \exists(C_4^2, \text{true})))$ given in Figure 6. We begin with Table 1.*

1. $j < k$, j and k are even. We set $k = 2$ and $j = 0$. It follows that $\text{cut}_k(c) = \forall(C_1^1, \exists(C_2^2, \forall(C_3^2, \text{false})))$ and $\text{cut}_j(c) = \forall(C_1^1, \text{false})$. Then,
 - (a) $C_2^2 \models_2 c_2$ and $C_2^2 \not\models_0 c_2$.
 - (b) $\emptyset \models_2 c_2$ 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 $\text{cut}_k(c_2) = c_2$ and $\text{cut}_j(c) = \forall(C_1^1, \text{false})$. Then,
 - (a) $C_4^2 \models_3 c_2$ and $C_4^2 \not\models_0 c_2$.
 - (b) $\emptyset \models_3 c_2$ 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 $\text{cut}_k(c) = \forall(C_1^1, \exists(C_2^2, \text{true}))$ and $\text{cut}_j(c) = \forall(C_1^1, \exists(C_2^2, \forall(C_3^2, \text{false})))$. Then,
 - (a) $C_2^2 \models_1 c_2$ and $C_2^2 \models_2 c_2$.
 - (b) $C_3^2 \models_1 c_2$ 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 $\text{cut}_k(c) = \forall(C_1^1, \exists(C_2^2, \text{true}))$ and $\text{cut}_j(c) = c_2$. Then,
 - (a) $C_2^2 \models_1 c_2$ and $C_2^2 \models_3 c_2$.
 - (b) $C_3^2 \models_1 c_2$ and $C_3^2 \not\models_3 c_2$.

Since $\text{cut}_j(c) = c_2$ this is also a counterexample for the “?” in the column denoted by “ $p \models c$ ”.

For the “?” in Table 2, the graph C_2^2 satisfies c_2 and therefore $k_{\max} = 3 = \text{nl}(c_2) - 1$. 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 up to layer, an increase of consistency can be detected in the following way: Let $t : G \implies H$ be a transformation. If the largest satisfied layer in H is greater than the largest satisfied layer in G , i.e. $k_{\max}(c, G) < k_{\max}(c, H)$, 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 leads to an increase of consistency, namely the insertion of a single edge or node of an existentially bound graph. To do this, 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 . 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 is replaced by an intermediate graph of the penultimate graph and the last graph of that truncated condition.

If c ends with an existential 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 universal condition, the opposite holds: The satisfaction of an intermediate condition implies the satisfaction of c . For this reason, we have designed intermediate conditions so that they only replace graphs on existential layers.

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

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

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

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

Proof. Assume that $G \models \text{cut}_k(c)$, i.e. $G \models_k c$. If there is an even $-1 \leq j < k$ such that $G \models_j c$, $G \models \text{IC}_k(c, C')$ 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 a restriction of p to the domain C . It follows that $G \models \text{IC}_k(c, C')$. \square

Example 3.6. Consider constraint c_1 given in Figure 6. Since $C_2^2 \in \text{IG}(C_1^1, C_2^1)$, we can construct an intermediate condition of c_1 at layer 1 with C_2^2 as $\text{IC}_1(c_1, C_2^2) = \forall(C_1^1 \exists(C_2^2, \text{true}))$. 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 *consistency-maintainment*, where increasement is a special case of maintenance. A transformation t is consistency-maintaining if it does not decrease consistency in the finer-grained sense as described above, while t is 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 $t : G \implies H$ is consistency-maintaining if the largest satisfied layer has not decreased, i.e. if $\text{k}_{\max}(c, G) \leq \text{k}_{\max}(c, H)$, 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 $C_{\text{k}_{\max}(c, G)+2}$ that does not satisfy $\exists(C_{\text{k}_{\max}(c, G)+3}, \text{true})$, an increasing insertion is the insertion of elements, such that for at least one occurrence p of $C_{\text{k}_{\max}(c, G)+2}$ it holds that $p \not\models \exists(C', \text{true})$ and $\text{tr}_t \circ p \models \exists(C', \text{true})$ for an intermediate graph $C' \in \text{IG}(C_{\text{k}_{\max}(c, G)+2}, C_{\text{k}_{\max}(c, G)+3})$. Decreasing insertions and deletions are the opposite of increasing ones. A decreasing insertion is the insertion of an occurrence of $C_{\text{k}_{\max}(c, G)+2}$ that does not satisfy $\exists(C_{\text{k}_{\max}(c, G)+3}, \text{true})$ and a decreasing deletion is the deletion of elements such that for one occurrence p of $C_{\text{k}_{\max}(c, G)+2}$ with $p \models \exists(C', \text{true})$ it holds that $\text{tr}_t \circ p \not\models \exists(C', \text{true})$ for an intermediate graph $C' \in \text{IG}(C_{\text{k}_{\max}(c, G)+2}, C_{\text{k}_{\max}(c, G)+3})$. If $\text{k}_{\max}(c, G) < \text{k}_{\max}(c, H)$ or the number of increasing insertions and deletions is greater than the number of decreasing ones, t is consistency-increasing.

To evaluate this, we define the *number of violations*. Intuitively, for all occurrences p of $C_{k_{\max}+2}$ the number of graphs $C' \in \text{IG}(C_{k_{\max}+2}, C_{k_{\max}+3})$ with $p \not\models \exists(C', \text{true})$ 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 $k \leq k_{\max}$ it is set to 0 and for all layers k with $k > k_{\max}+1$ it is set to ∞ . In this way, a transformation $t : G \implies H$ that increases the largest satisfied layer can be easily detected, since the number of violations in H at layer $k_{\max}+1$ will be set to 0.

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

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

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

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

In the following lemma, we will show that only these occurrences need to be considered.

Lemma 3.16. *Given a graph G , a constraint c in UANF and an odd $-1 \leq k < \text{nl}(c) - 2$ 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 $p = a_{k+1} \circ \dots \circ a_0 \circ q$, $a_i \circ \dots \circ q \models \text{sub}_{i+1}(\text{cut}_k(c))$ for all $0 \leq i \leq k$ and $q : \emptyset \hookrightarrow G$ is the empty morphism, it holds that $p \models \text{cut}_0(\text{sub}_{k_{\max}+2}(c))$.

Proof. Assume that $G \not\models_{k+2} c$ and for all occurrence $p : C_{k+2} \hookrightarrow G$ where $p = a_{k+1} \circ \dots \circ a_0 \circ q$ and $a_i \circ \dots \circ q \models \text{sub}_{i+1}(\text{cut}_k(c))$ for all $0 \leq i \leq k$ it holds that $p \models \text{cut}_0(\text{sub}_{k_{\max}+2}(c))$. 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 $p \not\models \text{cut}_0(\text{sub}_{k_{\max}+2}(c))$, $p = a_{k+1} \circ \dots \circ q$ and $a_i \circ \dots \circ q \models \text{sub}_{i+1}(\text{cut}_k(c))$ for all $0 \leq i \leq k$. This is a contradiction. \square

Only considering the occurrence of $C_{k_{\max}+2}$ as described above will lead to a more precise definition of the number of violations, and therefore to a more precise definition of consistency maintaining and increasing transformations and 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 *consistency-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 weaker notions, called *consistency-maintaining rules at layer* and *consistency-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 weaker notion will be important for our consistency-increasing application condition, since the graphs at the first unsatisfied layer must be considered, and constructing a consistency-increasing application condition will lead to huge 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

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

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

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

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 $-1 \leq k < \text{nl}(c)$ if all transformations $t : G \Longrightarrow_{\rho,m} H$ with $k_{\max}(c, G) = k$ are *consistency maintaining* w.r.t. c . Analogously, a rule ρ is called *consistency increasing* w.r.t. c at layer $-1 \leq k < \text{nl}(c)$ if all transformations $t : G \Longrightarrow_{\rho,m} H$ with $k_{\max}(c, G) = k$ 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 $\text{nv}_j(c, G) = 0$ for all $0 \leq j < \text{nl}(c)$. 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 consistency-increasing if the largest satisfied layer is increased. 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 $t : G \implies_{\rho,m} H$ is consistency-increasing w.r.t. c if*

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

Proof. There is no $\ell > k_{\max}(c, G)$ with $G \models_{\ell} c$. So, $nv_{k_{\max}(c,G)+1}(c, G) > 0$. Since $k_{\max}(c, H) > k_{\max}(c, G)$, it follows that $nv_{k_{\max}(c,G)+1}(c, H) = 0$, which immediately implies that t is consistency-increasing w.r.t. c . \square

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

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

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

We show that $G_x \models_{k_{\max}(c, G_0)+2} c$ holds after a maximum of $x := nv_{k_{\max}(c, G_0)+1}(c, G_0)$ transformations. Note that x must be finite, since G_0 contains only a finite number of occurrences of $C_{k_{\max}(c, G_0)+2}$. Since every transformation is consistency-increasing w.r.t. C , it follows that $nv_{k_{\max}(c, G_i)+1}(c, G_{i+1}) \leq nv_{k_{\max}(c, G_i)+1}(c, G_i) - 1$ after each transformation. Therefore, after at most x transformations, $nv_{k_{\max}(c, G_0)+1}(c, G_j) \leq nv_{k_{\max}(c, G_0)+1}(c, G_0) - x = 0$ and thus $G_x \models_{k_{\max}(c, G_0)+2} c$. 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 $G_k \implies_{\rho_k, m_k} G_{k+1}$, the sequence must be finite. \square

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 $t : G \xrightarrow{\rho, \text{id}_G} H$ via the rule $\rho = G \xleftarrow{l} \emptyset \xrightarrow{r} H$ 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 $\exists(C_{k+1}, \text{true})$ or inserting occurrences of universally bound graphs and inserting occurrences p of intermediate graphs $C' \in \text{IG}(C_{k-1}, C_k)$ such that each occurrence q of C_{k-1} with $q = p \circ a_{k-1}^r$ already satisfies $\exists(C', \text{true})$.

Direct consistency-increasing and *direct consistency-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 no 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 $C' \in \text{IG}(C_{k_{\max}+2}, C_{k_{\max}+3})$. This leads to the insertion of new violations only if an occurrence of $C_{k_{\max}+2}$ which satisfies $\exists(C', \text{true})$ in the originating graph does not satisfy $\exists(C', \text{true})$ 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 $C_{k_{\max}+2}$. Again, this will only cause a new violation if the new occurrence does not satisfy $\exists(C_{k_{\max}+3}, \text{true})$. 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 $j \leq k_{\max}$, 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 $j \leq k_{\max}$. 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* formulas ensure that the number of violations is not increased, and the *no satisfied layer reduction by insertion* and *no satisfied layer reduction deletion* formulas 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 $\forall(C_{\text{nl}(c)}, \text{false})$, the *no new violation by deletion* and *no new violation by insertion* formulas contain case discrimination. If the constraint c ends with $\forall(C_{\text{nl}(c)}, \text{false})$ and $k_{\max} = \text{nl}(c) - 2$, there is no graph $C_{k_{\max}+3}$ and thus no intermediate graphs. Therefore, there is no new violation by deletion, this formula is set equal to `true` and the *no new violation by insertion* formula will check that no new occurrence of $C_{k_{\max}+2}$ are introduced at all.

For the rest of this thesis, 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 $t : G \Rightarrow_{\rho,m} H$ is called direct consistency-maintaining w.r.t. c if $H \models c$. Otherwise, if $G \not\models c$, let $k_{\max} = k_{\max}(c, G)$ and $e = \text{sub}_{k_{\max}+2}(c)$. A transformation $t : G \Rightarrow_{\rho,m} H$ is called direct consistency maintaining w.r.t. c if the following formulas are satisfied.*

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

$$\begin{aligned} \forall p : C_{k_{\max}+2} \hookrightarrow G \left(\bigwedge_{C' \in \text{IG}(C_{k_{\max}+2}, C_{k_{\max}+3})} (p \models \text{IC}_0(e, C') \wedge \text{tr}_t \circ p \text{ is total}) \right. \\ \left. \implies \text{tr}_t \circ p \models \text{IC}_0(e, C') \right) \end{aligned} \quad (3.1)$$

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

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

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

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

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

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

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

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 $-1 \leq k < \text{nl}(c)$ if all transformations $t : G \Rightarrow_{\rho,m} H$ with $k_{\max}(c, G) = k$ 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 $t : G \Rightarrow H$ 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*

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

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

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

where $i, j \leq k_{\max}(c, G)$, 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. \square

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 $t : G \Rightarrow_{\rho,m} H$ w.r.t. c . Then, t is also a consistency-maintaining transformation.*

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

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

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

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

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 $\text{IC}_0(d, C_{k+2})$ if p satisfies $\text{IC}_0(d, C')$ for all $C' \in \text{IG}(C_{k+1}, C_{k+2})$. It follows that

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

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 , then 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 $-1 \leq k \leq \text{nl}(c)$, then 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 $\forall(C_{\text{nl}(c)}, \text{false})$ and $k_{\max} = \text{nl}(c) - 2$. 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 $C_{k_{\max}+2}$ that does not satisfy $\exists(C_{k_{\max}+3}, \text{true})$ is deleted, or an occurrence of $C_{k_{\max}+2}$ which does not satisfy $\exists(C', \text{true})$ in the first graph of the transformation satisfies $\exists(C', \text{true})$ in the second graph of the transformation where $C' \in \text{IG}(C_{k_{\max}+2}, C_{k_{\max}+3})$. Both cases result in the removal of a violation.
2. *Special increasing formula:* This formula is satisfied if an occurrence of $C_{k_{\max}+2}$ 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 $e = \text{sub}_{k_{\max}+2}(c)$.*

A transformation $t : G \Rightarrow_{\rho, m} H$ 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 $\text{sub}_{\text{nl}(c)-1}(c) = \forall(C_{\text{nl}(c)}, \text{false})$ and $k_{\max} = \text{nl}(c) - 2$ or the general increasing condition is satisfied otherwise.

