Analysis and Evaluation of Conformance Preserving Graph Transformation Rules

Fazle Rabbi, Yngve Lamo, and Lars Michael Kristensen

Western Norway University of Applied Sciences, Bergen, Norway
Fazle.Rabbi@hvl.no, Yngve.Lamo@hvl.no, Lars.Michael.Kristensen@hvl.no

Abstract. Model transformation is a formal approach for modelling the behavior of software systems. Over the past few years, graph based modeling of software systems has gained significant attention as there are numerous techniques available to formally specify constraints and the dynamics of systems. Graph transformation rules are used to model the behavior of software systems which is the core element in model driven software engineering. However, in general, the application of graph transformation rules cannot guarantee the correctness of model transformations. In this paper, we propose to use a graph transformation technique that guarantees the correctness of transformations by checking required and forbidden graph patterns. The proposed technique is based on the application of conformance preserving transformation rules which guarantee that produced output models conform to their underlying metamodel. To determine if a rule is conformance preserving we present a new algorithm for checking conformance preserving rules with respect to a set of graph constraints. We also present a formal proof of the soundness of the algorithm. We apply our technique to homogeneous model transformations where input and output models must conform to the same meta-model. The algorithm relies on locality of a constrained graph to reduce the computational cost.

Keywords: model transformation, graph constraint, metamodelling, formal correctness of model transformations, domain-specific modeling languages

1 Introduction

Model transformation is the process of transforming a model into another model and plays a key role in model driven software development. A transformation rule describes how a target model can be automatically generated from a source model. Often these models need to conform to the syntax and semantics of a metamodel. There are various applications of model transformations such as model migration, model synthesis, code generation, model simulation, model execution, and model repair. Formal development of transformation rules is an important concern since precisely defined rules can be used to verify that the automated transformations are correct [23]. Graph transformation is a formal technique to represent model transformation rules enabling reasoning and

studying properties of transformation systems. Depending on the source and target language, a transformation can be homogeneous or heterogeneous. In homogeneous model transformation, input models and output models belong to the same language. Heterogeneous model transformation transforms models from one language to another. In general, the result of the application of a model transformation rule may lead to inconsistency, i.e., the target model violating constraints defined in its metamodel. Therefore, the application of a model transformation rule requires conformance checking of the target model which is time consuming. To address this problem, it is of interest to develop techniques to reduce the complexity of conformance checking. Since the application of a conformance preserving transformation rule retains the conformance of a model, it eliminates the need for conformance checking of target models. This approach is particularly suited for the development of systems where models produced in every step of a model transformation are supposed to be valid i.e., conforming with respect to a set of constraints.

Current verification approaches for model transformation rules include theorem proving and model checking. The authors in [7] proposed a relational and logical approach to graph grammars that allow the analysis of asynchronous distributed systems with infinite state spaces. They used relational structures to define graph grammars and first-order logic to model graph transformations. They provided a semi-automated process to prove structural properties of reachable graphs using theorem proving. Another theorem proving technique was presented in [18] based on translating graph grammars into Event-B specifications preserving its semantics and then using theorem provers available for Event-B for analysis. Automatic verification of model transformation is gaining popularity and several methods have already been proposed. Baresi and Spoletini [4] proposed a methodology to analyze graph transformation systems by means of Alloy. Given an initial graph of a system, the method can be used to check the configurations that can be obtained by applying a sequence of transformation rules. In [24], Wang et al. investigated the use of the Alloy analyzer for analyzing model transformation systems. A bounded verification approach was used to check if a model transformation system is correct with respect to conformance by translating a metamodel specification into a relational logic specification in Alloy. The authors in [22] presented a formal semantics of the ATL model transformation language using rewriting logic and Maude. Through the formalization it was possible to simulate and verify model transformations. Although model checking is an elegant analysis method, it requires building the complete state space. This can easily lead to the state explosion problem thereby limiting its practical applicability. Hackel and Wagner [14] presented an approach that ensures the conformance of graph transformations by automatically adding application conditions to rules. Application conditions are derived by analyzing the constraints individually which can produce an unnecessary large number of application conditions.

In our approach, we use characteristics of model transformation rules and present an algorithm to check if a transformation rule is conformance preserv-

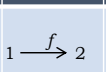
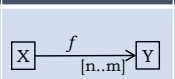
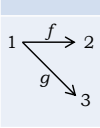
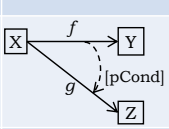
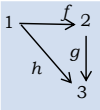
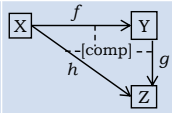
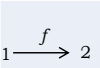
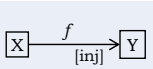
ing with respect to a given set of constraints. We focus on homogeneous model transformation. We do not automatically modify a rule, but provide an algorithm for checking the conformance preserving property of a transformation rule that can be used to provide feedback to the modeler. The approach is illustrated by an example from the healthcare domain.

This paper is an extended version of a previously published article, Rabbi, F., Kristensen, L.M, and Lamo, Y.: Static Analysis of Conformance Preserving Model Transformation Rules, in the proceedings of the 6th International Conference on Model-Driven Engineering and Software Development. This extended version of the paper includes a report on the proof of concept implementation of the proposed algorithm and also includes an evaluation of the proposed algorithm. The rest of the paper is organized as follows. Section 2 provides background on the theoretical foundation of our approach. Section 3 presents the concept of conformance preserving rules. Section 4 presents our algorithm for checking conformance preserving rules. Section 5 presents the evaluation of the proposed technique. Section 6 concludes the paper with a discussion of related and future work. We assume that the reader is familiar with graph transformation systems [10].