1. General increasing formula:

$$\exists p : C_{k_{\max}+2} \hookrightarrow G \left(\bigvee_{C' \in \text{IG}(k_{\max}+2, k_{\max}+3)} (p \not\models \text{IC}_0(e, C') \wedge (\text{tr}_t op \text{ is not total} \vee \text{tr}_t op \models \text{IC}_0(e, C'))) \right) \quad (3.5)$$

2. Special increasing formula:

$$\exists p : C_{k_{\max}+2} \hookrightarrow G (\text{tr}_t op \text{ is not total}) \quad (3.6)$$

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 $-1 \leq k < \text{nl}(c)$ if all transformations $t : G \xrightarrow{\rho, m} H$ with $k_{\max}(c, G) = k$ 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 a consistency-increasing transformation. Counterexamples in which the inversion of the implication does not hold can be easily constructed to show 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 $t : G \xrightarrow{\rho, m} H$ 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 $\text{nv}_{k_{\max}(c, G)+1}(c, H) < \text{nv}_{k_{\max}(c, G)+1}(c, G)$. Let $k_{\max} = k_{\max}(c, G)$ and $d = \text{sub}_{k_{\max}+2}(c)$ with $d \neq \text{false}$.

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

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

Since t is direct consistency maintaining, it follows that

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

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

If $d = \text{false}$, i.e. $\text{sub}_{\text{nl}(c)-1}(c) = \forall(C_{\text{nl}(c)}, \text{true})$ and $k_{\max} = \text{nl}(c) - 2$, 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. \square

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 , then 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 $-1 \leq k \leq \text{nl}(c)$, then ρ is also a consistency-increasing rule w.r.t. c at layer k .*

Example 3.7. Consider constraint c_1 given in Figure 6, the transformations t_1 , t_2 and the set $\text{IG}(C_1^1, C_2^1)$ given in Figure 8. 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 c_1 does not satisfy $\exists(I_3, \text{true})$, $\exists(I_4, \text{true})$, $\exists(I_5, \text{true})$ and $\exists(I_6, \text{true})$, the occurrence c_2 does not satisfy $\exists(I_2, \text{true})$, $\exists(I_3, \text{true})$, $\exists(I_4, \text{true})$, $\exists(I_5, \text{true})$ and $\exists(I_6, \text{true})$. In the second graph, these roles are swapped, i.e. c_1 satisfies exactly the intermediate conditions that c_2 satisfied in the first graph, and vice versa. But, t_1 is not a direct consistency-maintaining transformation, since the occurrence c_1 satisfies $\exists(I_2, \text{true})$ 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 c_1 does not satisfy $\exists(I_3, \text{true})$, $\exists(I_4, \text{true})$, $\exists(I_5, \text{true})$ and $\exists(I_6, \text{true})$. Both occurrences c_2 and c_3 do not satisfy $\exists(I_2, \text{true})$, $\exists(I_3, \text{true})$, $\exists(I_4, \text{true})$, $\exists(I_5, \text{true})$ and $\exists(I_6, \text{true})$.

In the second graph, c_1 does not satisfy $\exists(I_2, \text{true})$, $\exists(I_4, \text{true})$, $\exists(I_5, \text{true})$ and $\exists(I_6, \text{true})$ and both c_2 and c_3 do not satisfy $\exists(I_6, \text{true})$. Therefore, the number of violations in the second graph is 6.

But t_2 is not a direct consistency increasing transformation, since c_1 satisfies $\exists(I_3, \text{true})$ in the first but not in the second graph, and the no new violation by deletion formula is not satisfied.

$$\text{IG}(C_1^1, C_2^1) := \{I_1, I_2, I_3, I_4, I_5, I_6\}$$

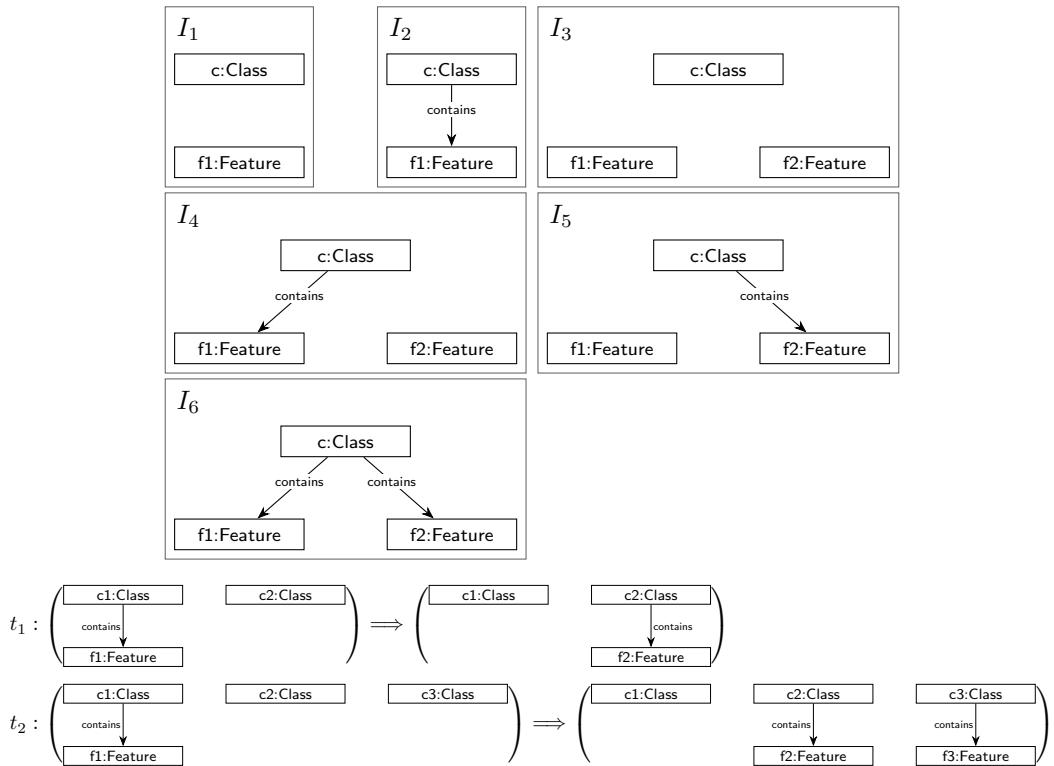


Figure 8: Example of the set of intermediate graph. Transformation t_1 is consistency-maintaining and not direct consistency-maintaining. Transformation t_2 is consistency-increasing and not direct consistency-increasing.

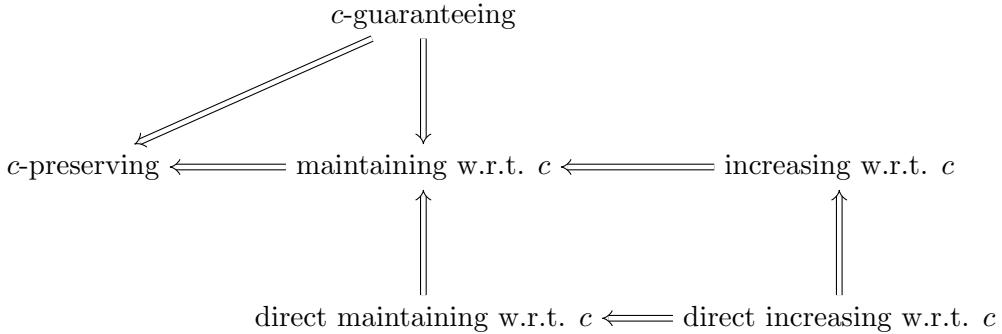


Figure 9: Relations of consistency notions.

3.5 Comparison with other concepts of Consistency

In this chapter, the notions of (direct) consistency increase and maintenance 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 maintenance are indeed new notions of consistency. These relationships are summarised in Figure 9.

First, we compare (direct) consistency increase and maintenance 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 $t : G \Rightarrow H$. Then,*

$$\begin{array}{lll} t \text{ is consistency-maintaining w.r.t. } c & \implies t \text{ is } c\text{-preserving} & \text{and} \\ t \text{ is consistency-maintaining w.r.t. } c & \nRightarrow t \text{ is } c\text{-guaranteeing} & \text{and} \\ t \text{ is direct consistency-increasing w.r.t. } c & \nRightarrow t \text{ is consistency sustaining w.r.t. } c & \end{array}$$

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 $\text{nv}_j(c, G) = 0$ for all $0 \leq j < \text{nl}(c)$. Since t is consistency maintaining it follows that $\text{nv}_j(c, H) = 0$ for all $0 \leq j < \text{nl}(c)$ and hence $H \models c$. It follows that t is a c -preserving transformation.

2. t is consistency maintaining w.r.t. $c \nRightarrow t$ is c -guaranteeing: Consider the transformation $t_2 : G \Rightarrow H$ shown in Figure 8 and constraint c_1 shown in Figure 6. As discussed in Example 3.7, t_2 is consistency-increasing and thus consistency-maintaining w.r.t. c_1 . But t_2 is not a c -guaranteeing transformation, since all occurrences of nodes of type `Class` do not satisfy $\exists(C_2^1, \text{true})$.

3. t is direct consistency maintaining w.r.t. $c \Rightarrow t$ 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 $d \neq \text{false}$ composed of the graphs given in Figure 6. And consider transformation t_2 given in Figure 10. 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 $\exists(C_2^1, \text{true})$ in G satisfies $\exists(C_2^1, \text{true})$ 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 $t : G \Rightarrow H$. Then,*

$$\begin{array}{ll} t \text{ is } c\text{-guaranteeing} & \Rightarrow t \text{ is consistency-maintaining w.r.t. } c \text{ and} \\ t \text{ is } c\text{-guaranteeing} & \not\Rightarrow t \text{ is consistency-increasing w.r.t. } c \text{ and} \\ t \text{ is } c\text{-preserving} & \not\Rightarrow t \text{ is consistency-maintaining w.r.t. } c \text{ and} \\ t \text{ is direct consistency improving w.r.t. } c & \not\Rightarrow t \text{ is consistency-maintaining w.r.t. } c \end{array}$$

Proof. 1. t is c -guaranteeing $\Rightarrow t$ 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 $\text{nv}_j(c, H) = 0$ for all $-1 \leq j < \text{nl}(c)$.

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

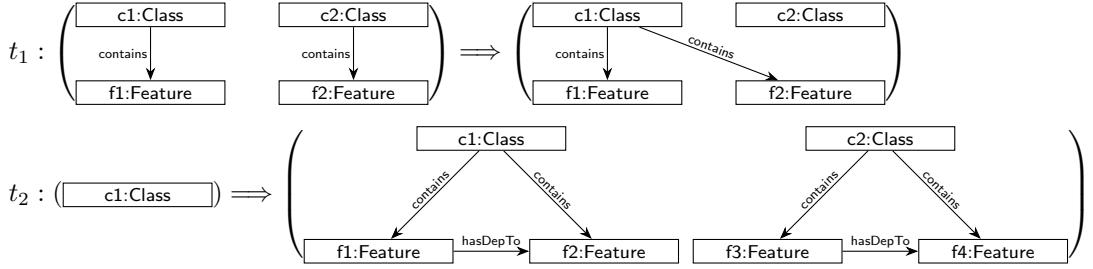


Figure 10: Transformations for the comparison of consistency concepts.

4. t is direct consistency-improving w.r.t. $c \not\Rightarrow t$ is consistency-maintaining w.r.t. c : Consider transformation t_1 given in Figure 10 and constraint c_1 given in Figure 6. The transformation t_1 is direct consistency-improving since no occurrence of C_1^1 is inserted, no occurrence of C_1^1 satisfying $\exists(C_2^1, \text{true})$ is deleted, and one occurrence of C_1 satisfies $\exists(C_2^1, \text{true})$. 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 maintenance are not related to (direct) consistency improvement and sustainment. We have shown only some of these relationships. Since (direct) consistency improvement (direct) consistency sustainment, consistency-preserving and consistency-guaranteeing 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 $t : G \Rightarrow H$.*

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

t is c -guaranteeing $\Rightarrow t$ is consistency-increasing w.r.t. c .