2 Modelling in DPF

We use Diagrammatic Logic [8] and the Diagram Predicate Framework (DPF) [19] for the formal development of metamodel specifications. In DPF, a model is represented by a diagrammatic specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ consisting of an underlying graph S together with a set of *atomic constraints* $C^{\mathfrak{S}}$ specified by a *predicate signature* Σ. A predicate signature consists of a collection of predicates, each having a name, an arity (shape graph, $\alpha^{\Sigma}(p)$), visualization and semantic interpretation (see Table 1). The underlying graph and arity of predicates specify type graphs with a data algebra as in [10]. A predicate is used to specify a constraint in a model by means of graph homomorphisms. DPF provides a general mechanism of diagrammatic modeling as it supports various kinds of graph structures. DPF provides a formalization of multi level meta-modelling by defining the conformance relation between models at adjacent levels of a meta-modelling hierarchy. DPF has a potentially unbounded number of metalevels.

There are two kinds of conformance: *typed by* and *satisfaction of constraints*. Figure 1 (top) shows a DPF metamodel specification $\mathfrak{S}$ of a Multiple Sclerosis Clinic. Multiple sclerosis (MS) is a progressive disabling disease of the brain and the spinal cord (central nervous system). In this disease, the immune system attacks the protective sheath (myelin) that covers nerve fibers and causes communication problems between the brain and the rest of the body. Eventually, the disease can cause the nerves themselves to deteriorate or become permanently damaged. The signs and symptoms of this disease vary. Treatments can help reduce the MS symptoms [3]. While some people with severe MS may loose mobility, others experience long periods of remission. It is therefore essential to monitor the progression of the symptoms of MS patients. In the DPF

p	Arity $\alpha^{\Sigma}(p)$	Visualization	Semantic interpretation
<mult(n,m)>			*f* must have at least n and at most m instances for each instance of X
<pre-Condition>			For each instance of *f* there exists an instance of *g* with the same source node
<composite>			For each composition of instances *f;g*, there exists an instance of *h such that h = f;g*
<injective>			Instances of *f* never maps distinct elements of its domain to the same element of its codomain

metamodel specification, we model an MS-application that aims to serve the following purposes:

- Patients can be registered to the MS clinic and can be assigned to medical doctors;
- Medical doctors can have appointment slots;
- Patients can be allocated to appointment slots;
- Patients with appointments to the MS clinic may participate in a survey where they can enter information about their symptoms;
- The doctor assigned to a patient can give an order to perform MRI for patients.

The metamodel specification is constrained by a set of predicates from the signature Σ. Constraints are added into the specifications by graph homomorphisms from the arity (shape graph) of the predicates to the model elements. Below is a list of constraints specified in $\mathfrak{S}$:

- **C1.** *A patient must have exactly one birthdate (specified by <mult(1,1)>)*
- **C2.** *An appointment time-slot allocated to a patient must belong to that patient's assigned doctor (specified by <composite>)*
- **C3.** *An order for MRI can only be given to a patient by that patient's assigned doctor (specified by <composite>)*
- **C4.** *An appointment time-slot cannot be allocated to more than one patient (specified by <injective>)*
- **C5.** *Only registered patients who have appointments are allowed to participate in survey (specified by <pre-condition>)*

To be a valid instance of a DPF metamodel specification, requires that the instance is typed by the shape graph of the metamodel specification and satisfies all the constraints specified in the metamodel. Formally, this means that there is a graph homomorphism ($\iota_I : I \to S$) from the graph I to the graph of $\mathfrak{S}$, where S is the underlying graph of $\mathfrak{S}$. We use a compact notation (I, ι_I) for representing a DPF instance.

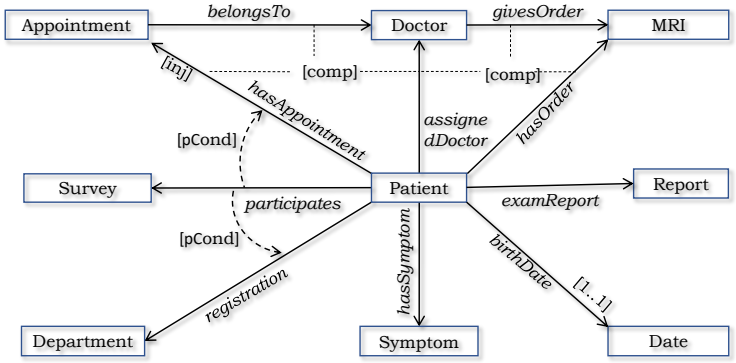


Fig. 1. Metamodel specification $\Im$ of an Orthopedic department

2.1 Coupled graph constraints

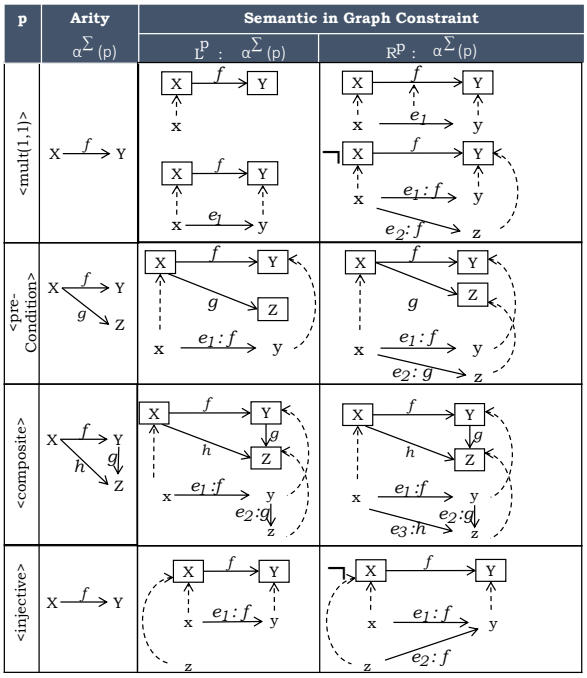
The semantics of a DPF predicate can be specified in various ways. In this paper, we use graph constraints to specify the semantics of the predicates. Typically a graph constraint $N \overset{n}{\leftarrow} L \overset{u}{\rightarrow} R$ consist of three graphs: left L, right R and an application condition N (positive or negative application condition) , and two injective graph homomorphisms n and u where the graphs are typed by the underlying graph of the model [10]. We propose to use graph constraints which conforms to two syntactic formats $\forall L^p \rightarrow \exists R^p$ and $\forall L^p \rightarrow \neg\exists R^p$ where the graphs are typed by the shape graph of the predicates. Therefore we use graph constraints of the following forms where superscript p indicates that the constraint is giving the semantics of a DPF predicate, p. The graph constraints are called coupled graph constraints as they link to predicates.

- $\forall(L^p : \alpha^\Sigma(p)) \rightarrow \exists(R^p : \alpha^\Sigma(p))$, read as "for all matches of the condition pattern L^p (typed by $\alpha^\Sigma(p)$) in a model, there exists a match of the required pattern R^p (typed by $\alpha^\Sigma(p)$) in the model"
- $\forall(L^p : \alpha^\Sigma(p)) \rightarrow \neg\exists(R^p : \alpha^\Sigma(p))$, read as "for all matches of the condition pattern L^p (typed by $\alpha^\Sigma(p)$) in a model, there does not exist a match of the forbidden pattern R^p (typed by $\alpha^\Sigma(p)$) in the model"

Here L^p and R^p are typed graphs over the arity of the predicate p and there exists an inclusion graph morphism $m_c : L^p \hookrightarrow R^p$. A coupled graph constraint gc may have a post-condition $PC(gc)$ imposed on R^p. Table 2 shows the semantics of predicates from signature Σ in terms of graph constraints. The semantic of the <*mult(1,1)*> predicate is given by two graph constraints where the patterns are typed by α^Σ(<*mult(1,1)*>), i.e., the arity of the <*mult(1,1)*> predicate.

Let $gc \in GC(p)$ be a graph constraint linked to a predicate p. A match (δ, m_L) of the condition pattern $(L^p : \alpha^\Sigma(p))$ for the graph constraint gc in a model (I, ι_I) is given by an atomic constraint $\delta : \alpha^\Sigma(p) \rightarrow S$ and an injective morphism m_L such

Table 2. A set of graph constraints giving semantics to the predicates in Σ



that constraint δ and the injective graph homomorphism m_L together with the typing morphisms $\iota_c : (L^p \cup R^p) \to \alpha^{\Sigma}(p)$ and $\iota_I : I \to S$ constitute a commuting square: $\iota_c;\delta = m_L;\iota_I$ as shown in Figure 2(a). If gc has a required pattern ($R^p : \alpha^{\Sigma}(p)$), then for any match (δ, m_L) of the condition pattern in (I, ι_I), a match (δ, m_R) of the required pattern must exist, which is given by the commuting diagram in Figure 2(b). If gc has a forbidden pattern ($R^p : \alpha^{\Sigma}(p)$), then for any match (δ, m_L) of the condition pattern in (I, ι_I), a match (δ, m_R) of the forbidden pattern must not exist such that it constitutes a commuting diagram as shown in Figure 2(c). A valid model is typed by its metamodel specification and conforms to the constraints specified in its metamodel specification. Formally, it states that a valid model (I, ι_I) satisfies all the constraints defined in $\mathfrak{S}$, which is written as $I \models \mathfrak{S}$.

3 Conformance preserving rules

DPF provides functionality to specify graph-based model transformations [20]. We use the standard double-pushout (DPO) [10] approach for defining transformation rules. A model transformation rule ($r : N \xleftarrow{n} L \xleftarrow{m_l} K \xrightarrow{m_r} R$) has a matching pattern (L), a gluing graph (K), a replacement pattern (R) and an

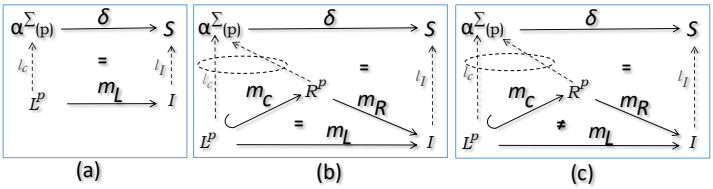
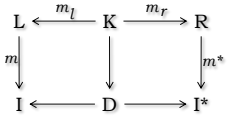


Fig. 2. (a) match of a condition pattern ; (b) match of a required pattern; (c) satisfaction of a forbidden pattern

optional negative application condition, $NAC(n : L \rightarrow N)$ where L, K, R, N are typed by $\Im$ and m_l, m_r, n are injective graph morphisms. We use a transformation approach where transformation rules have a set of negative application conditions as proposed by Lambers et al., in [16].

Given a model (I, ι_I), a model transformation $I \overset{r,m}{\Longrightarrow} I^*$ via a transformation rule $r : L \leftarrow K \rightarrow R$ with a set of negative application conditions NAC_r and a match $m : L \rightarrow I$ consists of the double pushout as shown in the diagram to the right. Here, the injective morphism m satisfies each NAC in NAC_r, written $m \models NAC_r$. When a rule is applied, some elements from the source model are deleted and some elements are added to the target model. The rest of the source model remain unchanged in the target model. A rule is applied as long as it satisfies its negative application conditions. Negative application conditions are typically used in graph transformation to prohibit an infinite number of rule applications. Figure 3 shows a model transformation rule for allocating resources (i.e., appointment) to patients in a model of the metamodel specification from Figure 1. The transformation rule r_1 encodes the following instructions:



– Allocate an appointment *appt* to patient pt_1 if *appt* belongs to the doctor whom pt_1 is assigned to

The typing information of a modelling element in r_1 appears after a colon (:). The green color (thick arrow) is used to represent elements that the rule is going to produce. The rule r_1 has one negative application condition to prohibit an infinite number of rule applications.