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

t is consistency improving w.r.t $c \iff t$ 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 $0 < \text{nv}_{k_{\max}(c,G)+1}(c, G) < \infty$ and $\text{nv}_j(c, H) = 0$ for all $-1 \leq j \leq \text{nl}(c)$.

2. Let $\text{nl}(c) = 1$. Since c is in UANF, $\text{sub}_1(c) = \text{false}$ and $\text{nv}_0(c, G)$ 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.

□

\implies	(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 every rule equipped with this application condition is (direct) consistency-increasing at layer, (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 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 rules equipped with these application conditions are direct consistency-maintaining and direct consistency-increasing rules at layer, respectively.

4.1 Application Condition for general Rules

We will now introduce consistency-maintaining and consistency-increasing application conditions at layer for general rules. This means that a rule equipped with these application conditions is consistency-maintaining at layer or consistency-increasing at layer. 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 at layer. This application condition has an odd parameter $-1 \leq k < \text{nl}(c)$ 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 k_{\max} even and $k_{\max} \neq \text{nl}(c) - 1$. In this case, it can be shown that the application condition with $k = \text{nl}(c) - 2$ produces a consistency-preserving application condition. So it is not a restriction that k must be

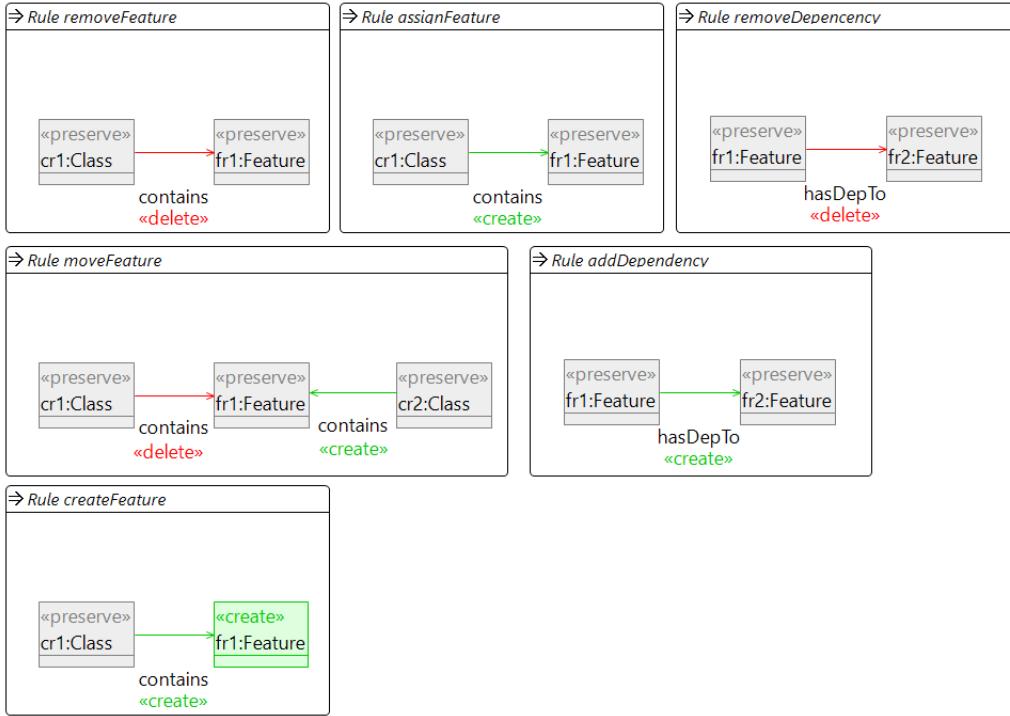


Figure 11: The rules used throughout the examples.

$$\begin{aligned}
 c = & \forall \left(\left(\boxed{\text{cc1:Class}} \right) \exists \left(\begin{array}{c} \boxed{\text{cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{cf2:Feature}} \end{array} \right) , \forall \left(\begin{array}{c} \boxed{\text{cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{cf2:Feature}} \end{array} \right) \xrightarrow{\text{hasDepTo}} \boxed{\text{cf3:Feature}} , \exists \left(\begin{array}{c} \boxed{\text{cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{cf2:Feature}} \end{array} \xrightarrow{\text{hasDepTo}} \boxed{\text{cf3:Feature}} , \text{true} \right) \right) \right) \\
 \text{ned_1} = & \text{true} \\
 \text{ned}_1 = & \neg \exists \left(\begin{array}{c} \boxed{\text{cr1 = cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{fr1 = cf2:Feature}} \end{array} , \text{true} \right) \\
 \text{ned}_3 = & \neg \exists \left(\begin{array}{c} \boxed{\text{rc1 = cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{fr1 = cf2:Feature}} \end{array} , \text{true} \right) \wedge \neg \exists \left(\begin{array}{c} \boxed{\text{rc1 = cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1 = cf1:Feature}} \quad \boxed{\text{cf2:Feature}} \end{array} \xrightarrow{\text{hasDepTo}} \boxed{\text{cf3:Feature}} , \text{true} \right) \wedge \\
 & \neg \exists \left(\begin{array}{c} \boxed{\text{rc1 = cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{fr1 = cf2:Feature}} \end{array} \xrightarrow{\text{hasDepTo}} \boxed{\text{cf3:Feature}} , \text{true} \right) \wedge \neg \exists \left(\begin{array}{c} \boxed{\text{cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{cf2:Feature}} \end{array} \xrightarrow{\text{hasDepTo}} \boxed{\text{fr1 = cf3:Feature}} , \text{true} \right)
 \end{aligned}$$

Figure 12: no existentially deleted conditions for the constraint and rule `removeFeature`.

odd. The maintaining application condition consists of the following three parts, which also use the k parameter:

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

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

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

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

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

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

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

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 $i_R^{P'}(R \setminus K) \cap i_{C_j}^{P'}(C_j \setminus C_{j-1}) \neq \emptyset$:

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

3. No existentially destroyed: If $k = -1$, we set $\text{ned}_k(\rho') := \text{true}$. 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 of all overlaps P' of L and C_j with $i_L^{P'}(L \setminus K) \cap i_{C_j}^{P'}(C_j \setminus C_{j-1}) \neq \emptyset$:

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

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

Example 4.1. 1. Consider the constraint c given in Figure 12 and the rule `removeFeature`. The conditions $\text{ned}_{-1}(\text{removeFeature})$, $\text{ned}_1(\text{removeFeature})$ and $\text{ned}_3(\text{removeFeature})$ are also given in Figure 12; $\text{ned}_{-1}(\text{removeFeature})$ is equal to `true`. $\text{ned}_1(\text{removeFeature})$ checks that no occurrences of C_2 are inserted, while $\text{ned}_3(\text{removeFeature})$ checks that no occurrences of C_2 and C_4 are inserted. Obviously, $\text{ned}_1(\text{removeFeature})$ is contained in $\text{ned}_3(\text{removeFeature})$. Note that there are conditions in $\text{ned}_3(\text{removeFeature})$ 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 12 and the rule `assignFeature`. The condition $\text{nui}_{-1}(\text{assignFeature})$ is equal to `true` since `assignFeature` cannot create any occurrences of the first universally bound graph of the constraint. The condition $\text{nui}_1(\text{assignFeature})$ is given in Figure 13.

The condition $\text{nui}_3(\text{assignFeature})$ is equal to $\text{nui}_1(\text{assignFeature})$ since there are only two universally bound graphs and $\text{nui}_1(\text{assignFeature})$ already considers both.

3. Consider the constraint c given in Figure 14 and the rule `addDependency`. Then, $\text{vio}_{-1}(\text{addDependency})$ is given in Figure 14. The condition $\text{vio}_1(\text{addDependency})$ is equal to `true` since $1 > \text{nl}(c) - 2 = 2 - 2$.

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

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

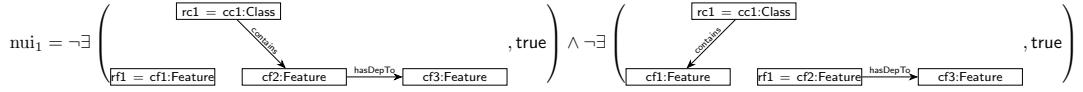


Figure 13: No universally inserted conditions for the constraint given in Figure 12 and rule `assignFeature`.

$$c = \forall ([\boxed{cc1:\text{Class}}], \exists ([\boxed{cc1:\text{Class}}] \xrightarrow{\text{contains}} \boxed{cf1:\text{Feature}} \xrightarrow{\text{hasDepTo}} \boxed{cf2:\text{Feature}} \xrightarrow{\text{hasDepTo}} \boxed{cf3:\text{Feature}}, \text{true}))$$

$$\text{vio}_{-1} = \neg \exists ([\boxed{cc1:\text{Class}}] \xrightarrow{\text{contains}} [\boxed{rf1 = cf1:\text{Feature}}] \xrightarrow{\text{hasDepTo}} [\boxed{rf2 = cf2:\text{Feature}}], \text{true})$$

$$\wedge \neg \exists ([\boxed{cc1:\text{Class}}] \xrightarrow{\text{contains}} \boxed{cf1:\text{Feature}} \xrightarrow{\text{hasDepTo}} \boxed{rf1 = cf2:\text{Feature}} \xrightarrow{\text{hasDepTo}} \boxed{rf2 = cf3:\text{Feature}}, \text{true})$$

Figure 14: No violation inserted conditions for the constraint c and rule `removeDependency`.

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

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

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 $\text{tr}_t \circ p = p'$ exists. Therefore, there is an overlap P of R and C_{k+2} with $i_R^P(R \setminus K) \cap i_{C_{k+2}}^P(C_{k+2} \setminus C_{k+1}) \neq \emptyset$ such that $m \models \text{Left}(\exists(i_R^P : R \hookrightarrow P, \text{true}), \rho')$. Hence, m does not satisfy $\text{nui}_k(\rho')$.

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 $\text{tr}_t \circ p' = p$ exists. Then, there is an overlap P of C_j and R with $i_R^P(R \setminus K) \cap i_{C_j}^P(C_j \setminus C_{j-1}) \neq \emptyset$ such that $m \models \text{Left}(\exists(i_R^P : R \hookrightarrow P, \text{true}), \rho)$. Hence, $m \not\models \text{nui}_k(\rho')$.

4. Assume that t does not satisfy the *no satisfied layer reduction by deletion* formula. Then, there is a morphism $p : C_j \hookrightarrow G$ with $j \leq k + 1$ and C_j existentially bound and such that $\text{tr}_t \circ p$ 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 $m \models \exists(i_L^P : L \hookrightarrow P, \text{true})$. Hence, $m \not\models \text{ned}_k(\rho')$.

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

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*

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

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

Proof. Let $\rho = R \xleftarrow{r} K \xrightarrow{l} L$ 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 $-1 \leq k \leq \text{nl}(c) - 1$. Obviously, $\text{nui}_{\text{nl}(c)-1}(\rho)$ contains $\text{nui}_j(\rho)$ for all $-1 < j \leq \text{nl}(c) - 1$. The set of intermediate graphs always contains the second graphs on which this set was built, so $\text{vio}_i(\rho)$ contains the condition

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

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

$$\left(\bigwedge_{\substack{-1 \leq i < \text{nl}(c) \\ i \text{ odd}}} \text{vio}_i(\rho) \right)$$

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

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

It follows that ρ is a direct consistency-maintaining rule at layer k for all $-1 \leq k < \text{nl}(c)$. Since $-1 \leq k_{\max} < \text{nl}(c)$ 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 . \square

In the following, we will introduce the direct consistency increasing application conditions at layer. For this, we will introduce the notion of *extended overlap*, which will be useful to detect whether a violation is removed by a transformation. Intuitively, given an overlap and a morphism a , the overlap is extended such that an overlap morphism satisfies $\exists(a, \text{true})$.

$$\begin{array}{ccc}
C_0 & \xleftarrow{e} & H \\
\downarrow i_{C_0}^G & & \downarrow i_H^P \\
G & \xleftarrow{i_G^P} & P
\end{array}$$

Figure 15: Diagram for the alternative definition of extended overlaps.

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

$$\text{eol}(G, i_{C_0}^G, e) := \{P \in \text{ol}(G, H) \mid i_G^P \circ i_{C_0}^G \models \exists(e : C_0 \hookrightarrow H, \text{true})\}.$$

In other words, $\text{eol}(G, i_{C_0}^G, e)$ is the set of all overlaps of G and H such that the square in Figure 15 is commutative, i.e. $i_H^P \circ e = i_G^P \circ i_{C_0}^G$. Using extended overlaps, we will be able to check whether a violation has been removed.

For the consistency-increasing application condition at layer, we will use the maintaining application condition at layer. 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 $-1 \leq k < \text{nl}(c)-1$, which specifies which constraint graphs are to be considered. Note that k must not be $\text{nl}(c)-1$, since all graphs with $k_{\max} = \text{nl}(c)-1$ satisfy the constraint and no consistency-increasing transformations are originating from those graphs. We also use 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. $k < \text{nl}(c)-2$, and C' is set to C_{k+2} otherwise. 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 ($\text{main}_k()$): 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* ($\text{exv}()$): 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 $\exists(C', \text{true})$. If $k = \text{nl}(c)-2$, then c ends with $\forall(C_{\text{nl}(c)}, \text{false})$ 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* ($\text{remv}()$): 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 $p \models \exists(C', \text{true})$. 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 $\exists(C', \text{true})$ after the transformation. The last case is the special case where the constraint ends with $\forall(C_{\text{nl}(c)}, \text{true})$ and $k = \text{nl}(c) - 2$. 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 at layer). Given a rule $\rho = (\text{ac}, \rho')$ with $\rho' = L \xleftarrow{l} K \xrightarrow{r} R$ and a constraint c in UANF. Let $0 \leq k < \text{nl}(c) - 1$ be odd and $C' = C_{k+2}$ if $k = \text{nl}(c) - 2$ and $C' \in \text{IG}(C_{k+2}, C_{k+3})$ otherwise. The increasing application condition of c for ρ at layer k with C' is defined as

$$\text{incr}_k(C', \rho) := \text{ac} \wedge \text{main}_k(\rho') \wedge \left(\bigvee_{P \in \text{eol}(L, C_{k+2})} \text{exv}(P, C') \wedge \text{remv}(P, C') \right) \quad (4.1)$$

with

1. Violation exists: Let $a_{k+2}^r : C_{k+2} \hookrightarrow C'$ be the restricted morphism of a_{k+2} and i_L^P and i_P^Q be overlap morphisms of P and Q , respectively:

$$\text{exv}(P, C') := \begin{cases} \exists(i_L^P : L \hookrightarrow P, \text{true}) & \text{if } \text{sub}_{k+2}(c) = \text{false} \\ \exists(i_L^P : L \hookrightarrow P, \bigwedge_{Q \in \text{eol}(P, i_{C_{k+2}}^P, a_{k+2}^r)} \neg \exists(i_P^Q : P \hookrightarrow Q, \text{true})) & \text{otherwise} \end{cases}$$

2. Violation removed:

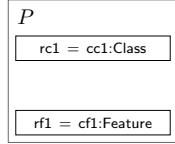
- (a) If $i_L^P(L \setminus K) \cap i_{C_{k+2}}^P(C_{k+2} \setminus C_{k+1}) \neq \emptyset$, $\text{remv}(P, C') := \text{true}$.
- (b) If $\text{sub}_{k+2}(c) = \text{false}$, i.e. $k = \text{nl}(c) - 2$,

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

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

$$\text{remv}(P, C') := \text{Left}(\forall(i_R^Q : R \hookrightarrow P', \bigvee_{Q \in \text{eol}(P', i_{C_{k+2}}^{P'}, a_{k+2}^r)} \exists(i_{P'}^Q : P' \hookrightarrow Q, \text{true})), \rho).$$

Example 4.2. 1. Consider constraint c_1 given in Figure 6 and the rule `assignFeature`. There is only one overlap P of L and C_1^1 which is shown in Figure 16. The $\text{exv}(P, C')$ and $\text{exv}(P, C')$ parts of $\text{incr}_{-1}(C', \text{assignFeature})$ with $C' = C_2^2$ and $C' = C_2^1$, respectively, are also given in Figure 16.



$$\begin{aligned}
 \text{exv}(P, C_2^2) &= \forall \left(\begin{array}{l} \boxed{\text{rc1} = \text{cc1:Class}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \end{array}, \neg \exists \left(\begin{array}{l} \boxed{\text{rc1} = \text{cc1:Class}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \end{array}, \text{true} \right) \right) \\
 \text{remv}(P, C_2^2) &= \forall \left(\begin{array}{l} \boxed{\text{rc1} = \text{cc1:Class}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \end{array}, \neg \exists \left(\begin{array}{l} \boxed{\text{rc1} = \text{cc1:Class}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \end{array}, \text{true} \right) \right) \\
 \text{exv}(P, C_2^1) &= \exists \left(\begin{array}{l} \boxed{\text{rc1} = \text{cc1:Class}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \end{array}, \neg \exists \left(\begin{array}{c} \boxed{\text{rc1} = \text{cc1:Class}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{cf2:Feature}} \end{array}, \text{true} \right) \wedge \neg \exists \left(\begin{array}{c} \boxed{\text{rc1} = \text{cc1:Class}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{f1} = \text{cf3:Feature}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{cf2:Feature}} \end{array}, \text{true} \right) \right) \\
 \text{remv}(P, C_2^1) &= \forall \left(\begin{array}{l} \boxed{\text{rc1} = \text{cc1:Class}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \end{array}, \neg \exists \left(\begin{array}{c} \boxed{\text{rc1} = \text{cc1:Class}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \\ \xrightarrow{\text{contains}} \\ \boxed{\text{cf2:Feature}} \end{array}, \text{true} \right) \right)
 \end{aligned}$$

Figure 16: Examples for $\text{exv}(P, C_2^2)$, $\text{exv}(P, C_2^1)$, $\text{remv}(P, C_2^2)$ and $\text{remv}(P, C_2^1)$ using the rule **assignFeature** and constraint c_1 given in Figure 6.

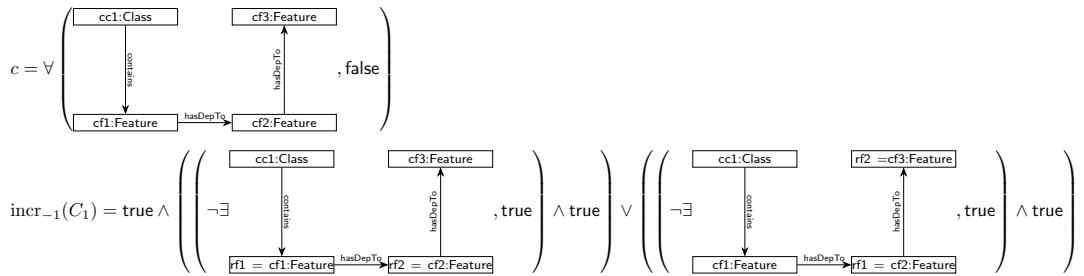


Figure 17: Example of $\text{incr}_{-1}(C_1, \rho)$ using this constraint and the rule **removeDependency**.

2. Consider the constraint and rule given in Figure 17. 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 $\rho = (\text{ac} \wedge \text{incr}_k(C', \rho'), \rho')$ with $-1 \leq k < \text{nl}(c) - 1$ odd and $C' \in \text{IG}(C_{k+2}, C_{k+3})$ if $k < \text{nl}(c) - 2$ and $C' = C_{k+2}$ otherwise is a direct consistency-increasing rule w.r.t. c at layer k .*

Proof. Let a transformation $t : G \implies_\rho H$ with $\text{k}_{\max}(c, G) = k$ be given. Since $\text{incr}_k(C', \rho)$ contains $\text{main}_k(\rho)$, 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 $k = \text{nl}(c) - 2$, i.e. c ends with a condition of the form $\forall(C_{\text{nl}(c)}, \text{false})$, assume that t does not satisfy the special increasing formula. Then there is no morphism $p : C_{k+2} \hookrightarrow G$ such that $\text{tr}_t \circ p$ is not total. So there is no overlap P of L and C_{k+2} with $i_L^P(L \setminus K) \cap i_{C_{k+2}}^P(C_{k+2} \setminus C_{k+1}) \neq \emptyset$ such that $m \models \exists(i_L^P : L \hookrightarrow P, \text{true})$. Since, $k = \text{nl}(c) - 2$ it follows that either $\text{remv}(P', C') = \text{false}$ or $m \not\models \text{exv}(P', C')$ for all $P \in \text{ol}(L, P)$. Therefore $m \not\models \text{incr}_k(C, \rho)$, this is a contradiction.
2. Otherwise let $P \in \text{ol}(L, C_{k+2})$. We show that $m \models \text{exv}(P, C') \wedge \text{remv}(P, C')$ implies that t satisfies the general increasing formula. If $m \models \text{exv}(P, C')$, there is a morphism $p : P \hookrightarrow G$ with $m = p \circ i_L^P$ and $p \models \neg \exists(i_P^Q : P \hookrightarrow Q, \text{true})$ for all $Q \in \text{eol}(P, i_{C_{k+2}}^P, a_{k+2}^r)$. Therefore, $q := p \circ i_{C_{k+2}}^P \not\models \exists(a_{k+2}^r : C_{k+2} \hookrightarrow C', \text{true})$.

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

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

In summary, ρ is a direct consistency-increasing rule at layer k . \square

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

$C'' \in \text{IG}(C_{k+2}, C')$ of C' , $\text{incr}_k(C', \rho)$ would be evaluated to `false`. For any larger improvements, i.e. the same improvement for a supergraph $C'' \in \text{IG}(C', C_{k+3})$ of C' , $\text{incr}_k(C', \rho)$ will also be evaluated to `false` if the repaired occurrence of C_{k+2} satisfies $\exists(C', \text{true})$. 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

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

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 $\text{incr}_k(C', \rho)$ 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 for 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 $\exists(C_{j+1}, \text{true})$. 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, is 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 $t : G \implies_{\rho} H$ 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 a subset of the set of direct consistency-maintaining rules at layer, since these rules must be plain rules, whereas direct consistency-maintaining rules at layer are allowed to have application conditions, e.g. $\text{main}(\cdot, \cdot)$.

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 $-1 \leq k < \text{nl}(c)$ 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 11 and constraints c_1 and c_2 given in Figure 6. 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 $\text{src}(e) = v$ or $\text{tar}(e) = v$ 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 $\text{src}(e) = v$ or $\text{tar}(e) = v$ 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 $\text{IG}(C_k, C_{k+1})$.

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

Lemma 4.5. *Given a plain rule $\rho = L \xleftarrow{l} K \xrightarrow{r} R$ and a constraint c in UANF. Let $-1 \leq k < \text{nl}(c)$ 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 $k < \text{nl}(c) - 2$.*

1. For each existentially bound graph C_j with $2 \leq j \leq k + 1$ and each overlap $P \in \text{ol}(L, C_j)$ 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 $P \in \text{ol}(R, C_j)$ 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 $P \in \text{ol}(L, C_{k+3})$ 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 $\rho = L \xleftarrow{l} K \xrightarrow{r} R$ be a rule that satisfies the characterisations listed in Lemma 4.5 with $-1 \leq k < \text{nl}(c)$ 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 $\text{k}_{\max}(c, G) = k$ such that t is not direct consistency-maintaining w.r.t. c . Therefore, either the no new violation by deletion, the no new violation by insertion, the no satisfied layer reduction by insertion or the no satisfied layer reduction by deletion formula is not satisfied.

1. If the no new violation by deletion formula is not satisfied, then $k < \text{nl}(c) - 2$. There is an occurrence $p : C_{k+2} \hookrightarrow G$ such that $p \models \text{IC}_0(\text{sub}_{k+2}(c), C')$ and $\text{tr}_t \circ p \not\models \text{IC}_0(\text{sub}_{k_{\max}+2}(c), C')$ with $C' \in \text{IG}(C_{k+2}, C_{k+3})$. So an overlap $P \in \text{ol}(L, C_{k+3})$ 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 $p = \text{tr}_t \circ q$ exists and $p \not\models \text{false}$ if $\text{sub}_{k+2}(c) = \text{false}$ and $p \not\models \text{IC}_0(\text{sub}_{k+2}(c), C_{j+3})$ otherwise. So there is an overlap $P \in \text{ol}(R, C_{k+2})$ 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 $\text{tr}_t \circ q = p$ exists. So there is an overlap $P \in \text{ol}(R, C_j)$ 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 $\text{tr}_t \circ p$ is not total. So there is an overlap $P \in \text{ol}(L, C_j)$ 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 . \square

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 $C \in \text{IG}(C_{k+2}, C_{k+3})$ is inserted. Of course, this second case only occurs if $k < \text{nl}(c) - 2$, 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 $L' \xleftarrow{l'} K \xrightarrow{r'} R$, $C_{k+2} \xleftarrow{\text{id}} C_{k+2} \xrightarrow{\text{id}} C_{k+2}$ and $\rho = L \xleftarrow{l} K \xrightarrow{r} R$ 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.

Definition 4.6 (basic increasing rule). *Given a constraint c in UANF and a direct consistency-maintaining rule $\rho = (\text{ac}, L \xleftarrow{l} K \xrightarrow{r} R)$ at layer $-1 \leq k \leq \text{nl}(c) - 2$, 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.
2. Intermediate inserting: If $k < \text{nl}(c) - 2$, there is an intermediate graph $C' \in \text{IG}(C_{k+2}, C_{k+3})$ such that $p \not\models \exists(a_{k+2}^r : C_{k+2} \hookrightarrow C', \text{true})$, $r \circ l^{-1} \circ p$ is total and $r \circ l^{-1} \circ p \models \exists(a_{k+2}^r : C_{k+2} \hookrightarrow C', \text{true})$. Then, ρ is called a inserting basic increasing rule with C .