One problem with this version of the transformation rule is that it does not guarantee the conformance of constraint **C4** (an appointment time-slot cannot be allocated to more than one patient). The application of the rule may allocate an appointment time-slot to more than one patients. Figure 4 illustrates how we check the satisfaction of the atomic constraint (<*injective*>, δ_1) over model (I^*, ι_{I^*}) by its graph constraint. Even if the rule is applied on a valid model, it does not guarantee that the result will be a valid model conforming to the metamodel specification. The portion of the model that is not conforming to the constraints are highlighted in red (thick arrow) in the figure.

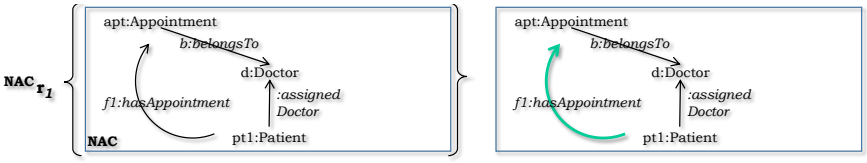


Fig. 3. Transformation rule r_1 for individual resource allocation of patients

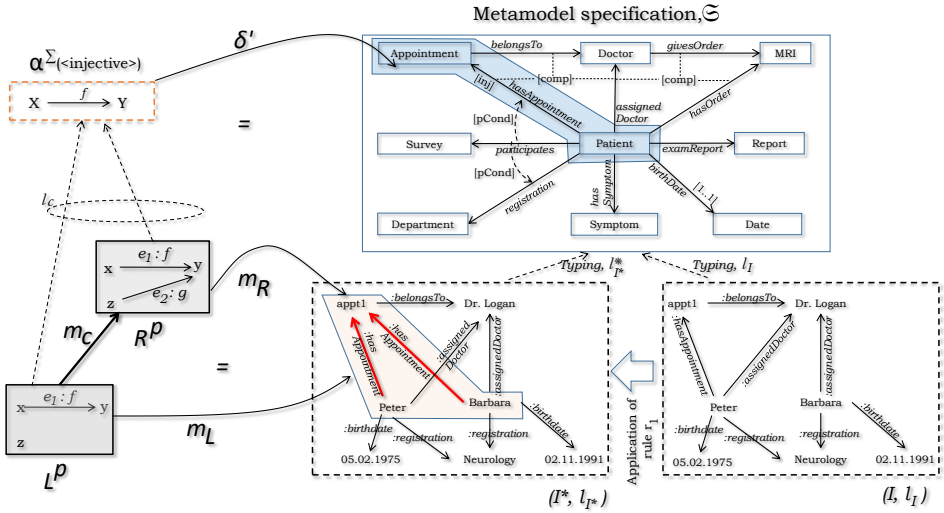


Fig. 4. Application of rule r_1 over a valid model (I, ι_I) and the checking for the satisfaction of a graph constraint

The rule r_1 can be enhanced so that while matching with a model it makes sure that the result will be a valid model. Since the appointment allocation in a valid model of $\mathfrak{S}$ can possibly violate atomic constraints **C2** and **C4**, we enhance rule r_1 with an additional negative application condition to make sure that when applied on a valid model of $\mathfrak{S}$, the output does not violate any of the above mentioned constraints. Figure 5 shows rule r_2 which is conformance preserving and therefore the application of rule r_2 will not require any further conformance checking.

A formal definition of conformance preserving transformation rule is given below:

Definition 1 (Conformance preserving rule). *Given a metamodel specification* $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$. *A transformation rule* r *is conformance preserving w.r.t a set of atomic constraints from* $C^{\mathfrak{S}}$ *if the application of* r *on any valid model* $(I, \iota_I) \models \mathfrak{S}$ *always results in a valid model of* $\mathfrak{S}$.

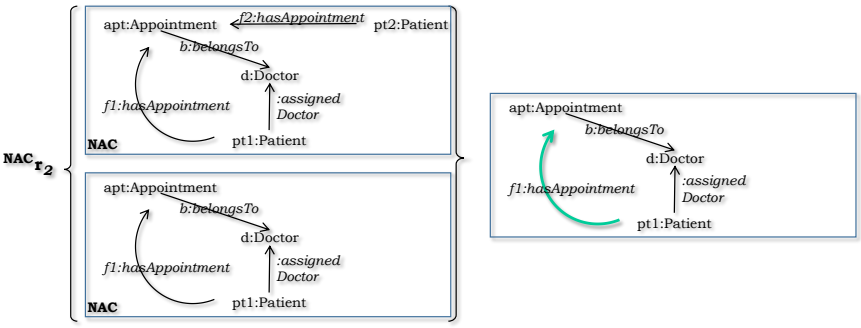


Fig. 5. Conformance preserving transformation rule (r_2) for individual resource allocation of patients

4 Analysis for checking conformance preserving rules

In this section, we present an algorithm to automatically check if a transformation rule is conformance preserving w.r.t a set of constraints specified in a metamodel. To develop an efficient method for determining if a rule is conformance preserving or not, we need to analyze the possibility of the rule to make changes that may violate a given constraint. If a rule makes changes to only the unconstrained portion of a graph, then we can claim that the rule will preserve conformance by its application. If a rule makes changes to the constrained portion of a graph, it is possible that the rule will preserve conformance by its application. We present an algorithm with the aid of a set of patterns to make sure that consistency preserving rules exhibit certain desirable structures.

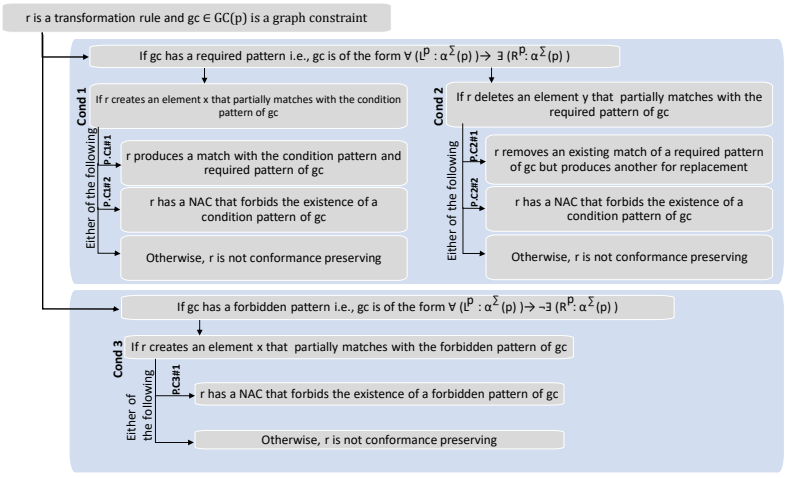


Fig. 6. Informal description of the algorithm illustrating the intuition

4.1 A sufficient condition for conformance

Here we present three conditions to determine if a transformation rule $r : L \leftarrow K \rightarrow R$ can make changes to the constrained portion of a graph i.e., if r can possibly affect an atomic constraint (p, δ):

- **Cond 1:** r creates an element x of type X where X is constrained by a predicate p and X is mapped by the condition pattern of a graph constraint $gc \in GC(p)$ via the typing morphism of L^p and the atomic constraint (p, δ), i.e., $X \in \iota_c; \delta(L^p)$;
- **Cond 2:** r deletes an element y of type Y where Y is constrained by a predicate p and Y is mapped by the elements from (*required pattern*, $R^p \setminus$ *condition pattern*, L^p) via the typing of $(R^p \cup L^p)$ and the atomic constraint (p, δ), i.e., $Y \in \iota_c; \delta(R^p \setminus L^p)$;
- **Cond 3:** r creates an element x of type X where X is constrained by a predicate p and X is mapped by the elements from (*forbidden pattern*, $R^p \setminus$ *condition pattern*, L^p) via the typing of $(R^p \cup L^p)$ and the atomic constraint (p, δ), i.e., $X \in \iota_c; \delta(R^p \setminus L^p)$;

Intuitively, **Cond 1**, **2**, and **3** checks if a rule can create a new match with the condition pattern, delete an existing match of a required pattern, or create a new match with the forbidden pattern of a graph constraint, respectively.

Lemma 1. *Given a metamodel specification $\mathfrak{S}$ with a set of constraints $C^{\mathfrak{S}}$. A transformation rule r is conformance preserving if it does not satisfy any of* ***Cond 1-3****.*

Proof. Let (I, ι) be a valid instance of $\mathfrak{S}$ and the application of r on (I, ι) produces an instance (I^*, ι^*). There are three ways (I^*, ι^*) may violate a constraint from $C^{\mathfrak{S}}$: (i) r produces a new match with the condition pattern L^p of a graph constraint where the corresponding required pattern is missing; (ii) r deletes an existing match of a required pattern; (iii) r produces a new match with the forbidden pattern. However, it can be seen that if r does not satisfy any of **Cond 1-3**, then it does not affect any constraint from $C^{\mathfrak{S}}$ because of the following reasons:

- r does not satisfy **Cond 1**; therefore, it does not produce any new match with the condition pattern L^p of a graph constraint.
- r does not satisfy **Cond 2**; therefore, it does not delete any existing match of a required pattern.
- r does not satisfy **Cond 3**; therefore, it does not produce any new match with the forbidden patterns.

4.2 Desired patterns for conformance

It is possible for a rule r to be conformance preserving even if it satisfies some conditions from **Cond 1-3** and complies with desired patterns described below. Figure 6 illustrates a diagram representing the intuition of the proposed method where **P.C1#1, P.C1#2,...** indicates a pattern number.

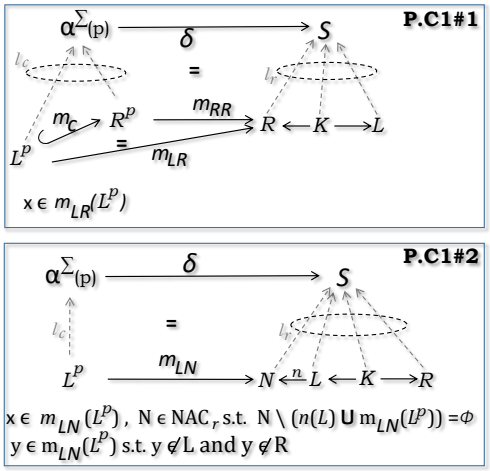


Fig. 7. Patterns for a conformance preserving rule r that satisfies **Cond 1**

*Patterns for **Cond 1**:* In our approach, if a rule satisfies **Cond 1** for a graph constraint gc, it has to comply with the patterns specified in Figure 7. Patterns specified in the figure makes sure that if the creation of an element produces a new match with the condition pattern of a graph constraint, the required pattern must exist (**P.C1#1**); otherwise a new match with the condition pattern is not produced by the application of rule r (**P.C1#2**). Note that in the graph patterns, solid arrows are representing injective graph homomorphisms. In pattern (**P.C1#2**), we check for the non existence of condition pattern by the following condition: $N \in NAC_r$ *s.t.* $N \setminus (n(L) \cup m_{LN}(L^p)) = \emptyset, y \in m_{LN}(L^p)$ *s.t.* $y \notin L$ *and* $y \notin R$.