Example 4.4. Consider the rule `assignFeature` given in Figure 11 and constraint c_1 given in Figure 6. Then, `assignFeature` is an inserting basic rule with $C_2^2 \in \text{IG}(C_1^1, C_2^1)$

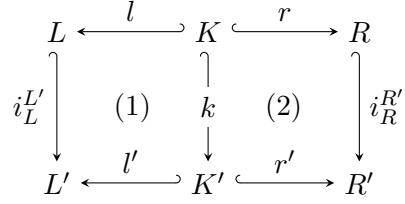


Figure 18: Pushout diagram for Lemma 4.8.

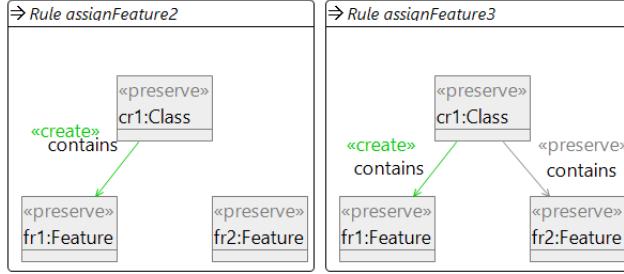


Figure 19: Derived rules of `assignFeature` and c_1 .

w.r.t. c_1 but is not an inserting basic rule with respect to the constraint $\forall(C_2^2, \exists(C_2^1, \text{true}))$ 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 $\rho = (\text{ac}, L \xleftarrow{l} K \xrightarrow{r} R)$. The set of derived rules from ρ at layer $-1 \leq k \leq \text{nl}(c) - 2$, where k is odd, contains rules characterised as follows: Let

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

For $P \in \mathbf{G}$ and $(L', i_L^{L'}, i_P^{L'}) \in \text{ol}(L, P)$: If the diagram shown in Figure 18 is a transformation, i.e. (1) and (2) are pushouts. The rule

$$\rho' = (\text{Shift}(\text{ac}, i_L^{L'}), L' \xleftarrow{l'} K' \xrightarrow{r'} R')$$

is a derived rule of ρ at layer k .

Example 4.5. Consider the rule `assignFeature` given in Figure 11 and constraint c_1 given in Figure 6. The set of derived rules from ρ at layer 1 is given in Figure 19.

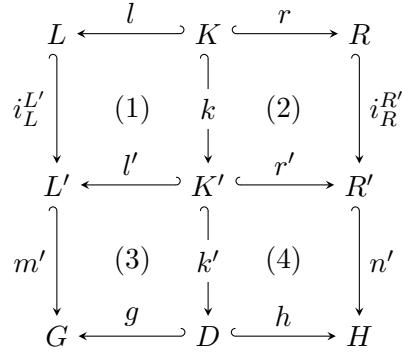


Figure 20: Pushout diagram for the construction of basic increasing rules.

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.

Lemma 4.8. *Given a graph G and rules $\rho = L \xleftarrow{l} K \xrightarrow{r} R$ and $\rho' = L' \xleftarrow{l'} K' \xrightarrow{r'} R'$ 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 18.*

Proof. Consider the transformation composed of the pushouts (3) and (4) given in Figure 20. 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 $m' \circ i_L^{L'}$. \square

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 semantics, and to use the less complex and less restrictive basic application conditions instead of the general ones.

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 $\rho = (\text{ac}, L \xleftarrow{l} K \xrightarrow{r} R)$. 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 $P \in \text{ol}(C_{k+2}, L)$ 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 ρ and m satisfy the dangling edge condition. One sees easily that $GP_{\rho, m} = q(GP_{\rho, i_L^P})$ and $DP_{\rho, m} \subseteq q(DP_{\rho, i_L^P})$. It follows that $DP_{\rho, i_L^P} \subseteq GP_{\rho, i_L^P}$ and therefore, there is a transformation $t : P \Longrightarrow_{\rho, i_L^P} R'$. In particular, there is a derived rule $\rho' = (\text{ac}', L' \xleftarrow{l'} K' \xrightarrow{r'} R)$ of ρ at layer k , where $L' = P$. We set $m' = q$, since $m = m' \circ i_L^{L'} \models \text{ac}$, it follows that $m' = \text{ac}'$, and since ρ' removes and inserts the same elements as ρ , there is the transformation $t' : G \Longrightarrow_{\rho', m'} H$. \square

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 direct consistency-maintaining at a certain layer k it is sufficient to check whether $m \circ i \not\models \exists(C_{k+3}, \text{true})$ if ρ is a deleting rule, and whether $m \circ i \not\models \exists(C', \text{true})$ 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 $\rho = (\text{ac}, L \xleftarrow{l} K \xrightarrow{r} R)$ w.r.t. c at layer $-1 \leq k \leq \text{nl}(c) - 2$, where k is odd. The basic application condition of ρ w.r.t. c at layer $-1 \leq j \leq \text{nl}(c) - 2$ is given by*

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

with

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

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 $\exists(C, \text{true})$. But an application of this rule could also lead to a consistency increasing transformation w.r.t. c if the increasing morphism satisfies $\exists(C, \text{true})$ 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 6, there is a consistency increasing transformation $t : C_2^2 \xrightarrow{\text{assignFeature}, m} C_2^1$ such that $m \not\models \text{basic}_{-1}(\text{assignFeature})$, but there is also a transformation $t : C_2^2 \xrightarrow{\text{assignFeature3}, m'} C_2^1$ with $m' \models \text{basic}_{-1}(\text{assignFeature3})$.

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 $\rho = (\text{ac}, L \xleftarrow{l} K \xrightarrow{r} R)$ w.r.t c at layer $-1 \leq k \leq \text{nl}(c) - 2$, where k is odd.*

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

Proof. Given a graph G with $k_{\max} = k$. We show that each transformation $t : G \xrightarrow{\rho', m} H$ 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 $\exists(C_{k+3}, \text{true})$. 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 $\exists(C', \text{true})$, since $m \models \text{basic}_k(\rho)$. Since ρ' is a inserting rule it holds that $\text{tr}_t \circ m \circ i = n \circ r \circ l^{-1} \circ i \models \exists(C', \text{true})$ 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 . \square

Example 4.6. *Again, consider the rule `assignFeature`, its derived rule `assignFeature3` and c_1 . The basic application condition for these rules at layer -1 w.r.t. c_1 is given in Figure 21.*

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

$$\text{basic_1(assignFeature)} = \neg \exists \left(\begin{array}{c} \boxed{\text{rc1} = \text{cc1:Class}} \\ \downarrow \text{surfaces} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \end{array}, \text{true} \right) \wedge \neg \exists \left(\begin{array}{c} \boxed{\text{rc1} = \text{cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \quad \boxed{\text{cf2:Feature}} \end{array}, \text{true} \right)$$

$$\text{basic_1(assignFeature3)} = \neg \exists \left(\begin{array}{c} \boxed{\text{rc1} = \text{cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{rf1} = \text{cf1:Feature}} \quad \boxed{\text{cf2:Feature}} \end{array}, \text{true} \right) \wedge \neg \exists \left(\begin{array}{c} \boxed{\text{rc1} = \text{cc1:Class}} \\ \downarrow \text{contains} \\ \boxed{\text{cf1:Feature}} \quad \boxed{\text{rf1} = \text{cf3:Feature}} \end{array}, \text{true} \right)$$

Figure 21: Basic 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 that do not satisfy $\exists(C_{k+1}, \text{true})$ are inserted, and any occurrences of C_k that satisfied $\exists(C_{k+1}, \text{true})$ also satisfy $\exists(C_{k+1}, \text{true})$ after the repair. A set of constraints \mathcal{C} is *circular conflict free* if there exists a sequence c_1, \dots, c_n with $c_i \in \mathcal{C}$ such that repairing a constraint c_i does 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 $i > k_{\max} + 2$, but if $i \leq k_{\max} + 2$ it could lead either to the insertion of new violations or a reduction of the largest satisfied layer. In addition, 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. An existentially bound graph C_k causes a conflict for a universally bound graph C_j if there is a transformation $t : G \Rightarrow_{\rho} H$ with $\rho = C_{k-1} \xleftarrow{\text{id}} C_{k-1} \xrightarrow{a_{k-1}} C_k$ such that

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

A universally bound graph C_j causes a conflict for an existentially bound graph C_k if there is a transformation $t : G \Rightarrow_{\rho} H$ with $\rho = C_j \xleftarrow{a_{j-1}^r} C \xrightarrow{\text{id}} C$ for any $C \in \text{IG}(C_{j-1}, C_j)$

such that

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

In addition, we introduce *conflict graphs*, which represent the conflicts within a constraint 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 $C_k = C_{j_1}, \dots, C_{j_n} = C_k$ 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 is possible to choose whether a violation is removed by deletion or insertion. Otherwise, it must be done alternately. That is after a violation is removed by deletion, all violations introduced by that deletion must be removed by insertion, and vice versa. This also makes the definition of a repairing set 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 $0 \leq k < \text{nl}(c)$ there is a node labelled k . If C_k causes a conflict for C_j , there is an edge e with $\text{src}(e) = k'$ and $\text{tar}(e) = j'$ 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 22. Transformation t_1 shows that C_1 has a conflict with C_2 because the rule $\rho = C_1 \xleftarrow{\text{id}} C_1 \xleftarrow{a_1} C_2$ 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 $C_2 \xleftarrow{a_1} C_1 \xleftarrow{\text{id}} C_1$ 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 23.

In general, the statement “ C_j causes a conflict for C_k ” does not imply that “ C_k causes a conflict for C_j ” as shown by constraint c_4 given in Figure 22. The conflict graph of c_4 is also shown in Figure 23. It can be seen that c_4 is a circular conflict-free constraint.

We will now present two characterisations of conflicts. The first one is based on the applicability of certain rules and the second one is based on the notion of basic maintaining rules.

For C_k , which is existentially bound, and C_j , which is universally bound, the first 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 $C_k \setminus C_{k-1}$

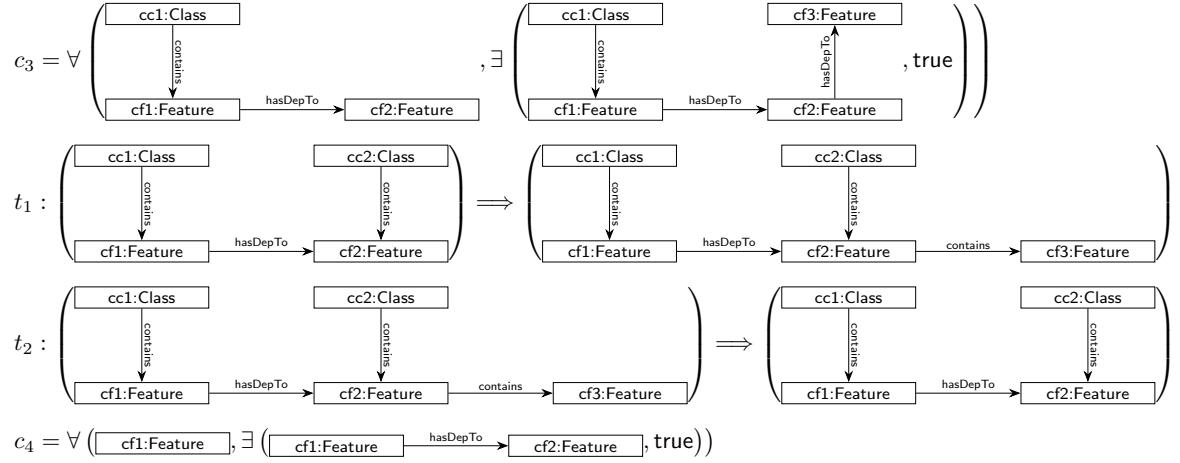


Figure 22: Constraint c_3 and the transformation that show the existence of conflicts between C_1 and C_2 and C_2 and C_1 .

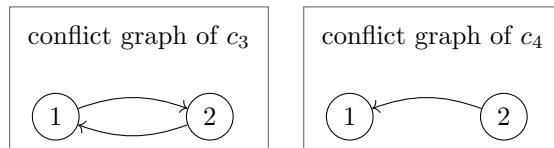


Figure 23: Conflict graphs of c_3 and c_4 .

is applicable. If this is not possible, there is no transformation as described in Definition 5.2. If C_j is universally bound and C_k is existentially bound, the characterisation checks whether for each overlap of C_k and C_j such that the elements of $C_j \setminus C_{j-1}$ and C_k overlap, a rule is applicable which only removes elements of $C_j \setminus C_{j-1}$. 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 bound 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 $P \in \text{ol}(C_k, C_j)$ with*

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

and the rule $\rho = C_k \xleftarrow{\text{id}} C_{k-1} \xrightarrow{\text{id}} C_{k-1}$ is applicable at match $i_{C_k}^P$.

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

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

and a rule $\rho = C_j \xleftarrow{a_{j-1}^r} C \xrightarrow{\text{id}} C$ with $C \in \text{IG}(C_{j-1}, C_j)$ and $i_{C_j}^P(C_k \setminus C) \cap i_{C_k}^P(C_k) \neq \emptyset$ is applicable at match $i_{C_j}^P$.

Proof. Given a condition c in UANF.

1. “ \implies ”: Let C_k be an existentially bound graph that causes a conflict for a universally bound graph C_j . Then, there is a transformation $t : G \implies_{\rho, m} H$ with $\rho = C_{k-1} \xleftarrow{\text{id}} C_{k-1} \xrightarrow{a_{k-1}^r} C_k$ 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 $p(C_j) \cap n(C_k \setminus C_{k-1}) \neq \emptyset$, where n is the co-match of t . The overlap $(P, p, i_{C_k}^P)$ with $P = p(C_j) \cup n(C_k)$ and $i_{C_k}^P = n$ is the one we are looking for. Since ρ^{-1} is applicable at match n , it holds that $DP_{\rho^{-1}, n} \subseteq GP_{\rho^{-1}, n}$. It also holds that $GP_{\rho^{-1}, n} = GP_{\rho^{-1}, i_{C_k}^P}$ and $DP_{\rho^{-1}, i_{C_k}^P} \subseteq DP_{\rho^{-1}, n}$. It follows that $DP_{\rho^{-1}, i_{C_k}^P} \subseteq GP_{\rho^{-1}, n} = GP_{\rho^{-1}, i_{C_k}^P}$ and therefore, the rule ρ^{-1} is applicable at the match $i_{C_k}^P$.

“ \impliedby ”: Let C_k be an existentially and C_j a universally bound graph such that there exists an overlap $P \in \text{ol}(C_k, C_j)$ with $i_{C_k}^P(C_k \setminus C_{k-1}) \cap i_{C_j}^P(C_j \setminus C_{j-1}) \neq \emptyset$ so that the rule $\rho = C_k \xleftarrow{a_{k-1}^r} C_{k-1} \xrightarrow{\text{id}} C_{k-1}$ is applicable at match $i_{C_k}^P$. Then the inverse transformation of $t : P \implies_{\rho, i_{C_k}^P} H$ is the transformation we are looking for and C_k causes a conflict for C_j .

2. “ \implies ”: Let C_j be a universally bound graph that causes a conflict for an existentially bound graph C_k . Then, there is a transformation $t : G \implies_{\rho, m} H$ with $\rho = C_j \xleftarrow{a_{j-1}^r} C \xrightarrow{\text{id}} C$ and $C \in \text{IG}(C_{j-1}, C_j)$ such that $\text{tr}_t \circ p$ is no total for an

occurrence $p : C_k \hookrightarrow G$. The overlap $(P, i_{C_k}^P, i_{C_j}^P)$ with $P = p(C_k) \cup m(C_j)$, $i_{C_k}^P = p$ and $i_{C_j}^P = m$ is the one we are looking for and $i_{C_j}^P(C_j \setminus C) \cap i_{C_k}^P(C_k) \neq \emptyset$ must hold since ρ only deletes elements of $C_j \setminus C$. Since ρ is applicable at match m , it holds that $DP_{\rho,m} \subseteq GP_{\rho,m}$. It also holds that $GP_{\rho,m} = GP_{\rho,i_{C_j}^P}$ and $DP_{\rho,i_{C_j}^P} \subseteq DP_{\rho,m}$.

It follows that $DP_{\rho,i_{C_j}^P} \subseteq GP_{\rho,i_{C_j}^P}$ and therefore, ρ is applicable at match $i_{C_j}^P$.

“ \Leftarrow ”: Let C_j be universally and C_k existentially bound such that there is an overlap $P \in \text{ol}(C_j, C_k)$ with $i_{C_j}^P(C_j \setminus C_{j-1}) \cap i_{C_k}^P(C_k) \neq \emptyset$ such that a rule $\rho = C_j \xleftarrow{\text{id}} C \xrightarrow{\text{id}} C$ with $C \in \text{IG}(C_{j-1}, C_j)$ is applicable at match $i_{C_j}^P$. Then, the transformation of $t : P \Rightarrow_{\rho, i_{C_j}^P} H$ is the transformation we are looking for and C_j causes a conflict for C_k .

□

Note that this characterisation 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 $\rho = C_k \xleftarrow{\text{id}} C_{k-1} \xrightarrow{\text{id}} C_{k-1}$ and $C_j \xleftarrow{\text{id}} C_j \xrightarrow{\text{id}} C_j$ 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 $C_j \xleftarrow{\text{id}} C \xrightarrow{\text{id}} C$ and $C_k \xleftarrow{\text{id}} C_k \xrightarrow{\text{id}} C_k$ are parallel independent for all $C \in \text{IG}(C_{j-1}, C_j)$.

Our second characterisation of conflicts is based on the notion of basic maintaining rules.

Lemma 5.4. *Let a condition c in UANF be given.*

1. *Let C_k be an existentially and C_j be a universally bound graph of c . Then, C_k causes a conflict for C_j if and only if the rule $\rho = C_{k-1} \xleftarrow{\text{id}} C_{k-1} \xleftarrow{\text{id}} C_k$ is not a basic consistency maintaining rule w.r.t. $\forall(a_{j-1} \circ \dots \circ a_0 : C_0 \hookrightarrow C_j, \text{false})$.*
2. *Let C_j be a universally and C_k be an existentially bound graph of c . Then, C_j causes a conflict for C_k if and only if a rule $\rho = C_j \xleftarrow{\text{id}} C \xrightarrow{\text{id}} C$ with $C \in \text{IG}(C_{j-1}, C_j)$ is not a basic consistency maintaining rule w.r.t. $\exists(a_{k-1} \circ \dots \circ a_0 : C_0 \hookrightarrow C_k, \text{true})$.*

Proof. 1. Let C_k be an existentially and C_j a universally bound graph of c .

“ \Rightarrow ”: Assume that C_k causes a conflict for C_j . Therefore, there is a transformation $t : G \Rightarrow_{\rho} H$ with $\rho = C_{k-1} \xleftarrow{\text{id}} C_{k-1} \xleftarrow{\text{id}} C_k$ such that a new occurrence $p : C_j \hookrightarrow H$ has been inserted. Then, t does not satisfy the no satisfied layer reduction by insertion formula and ρ is not a basic maintaining rule w.r.t. $\forall(a_{j-1} \circ \dots \circ a_0 : C_0 \hookrightarrow C_j, \text{false})$.

“ \Leftarrow ”: Assume that $\rho = C_{k-1} \xleftarrow{\text{id}} C_{k-1} \xleftarrow{a_{k-1}} C_k$ is not a basic maintaining rule w.r.t. $\forall(a_{j-1} \circ \dots \circ a_0 : C_0 \hookrightarrow C_j, \text{false})$. Since this constraint only contains universally bound graphs, there must exist a transformation $t : G \Rightarrow_\rho H$ that does not satisfy the no satisfied layer reduction by insertion formula. Therefore, a new occurrence of C_j has been inserted by t and with Definition 5.1 follows that C_k causes a conflict for C_j .

2. Let C_j be a universally and C_k be an existentially bound graph of c and $c' = \exists(a_{k-1} \circ \dots \circ a_0 : C_0 \hookrightarrow C_k, \text{true})$.

“ \Rightarrow ”: Assume that C_j causes a conflict for C_k . Therefore, there is a transformation $t : G \Rightarrow_\rho H$ with $\rho = C_j \xleftarrow{a_{j-1}^r} C \xleftarrow{\text{id}} C$, for a $C \in \text{IG}(C_{j-1}, C_j)$ such that an occurrence of C_k has been destroyed. Then, t does not satisfy the no satisfied layer reduction by insertion formula. Therefore, ρ is not a basic consistency-maintaining rule w.r.t. c' .

“ \Leftarrow ”: Assume that $\rho = C_j \xleftarrow{a_{j-1}^r} C \xleftarrow{\text{id}} C$ is a not a basic increasing rule w.r.t. c' . Therefore, there is transformation $t : G \Rightarrow_\rho H$ that does not satisfy the no satisfied layer reduction by insertion formula and an occurrence of C_k has been removed by t . It follows that C_j causes a conflict for C_k .

□

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 each 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 5.5 (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

$$C_{k-1} = G_0 \xrightarrow{\rho_1, m_1} G_1 \xrightarrow{\rho_2, m_2} \dots \xrightarrow{\rho_n, m_n} G_n$$

with $\rho_i \in \mathcal{R}$ is called a repairing sequence for C_k if $G_n \models_k c$, $\text{tr}_{t_i} \circ \dots \circ \text{tr}_{t_1} \circ \text{id}_{C_{k-1}}$ is total for all $1 \leq i \leq n$. And for each universally bound graph C_j such that C_k

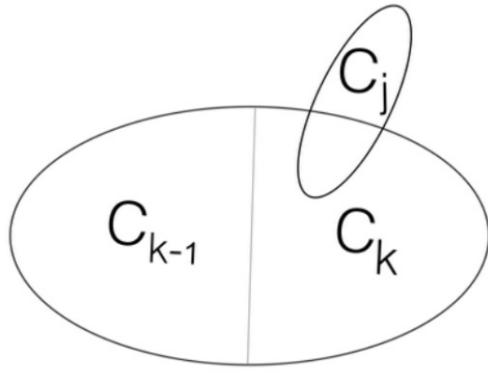


Figure 24: Sketch of an overlap of an existentially bound graph C_k and a universally bound graph C_j that could lead to an insertion of C_j via repairing sequences.

does not cause a conflict for C_j , the concurrent rule of this sequence is a basic consistency maintaining rule w.r.t. $\forall(C_j, \text{false})$.

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

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

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

In both cases of Definition 5.5 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 $C \in \text{IG}(C_{k-1}, C_k)$ 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 24. 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 bound or $G_n = C$ with $C \in \text{IG}(C_{k-1}, C_k)$ 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

$$C_{k-1} \xrightarrow{\rho_1, m_1} \dots \xrightarrow{\rho_n, m_n} C_k$$

with $\rho_i \in \mathcal{R}$ such that $\text{tr}_{t_i} \circ \dots \text{tr}_{t_1} \circ \text{id}_{C_{k-1}}$ is total for all $1 \leq i \leq n$ 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

$$C_k \xrightarrow{\rho_1, m_1} \dots \xrightarrow{\rho_n, m_n} C$$

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

Proof. 1. If C_k is existentially bound, we need to show that the concurrent $\rho = C_{k-1} \xleftarrow{\text{id}} C_{k-1} \xrightarrow{a_k} C_k$ is a basic consistency maintaining rule w.r.t. $\forall(C_j, \text{false})$ 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. $\forall(C_j, \text{false})$. With Lemma 5.4 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 $\rho = C_k \xleftarrow{a_{k-1}^T} C \xrightarrow{\text{id}} C$. Then, ρ is a basic consistency maintaining rule w.r.t. $\forall(C_j, \text{false})$ 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]. Let us now show that a repairing sequence for a graph C_k of a constraint c is applicable at each occurrences of C_k .

Theorem 5.2. *Given a graph G , a constraint c in UANF and and a graph C_k of c with $0 \leq k \leq \text{nl}(c)$.*

1. If C_k is existentially bound. Then, each repairing sequence

$$C_{k-1} = G_0 \xrightarrow{\rho_1, m_1} G_1 \xrightarrow{\rho_2, m_2} \dots \xrightarrow{\rho_n, m_n} G_n$$

for C_k is applicable at every occurrence $p : C_{k-1} \hookrightarrow G$ of C_{k-1} .

2. If C_k is universally bound. Then each repairing sequence

$$C_k = G_0 \xrightarrow{\rho_1, m_1} G_1 \xrightarrow{\rho_2, m_2} \dots \xrightarrow{\rho_n, m_n} G_n$$

for C_k is applicable at every occurrence $p : C_k \hookrightarrow G$ of C_k .

Proof. We show that the concurrent rule of the repairing sequence is applicable. Then, the applicability of the repairing sequence follows immediately.

1. If C_k is existentially bound, let a repairing sequence

$$C_{k-1} = G_0 \xrightarrow{\rho_1, m_1} G_1 \xrightarrow{\rho_2, m_2} \dots \xrightarrow{\rho_n, m_n} G_n$$

for C_k be given. Since $\text{tr}_{t_i} \circ \dots \circ \text{id}_{C_{k-1}}$ is total for all $0 \leq i \leq n$, the concurrent rule is given by $\rho = C_{k-1} \xleftarrow{\text{id}} C_{k-1} \xrightarrow{r} G_n$ and ρ does not delete any elements. Therefore, the dangling edge condition cannot be violated since $GP_{\rho,p} = V_{C_{k-1}}$. It follows that ρ and in particular the repairing sequence is applicable at match p .