Figure 8 shows an example of a conformance preserving rule, r. The rule deletes two edges $1 : A \rightarrow 2 : B$ and $3 : C \rightarrow 2 : B$ and creates three edges $1 : A \rightarrow 4 : D$, $4 : D \rightarrow 2 : B$, and $3 : C \rightarrow 2 : B$. The application of rule r over a valid instance is shown in the figure. The output of the transformation is a DPF model instance that produces a new match with the condition pattern, L^p. Since the transformation also produces a corresponding match with the required pattern, R^p, the output model instance is a valid DPF model. To help the reader understanding about the portion of the model being affected by the graph constraint, we show the graph constraint with the typing information using the composition $\iota;\delta$ in the left bottom part of Figure 8.

Given the same model and graph constraint as in Figure 8, an example of a transformation rule r' that is not conformance preserving is shown in Figure 9. The application of rule r' produces two matches with the condition pattern L^p but it produces one corresponding match with the required pattern R^p. Hence it is not satisfying *P.C*1#1 and therefore r' is not conformance preserving.

Figure 10 shows an example of a conformance preserving transformation rule r_2 that complies with (*P.C*1#2). Even though rule r_2 creates and edge that

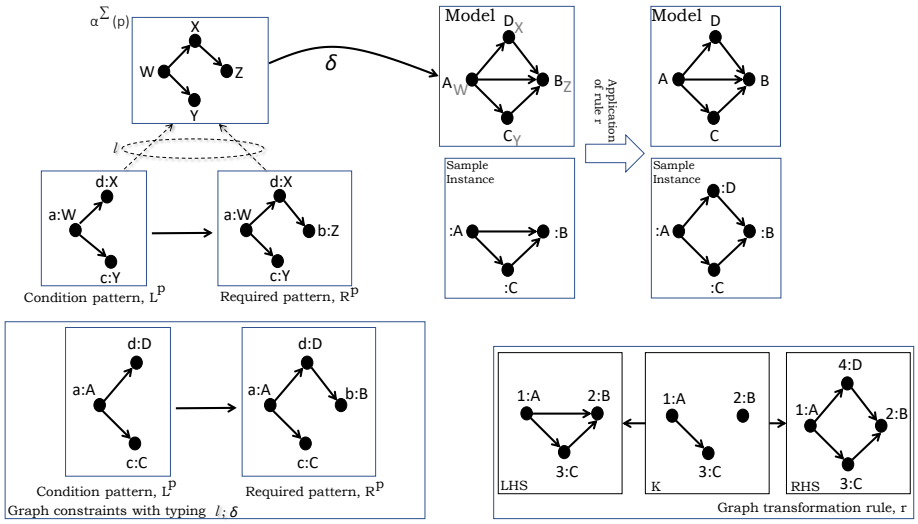


Fig. 8. Example of a conformance preserving transformation rule that complies with *P.C*1#1

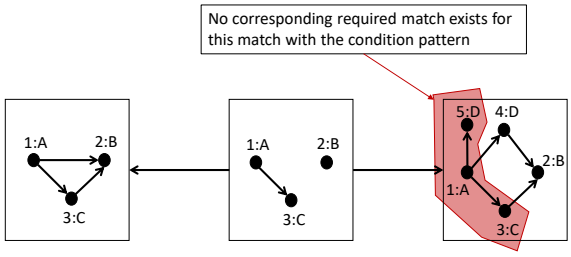


Fig. 9. A transformation rule that does not comply with (*P.C*1#1)

partially matches with the condition pattern L^p i.e., $a:A \rightarrow d:D$ matches with $1:A \rightarrow 4:D$, the rule r_2 has a NAC to ensure that the application of rule r_2 does not produce a complete match with L^p. Therefore we can see that rule r_2 is conformance preserving as it complies with (*P.C*1#2).

Figure 11 shows an example of a transformation rule $r_2^{'}$ that does not comply with (*P.C*1#2). In this rule $X \in NAC_{r_2^{'}}$ but $X \setminus (n(LHS) \cup m_{LN}(L^p)) \neq \emptyset$ as specified in (*P.C*1#2) (see Figure 7). Because of the additional elements $(7:G, 1:A \rightarrow 7:G)$ in X, the rule does not comply with (*P.C*1#2).

Patterns for ***Cond 2****:* The patterns presented in Figure 12 makes sure two of the following:

– (**P.C2#1**): If the deletion of an element removes an existing match of a required pattern of a graph constraint, then another match of a required pattern is produced;

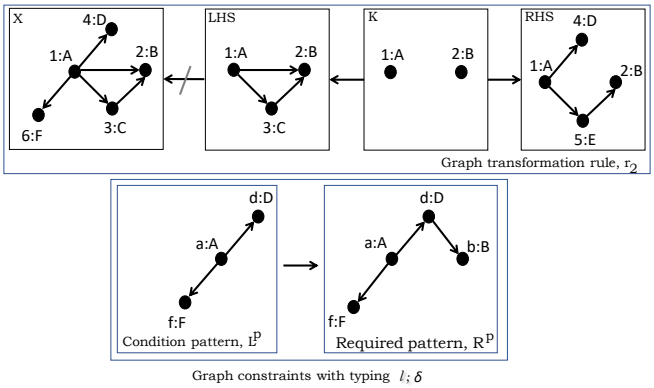


Fig. 10. A conformance preserving rule that complies with (P.C1#2)