2. If C_k is universally bound, let

$$C_k = G_0 \xrightarrow{\tau_1}_{\rho_1, m_1} G_1 \xrightarrow{\tau_2}_{\rho_2, m_2} \dots \xrightarrow{\tau_n}_{\rho_n, m_n} G_n$$

be a repairing sequence for C_j . Since for each node $v \in V_{G_0}$ there is a node $v' \in V_{G_n}$ with $v' = \text{tr}_{t_n}(\dots \text{tr}_{t_1}(v))$, the concurrent rule of the sequence is given by $\rho = C_k \xleftarrow{a_{k-1}^r} C \xrightarrow{r} G_n$ where $C \in \text{IG}(C_{k-1}, C_k)$ and $E_C = E_{C_k}$. It follows that ρ does not delete any nodes and therefore, the dangling edge condition cannot be violated since $GP_{\rho,p} = V_{C_k}$. It follows that ρ and in particular the repairing sequence is applicable at match p .

□

We will now define under which circumstances our repair approach is able to repair a constraint given a rule set \mathcal{R} .

Definition 5.6 (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 for c if there is a repairing sequence for each existentially bound graph of c and, if $\text{nl}(c)$ is odd, i.e. c ends with a condition of the form $\forall(C_{\text{nl}(c)}, \text{false})$, \mathcal{R} contains a repairing sequence for $C_{\text{nl}(c)}$.

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 $\forall(C_1, \text{false})$ such that $E_{C_1} = \emptyset$.

Theorem 5.3. 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

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

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 (Figure 6), c_4 (Figure 22) and the sequences shown in Figure 25. 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 for c_4 . The second sequence is a repairing sequence for the existentially bound graph of c_4 , since the last graph satisfies c_4 . The existentially bound graph causes a conflict for 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 for c_4 .

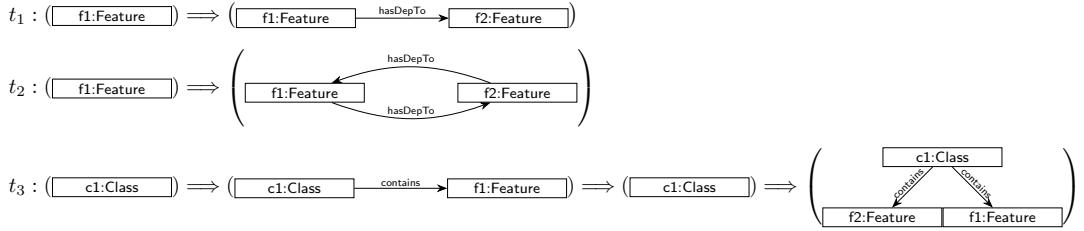


Figure 25: Repairing sequences for c_1 and c_4 .

Algorithm 1: Determine k_{\max} .

Data: A graph G , a constraint c in UANF.
Result: k_{\max} .

```

1 if  $G \models c$  then
2   | return  $\text{nl}(c) - 1$ ;
3 end
4 for  $i \leftarrow -1$  to  $\text{nl}(c) - 1$  by 2 do
5   | if  $G \not\models \text{cut}_i(c)$  then
6     |   | return  $i - 2$ ;
7   end
8 end

```

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 for c_1 .

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 start with an algorithm that computes k_{\max} given a graph G and a constraint c in UANF, as shown in Algorithm 1.

Theorem 5.4. *Given a graph G and a constraint c in UANF, Algorithm 1 returns $k_{\max}(c, G)$.*

Proof. If $G \models c$, then $k_{\max} = \text{nl}(c) - 1$ 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 $-1 \leq k < \text{nl}(c)$. Therefore, 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 k_{\max} . \square

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

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

The condition d is equal to `false` if $k_{\max} + 2 = \text{nl}(c) - 2$ and equal to $\exists(C_{k_{\max}+2}, \text{true})$ otherwise. All these occurrences are contained in the set P (line 3). If P is empty, Lemma 3.16 implies $G \models_{k_{\max}+2} c$, 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 an existential constraint c , i.e. their equivalent constraint in UANF is $\forall(\emptyset, c)$, there may exist occurrences of $C_{k_{\max}+2}$ 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 $\text{cut}_0(\text{sub}_{k_{\max}+2}(c))$. 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 $C_{k_{\max}+2}$ since this graph is universally bound. If this is the case, we use the repairing sequence for $C_{k_{\max}+3}$. This must exist because $C_{k_{\max}+3}$ is existentially bound and \mathcal{R} is a repairing set for c .

If the repairing sequence for $C_{k_{\max}+2}$ was applied, occurrences of existentially bound graphs may have been destroyed. Note that these can only be occurrences of graphs C_i such that $C_{k_{\max}+2}$ 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 $C_{k_{\max}+3}$ has been applied, occurrences of universally bound graphs may have been inserted. Again, these can only be occurrences of graphs C_i such that $C_{k_{\max}+2}$ has a conflict with C_i . This could lead to a reduction of the largest satisfied layer and the algorithm finds all new occurrences of these universally bound graphs (line 11). In both cases, the algorithm only finds potentially increasing occurrences at the respective layer. 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 $p \models \exists(C_{j+1}, \text{true})$ and $\text{tr} \circ p \not\models \exists(C_{j+1}, \text{true})$ 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 extension leads to an increase of the largest satisfied layer, we select one at random. Repairing these occurrences may again result in the insertion of universally bound graphs or the removal of existentially bound graphs. These occurrences are added to M , and this process is repeated until the largest satisfied layer is restored, i.e. $H \models_{k_{\max}} c$ (line 13 – 26). The whole process is repeated until a graph satisfying c is derived.

This shows why c must be free of circular conflicts. For a constraint with circular conflicts, a new occurrence of $C_{k_{\max}+2}$ can be inserted and an occurrence of $C_{k_{\max}+3}$ can be removed during the recovery phase. 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 22. The set of rules used for the transformations t_1 and t_2 in figure 22 forms a repairing set. During a repair process using Algorithm 2, where the starting graph is the first graph of t_1 , Algorithm 2 can enter an infinite loop by alternately applying t_1 and t_2 .

Optimisation of the repair algorithm in terms of the number of elements inserted or deleted can be achieved by using partial repairing sequences where possible. For example, consider the repairing sequence

$$C_k \implies C' \implies \dots \implies C_{k+1}$$

with $C' \in \text{IG}(C_k, C_{k+1})$. For an occurrence p of C_k , which already satisfies the condition $\exists(C_1, \text{true})$, it may be sufficient to apply only the sequence

$$C_1 \implies \dots \implies C_{k+1}$$

at p . Then, we need to check that no occurrences of existentially bound graphs have been destroyed, and that no occurrences of universally bound graphs C_i such that C_k does not cause a conflict for C_i have been inserted. This can be achieved by replacing the application of the repairing sequence by an application of its concurrent rule, and by equipping this concurrent rule with its (basic) consistency-increasing application condition of c at layer k with C_{k+1} . If the application condition is not satisfied, another (partial) repairing sequence must be used. Although this would lead to an optimisation in terms of the number of elements inserted and deleted, it would lead to an increase in runtime due to the construction of application conditions.

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

Theorem 5.5. *Given a graph G , a circular conflict free constraint 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 $C_{k_{\max}+2}$ causes a (transitive) conflict with C_j , since the repairing sequence is not able to destroy or insert occurrences of C_i such that $C_{k_{\max}+2}$ 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 $\text{cut}_{k_{\max}(c,G)}(c)$, 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

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 $H \models c$.

```

1 while  $G \not\models c$  do
2   Determine  $k_{\max}$  using Algorithm 1 ;
3    $P \leftarrow \{p : C_{k_{\max}+2} \hookrightarrow H \mid p \not\models \text{cut}_0(\text{sub}_{k_{\max}+2}(c)) \text{ and } p \text{ is a potentially increasing occurrence at layer } k_{\max} \text{ w.r.t. } c\}$ ;
4   Choose  $p \in P$  uniformly at random ;
5   Choose  $r \in \{0, 1\}$  uniformly at random;
6   if  $r = 0$  and  $\mathcal{R}$  contains a repairing sequence for  $C_{k_{\max}+2}$  then
7     Apply the repairing sequence for  $C_{k_{\max}+2}$  at match  $p$  and let  $H$  be the derived graph ;
8      $M \leftarrow \{q : C_j \hookrightarrow H \mid j < k_{\max}+2 \text{ odd and } \neg\exists q' : C_j \hookrightarrow G(\text{tr} \circ q' = q) \text{ and } q \text{ is a potentially increasing occurrence at layer } j-2 \text{ w.r.t. } c\}$ ;
9   else
10    Apply the repairing sequence for  $C_{k_{\max}+3}$  at match  $p$  and let  $H$  be the derived graph and  $\text{tr}$  the track morphism ;
11     $M \leftarrow \{q : C_j \hookrightarrow H \mid j < k_{\max}+2 \text{ odd and } \exists q' : C_j \hookrightarrow G(q = \text{tr} \circ q' \text{ is total, } q' \models \exists(C_{j+1}, \text{true}) \text{ and } q \not\models \exists(C_{j+1}, \text{true})) \text{ and } q \text{ is a potentially increasing occurrence at layer } j-2 \text{ w.r.t. } c\}$ ;
12  end
13  while  $H \not\models_{k_{\max}(c, G)} c$  do
14    Choose  $p : C_j \hookrightarrow H \in M$  uniformly at random ;
15    Choose  $r \in \{0, 1\}$  uniformly at random ;
16    if  $r = 0$  and  $\mathcal{R}$  contains a repairing sequence for  $C_j$  then
17      Apply the repairing sequence for  $C_j$  at match  $p$  and let  $H'$  be the derived graph ;
18       $M' \leftarrow \{q : C_{j'} \hookrightarrow H' \mid j' \text{ odd and } \neg\exists q' : C_{j'} \hookrightarrow H(\text{tr} \circ q' = q) \text{ and } q \text{ is a potentially increasing occurrence at layer } j'-2 \text{ w.r.t. } c\}$ ;
19    else
20      Apply the repairing sequence for  $C_{j+1}$  at match  $p$  and let  $H'$  be the derived graph and  $\text{tr}$  the track morphism;
21       $M' \leftarrow \{q : C_{j'} \hookrightarrow H' \mid j < k_{\max}+2 \text{ odd and } \exists q' : C_{j'} \hookrightarrow H(q = \text{tr} \circ q' \text{ is total, } q' \models \exists(C_{j'+1}, \text{true}) \text{ and } q \not\models \exists(C_{j'+1}, \text{true})) \text{ and } q \text{ is a potentially increasing occurrence at layer } j'-2 \text{ w.r.t. } c\}$ ;
22    end
23     $M \leftarrow M' \cup \{\text{tr} \circ q \mid q \in M \setminus \{p\} \text{ and } \text{tr} \circ q \text{ is total}\}$  ;
24     $H \leftarrow H'$ ;
25  end
26   $G \leftarrow H$ ;
27 end
28 return  $G$ ;

```

$\exists(C_j, \text{true})$ and $\text{tr} \circ p$ does not satisfy $\exists(C_j, \text{true})$ 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). So we only need to consider the occurrences contained in M . Applying repairing sequences to occurrences $p : C_j \rightarrow H \in M$ 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 $C_{k_{\max}+2}$, otherwise, C_j would have caused a (transitive) conflict for $C_{k_{\max}+2}$ and therefore $C_{k_{\max}+2}$ has a circular conflict. Therefore no occurrences of $C_{k_{\max}+3}$ are destroyed and no occurrences of $C_{k_{\max}+2}$ are inserted. In addition, $C_{k_{\max}+2}$ causes a (transitive) conflict for C_i , and repairing any $p \in M'$ will not lead to the insertion of an occurrence of $C_{k_{\max}+2}$ or removal of an occurrence of $C_{k_{\max}+3}$.

Since c is circular conflict free, there must exist graphs C_i , such that C_i does not cause a conflict with any other graph $C_{i'}$ and $C_{k_{\max}+2}$ 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 $C_{k_{\max}+2}$ 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 $H \models_{k_{\max}(c, G)} c$, 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 $\exists(C_{j+1}, \text{true})$ or have been removed.

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 . \square

Example 5.3. Consider constraint $c = \forall(C_2^2, \exists(C_2^1, \text{true}))$ which is composed of the graphs shown in Figure 6. This constraint is circular conflict-free and a repairing set for c is given in Figure 26. 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 `{remove, insert}`, Algorithm 2 could return one of the graphs G_1, G_2 or G_3 given in Figure 26, 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 5.7 (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 $G \models \mathcal{C}$.

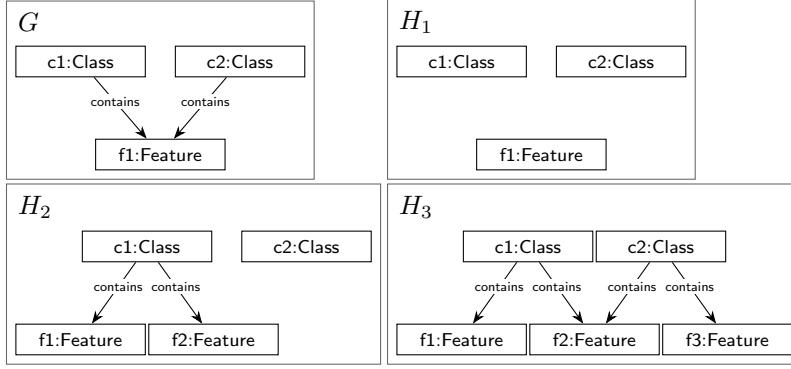


Figure 26: Possible outputs of the repairing process for G and $\forall(C_2^2, \exists(C_2^1, \text{true}))$ using the rule set $\{\text{removeFeature}, \text{createFeature}\}$.