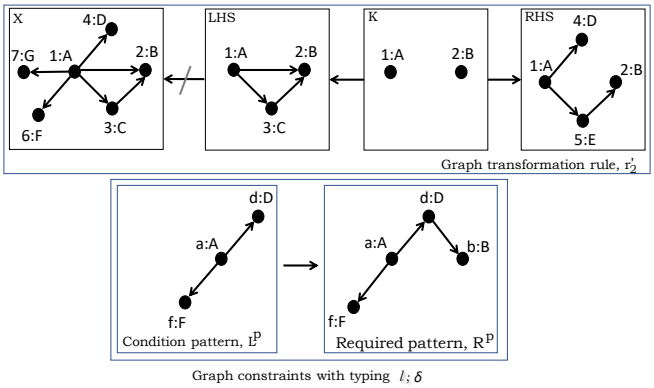


Fig. 11. A transformation rule that does not comply with (*P.C*1#2)

– (**P.C2#2**): Deletion of an element that partially matches with a required pattern, does not remove an existing match of a required pattern.

For checking the non existence of a required pattern in (**P.C2#2**), it is sufficient to check for the non existence of the corresponding condition pattern. If the condition pattern does not exist, we can safely remove portion of a model that partially matches with the requird pattern. Note that in pattern (**P.C2#2**), we check for the non existence of condition pattern by the following condition: $N \in NAC_r$ *s.t.* $N \setminus (n(L) \cup m_{LN}(L^p)) = \emptyset, z \in m_{LN}(L^p)$ *s.t.* $z \notin L$ *and* $z \notin R$. In the rest of the section we provide some examples of transformation rules that comply with the patterns described above.

Figure 13 shows an example of a conformance preserving rule that complies with (**P.C2#1**). The rule removes an existing match with a required pattern by deleting elements $2 : B$ and $3 : C \rightarrow 2 : B$ but produces another match of the required pattern by creating elements $4 : B$ and $3 : C \rightarrow 4 : B$.

Figure 14 shows an example of a conformance preserving transformation rule that complies with (**P.C2#2**). The rule deletes an element $1 : A \rightarrow 6 : F$ that

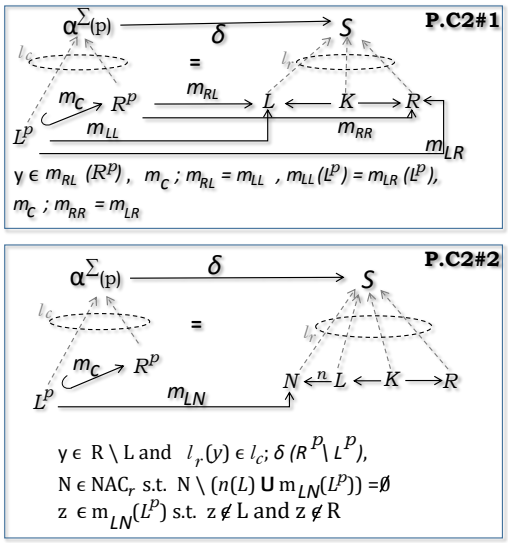


Fig. 12. Patterns for a conformance preserving rule that satisfies **Cond 2**

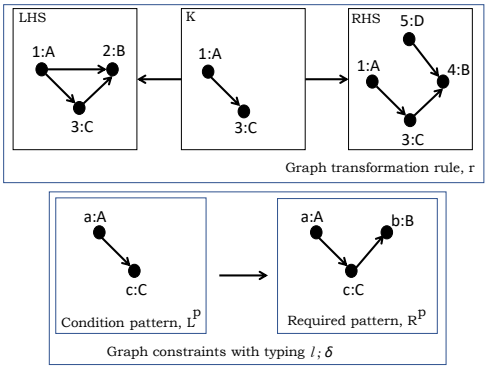


Fig. 13. Example of a conformance preserving rule that complies with (**P.C2#1**)

partially matches with the required pattern, but the rule has a NAC that ensures the non existence of the condition pattern of the graph constraint shown at the bottom of Figure 14.

*Patterns for **Cond 3**:* The pattern **P.C3#1** presented in Figure 15 makes sure that the creation of an element does not produce a match with the forbidden pattern of a graph constraint. We check for the non existence of the forbidden pattern by the following condition: $N \in NAC_r\, s.t.\, N \setminus (n(L) \cup m_{RN}(R^p)) = \emptyset, y \in m_{RN}(R^p)\, s.t.\, y \notin R$.

Figure 16 shows an example of a conformance preserving transformation rule that creates an element $1 : A \rightarrow 4 : D$ which partially matches with the

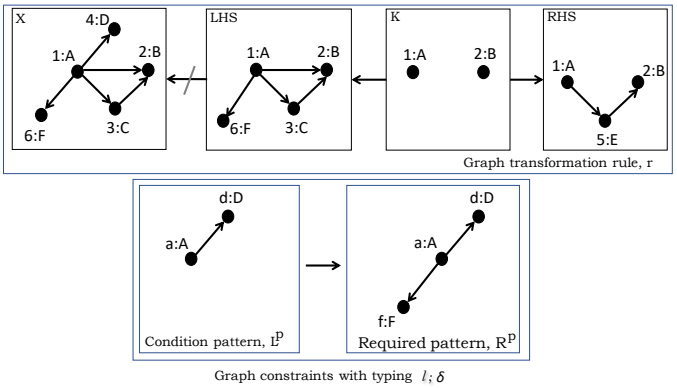


Fig. 14. Example of a conformance preserving rule that complies with (**P.C2#2**)

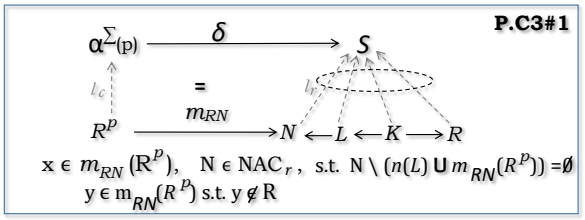


Fig. 15. Pattern for a conformance preserving rule *r* that satisfies **Cond 3**

forbidden pattern of a graph constraint. However, the rule makes sure that a complete match with the forbidden pattern does not exist by means of a negative application condition. Therefore, it complies with **P.C3#1**.