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 5.8 (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 causes a conflict for another constraint c' if one of its graphs causes a conflict for a graph of c' .

Definition 5.9 (conflict between constraints). *Let the constraints c, c' in UANF and a set of rules \mathcal{R} be given. Then c causes a conflict for c' w.r.t. \mathcal{R} if a repairing sequence*

$$C_k = G_0 \xrightarrow{\rho_1, m_1} \dots \xrightarrow{\rho_n, m_n} G_n$$

for a graph, C_k of c exists such that the concurrent rule of that sequence is not a basic consistency maintaining rule w.r.t. $\forall(C_j, \text{false})$ or $\exists(C_j, \text{true})$ 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 causes no conflict for c' .

Lemma 5.10. *Given a set of rules \mathcal{R} and constraints c and c' in UANF such that c causes no conflict for c' w.r.t. \mathcal{R} . Then, the concurrent rule ρ of any graph of c is a c' -preserving rule.*

Proof. Suppose ρ is not a c' -preserving rule. Then there exists a transformation $t : G \xrightarrow{\rho, m} H$ 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

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 $H \models \bigwedge_{c \in \mathcal{C}} c$.

```

1  $(c_1, \dots, c_n) \leftarrow$  topological ordering of  $\mathcal{C}$  w.r.t.  $\mathcal{R}$  ;
2 for  $i \leftarrow 1$  to  $n$  do
3   | Repair  $c_i$  in  $G$  with Algorithm 2, let  $H$  be the returned graph ;
4   |  $G \leftarrow H$  ;
5 end
6 return  $G$ ;
```

that ρ is not a basic maintaining rule w.r.t. $\forall(C_j, \text{false})$ for all universally bound graphs C_j of c' or ρ is not a basic maintaining rule w.r.t. $\exists(C_j, \text{true})$ for all existentially bound graphs C_j of c' , which is a contradiction. \square

The *conflict graph* for a set of constraints and *circular conflicts of a set of constraints* are defined in a similar way to the conflict graph and circular conflicts for one constraint. A set of constraints is called *circular conflict free* if each of its constraints is circular conflict-free and there is no sequence $c = c_0, \dots, c_n = c$ 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 5.11 (conflict graphs, circular conflicts). Given a set of rules \mathcal{R} and a set of constraints \mathcal{C} in UANF. The conflict graph of \mathcal{C} w.r.t. \mathcal{R} is constructed in the following way. For each constraint $c \in \mathcal{C}$ there is a node. If c causes a conflict for c' w.r.t. \mathcal{R} , there is an edge e with $\text{src}(e) = c$ and $\text{tar}(e) = c'$.

A constraint c causes a transitive conflict w.r.t. \mathcal{R} for c' if the conflict graph of \mathcal{C} w.r.t. \mathcal{R} contains a path from c to c' . A constraint c has a circular conflict w.r.t. \mathcal{R} if c has a transitive conflict with itself. A set of constraints \mathcal{C} is called circular conflict free w.r.t. \mathcal{R} if every constraint in \mathcal{C} is circular conflict-free and \mathcal{C} contains no circular conflicts w.r.t. \mathcal{R} .

Example 5.4. Consider the rules `resolve`, `resolve2`, `createFeatures` and constraints c_1 and c_5 given in Figures 27 and 6. The constraint set $\mathcal{C} = \{c_1, c_5\}$ is a multiplicity which expresses that “Each node of type `Class` is connected to exactly two nodes of type `Feature`”. With the rule set $\mathcal{R}_1 = \{\text{resolve}, \text{createFeatures}\}$, there is only one conflict in \mathcal{C} ; c_1 causes a conflict for c_5 , since applying `createFeatures` could lead to inserting the universally bound graph of c_5 . With the rule set $\mathcal{R}_2 = \{\text{resolve2}, \text{createFeatures}\}$ there are two conflicts. Again, c_1 causes a conflict for c_5 and c_5 causes a conflict for c_1 , since applying `resolve` can destroy an occurrence of the existentially bound graph of c_1 .

Therefore, our approach will terminate with the rule set \mathcal{R}_1 but not with \mathcal{R}_2 because \mathcal{C} is not circular conflict free w.r.t. \mathcal{R}_2 .

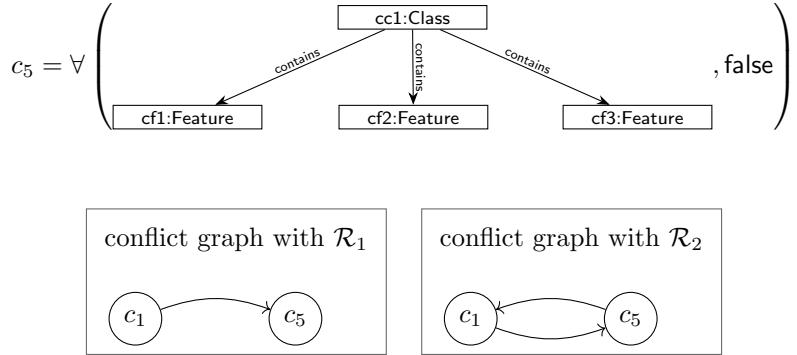


Figure 27: Constraints c_5 and conflicts graphs of the constraint set $\{c_1, c_5\}$ with the rule sets $\mathcal{C}_1 = \{\text{resolve, createFeatures}\}$ and $\mathcal{C}_2 = \{\text{resolve2, createFeatures}\}$.

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

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 5.6. *Given rule set \mathcal{R} , a graph G and a satisfiable set of constraints in UANF \mathcal{C} , which is circular conflict free w.r.t. \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, \dots, c_n) be a topological ordering of \mathcal{C} w.r.t. \mathcal{R} . Then, no constraint c_j with $j \neq 1$ causes a conflict for c_1 , and Lemma 5.10 implies that the concurrent rule of every repairing sequence for every graph of c_i with $2 \leq i \leq n$ is a c_1 -preserving rule. In general, the concurrent rule of each repairing sequence for graphs of c_j is a c_i -preserving rule if $i < j$. 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 2. Since each concurrent rule of each repairing sequence for graphs of c_{m+1} is a c_i -preserving rule for all $1 \leq i \leq m$ and the application of repairing sequence can be replaced by an application of its concurrent rule, 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 satisfies \mathcal{C} . \square

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 transformations based on their invariant checker introduced in [2] for so-called *well-formedness constraints*.

These are constraints of the form $\neg\exists(c_1, \text{true})$ or $\neg\exists(C_1, \neg\exists C_2, \text{true})$. 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 with a rule of the refactoring specification 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 constraints of the form $\forall(C_1, \exists(C'_1, \text{true})) \wedge \dots \wedge \forall(C_n, \exists(C'_n, \text{true}))$. Although the constructed application conditions are not presented as nested conditions, they can be transformed into nested conditions of the form $\forall(C_1, \exists(C_1^1, \text{true}) \vee \dots \vee \exists(C_1^{k_1}, \text{true})) \wedge \dots \wedge \forall(C_n, \exists(C_n^1, \text{true}) \vee \dots \vee \exists(C_n^{k_n}, \text{true}))$.

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 for constructing consistency-sustaining application conditions, a sufficient criterion for consistency-sustaining transformations and a necessary criterion for consistency-improving transformations. Both criteria have been implemented and evaluated.

Constructing optimized constraint-preserving application conditions for model transformation rules: Nassar et al. [14] have introduced a method for constructing consistency-sustaining and consistency-preserving application conditions in the frame-

work 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 $\exists(C, \text{true})$ or is of the form $\exists(C_1, \forall(C_2, \text{false}))$ or $\forall(C_1, \text{false})$. 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 have also introduced a graph repair approach given a set of rules \mathcal{R} . The approach described above 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] have 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 $\forall(C_1, \exists(C_2, \text{true}))$ and $\forall(C_1, \text{false})$. 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 repairing process.

7 Conclusion

In summary, we have introduced a rule-based graph repair approach for circular conflict-free constraints and, in particular, for a circular conflict-free set of constraints, and have shown its correctness and termination. Our approach can also be used for constraints and sets of constraints that are not circular conflict-free. However, there is no guarantee that the algorithm will terminate and thus return a consistent graph.

We have introduced new notions of consistency, called (direct) consistency-maintaining and (direct) consistency-increasing transformations and rules, which are finer-grained than previously known concepts. To do this, we have introduced the notion of *satisfaction up to layer*, which is a notion of partial consistency, and showed what can be deduced from it. For example, if k is even and a graph satisfies a constraint up to layer k , then the graph also satisfies c .

We have used the notions of direct consistency-maintaining and direct consistency-increasing transformations to characterise the circumstances under which a given rule

set \mathcal{R} is able to repair a constraint or set of constraints. To do this, we first introduced the notions of *conflicts within conditions* and *conflicts between conditions*. Intuitively, given a constraint c , a universally bound graph C_k causes a conflict with an existentially bound graph C_j if an occurrence of C_j can be destroyed by destroying an occurrence of C_k . An existentially bound graph C_j causes a conflict for a universally bound graph C_k if an occurrence of C_k can be inserted by inserting elements of $C_j \setminus C_{j-1}$. A set of rules \mathcal{R} is capable of repairing a circular conflict-free constraint if there exists a sequence of transformations that repairs an occurrence of a universally bound graph C_k so that it satisfies $\exists(C_{k+1}, \text{true})$ or destroys its occurrence. Using the notions of direct consistency-maintaining rules and transformations, we ensure that these sequences cannot create or delete occurrences of graphs such that either C_k or C_{k+1} does not cause a conflict for them.

We have compared the notions of (direct) consistency-maintaining and (direct) consistency-increasing transformations and rules with the notions of consistency-guaranteeing, consistency-preserving, (direct) consistency-improving and (direct) consistency sustaining transformations and rules [12, 10]. In general, the notions of (direct) consistency-sustaining and (direct) consistency-improving transformations are not at all related to our newly introduced notions. Only in the special case $\text{nl}(c) = 1$, our notions are identical to those of (direct) consistency-sustaining and (direct) consistency-improving transformations. Our notions are related to those of consistency-preserving and consistency-guaranteeing transformations. If a constraint c is not satisfied, a c -guaranteeing transformation is consistency-increasing w.r.t. c . In general, a consistency-guaranteeing transformation is also consistency-maintaining and a consistency-maintaining transformation is also consistency-preserving.

Furthermore, we have introduced the weaker notions of *(direct) consistency-maintaining rules at layer* and *(direct) consistency-increasing rules at layer*, which allow the construction of less complex application conditions. A rule ρ is direct consistency-maintaining or direct consistency-increasing at layer k if all of its applications at graphs G with $k_{\max} = k$ are (direct) consistency-maintaining or (direct) consistency-increasing. Using these notions, we have introduced four constructions for application conditions and shown that rules equipped with these conditions are direct consistency-maintaining at layer, direct consistency maintaining and direct consistency-increasing at layer. In particular, we have introduced two types of consistency-increasing application conditions at layer. First, application conditions for general rules, and second, application conditions for a special set of rules, called basic rules. Since basic rules, by definition, are not able to introduce new violations or decrease the satisfaction up to layer, it is sufficient to check that at least one violation is removed. Since the left-hand side of a basic increasing rule $L \xleftarrow{l} K \xrightarrow{r} R$ at layer k must contain an occurrence p of C_k such that either $r \circ l^{-1} \circ p \models \exists(C', \text{true})$ and $p \not\models \exists(C, \text{true})$ for an intermediate graph $C' \in \text{IG}(C_k, C_{k+1})$ or $r \circ l^{-1} \circ p$ is not total, it is sufficient to check whether $m \circ p$ does not satisfy $\exists(C', \text{true})$. This leads to less complex application conditions compared to the application conditions of other approaches. Compared to the general ones, these application conditions are less complex and less restrictive.

We have introduced *derived rules* to ensure that the condition that L must contain an occurrence of C_k is not a restriction on the notion of basic rules. The left-hand sides of derived rules contain an occurrence of C_k , and we have shown that these rules are only applicable if the rule from which this rule is derived is applicable at a smaller match such that both transformations construct the same graph.

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. 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. The notion of conflict between constraints is very strict, and we are confident that there exists 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 the set of all graphs of all constraints. Then, we can 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.

We also suggest the following more technical optimisations for the general application conditions. First, not all overlaps in the set $\mathbf{P}_{C'}$ need to be considered when constructing the no violation inserted part of the maintaining application condition since some of them are implied by others. Second, the application condition constructed from the violation removed part of the consistency increasing application condition is far too restrictive, since all occurrences of P' that extend R must satisfy $\exists(C', \text{true})$ after the transformation. We are confident that, with some additional theory, the $\text{exv}(P, C')$ and $\text{remv}(P, C')$ parts of the application condition can be combined to construct less restrictive and less complex application conditions.

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](#). In *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.