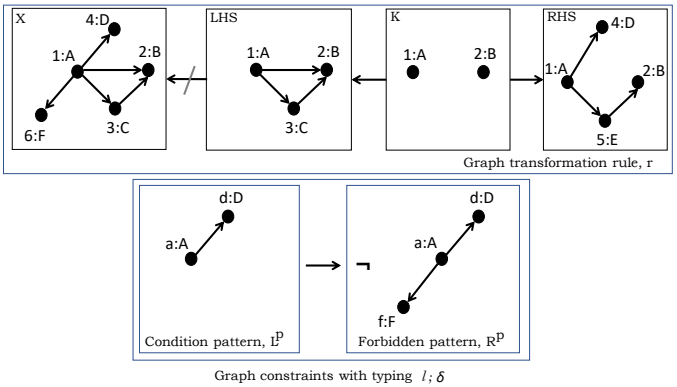


Fig. 16. Example of a conformance preserving transformation rule that complies with **P.C3#1**

4.3 Algorithm for checking conformance preserving rule

Algorithm 1 provides a method for checking the conformance preserving property of a rule w.r.t a set of graph constraints.

Algorithm 1 Check for conformance preserving rule

```
Require: a coupled transformation rule r, a set of graph constraints GC
  C := R \ L of r      //set of elements created by r
  D := L \ R of r      //set of elements deleted by r
  for each x in C do
     for each gc ∈ GC do
        if gc has a required pattern R^p
           and r satisfies Cond 1 for x and gc then
           if r does not comply with P.C1#1 or P.C1#2 for x then
              return ''may not be conformance preserving''
        if gc has a forbidden pattern R^p
           and r satisfies Cond 3 for x and gc then
           if r does not comply with P.C3#1 for x then
              return ''may not be conformance preserving''
  for each y in D do
     for each gc ∈ GC do
        if gc has a required pattern R^p
           and r satisfies Cond 2 for y and gc then
           if r does not comply with P.C2#1 or P.C2#2
           for y then
              return ''may not be conformance preserving''
  return ''conformance preserving''
```

Theorem 1 (Soundness of Algorithm. 1) *Let $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ be a metamodel specification and r (typed by S) a transformation rule which is determined to be conformance preserving w.r.t $C^{\mathfrak{S}}$ by Algorithm 1. If r is applied on a valid model $(I, \iota_I) \models \mathfrak{S}$ then the result (I^*, ι_{I^*}) will be a valid model of $\mathfrak{S}$.*

Proof. Let GC be a set of constraints giving semantics to the set of constraints $C^{\mathfrak{S}}$. To prove the theorem by contradiction, it is sufficient to show that there exists a $gc \in GC$ such that (I^*, ι_{I^*}) does not satisfy gc. There are three ways in which it is possible for (I^*, ι_{I^*}) to violate the graph constraint:

i gc is of the form $\forall(L^p : \alpha^{\Sigma}(p)) \rightarrow \exists(R^p : \alpha^{\Sigma}(p))$ and a new match (δ, m_{LI^*}) is produced from $(L^p : \alpha^{\Sigma}(p))$ to I^* but a corresponding match from $(R^p : \alpha^{\Sigma}(p))$ to I^* is missing.

ii gc is of the form $\forall(L^p : \alpha^{\Sigma}(p)) \rightarrow \exists(R^p : \alpha^{\Sigma}(p))$ and a required match from $(R^p : \alpha^{\Sigma}(p))$ to I is removed but a corresponding match from $(L^p : \alpha^{\Sigma}(p))$ to I still remains in I^*.

iii gc is of the form $\forall(L^p : \alpha^{\Sigma}(p)) \rightarrow \neg\exists(R^p : \alpha^{\Sigma}(p))$ and a new match is produced from the forbidden pattern $(R^p : \alpha^{\Sigma}(p))$ to I^*.

Case (i): r satisfies **Cond 1** since a new match for the condition pattern is produced. According to Algorithm 1, r must comply with either **P.C1#1** or **P.C1#2**. The pattern in **P.C1#2** has a *NAC* that prevents the existence of pattern that matches with $(L^p : \alpha^{\Sigma}(p))$. Since a new match with $(L^p : \alpha^{\Sigma}(p))$ is produced in (i), r must comply with **P.C1#1**. Therefore, when the rule is applied, a corresponding match (δ, m_{RI^*}) from $(R^p : \alpha^{\Sigma}(p))$ to I^* for the match (δ, m_{LI^*}) must exist. Therefore (I^*, ι_{I^*}) satisfies the graph constraint gc. Hence we reach to a contradiction.

Case (ii): This case is explained by considering three matches:

- $(\delta, m_{LI}) : (L^p : \alpha^{\Sigma}(p)) \rightarrow (I : S)$,
- $(\delta, m_{RI}) : (R^p : \alpha^{\Sigma}(p)) \rightarrow (I : S)$,
- $(\delta, m_{LI^*}) : (L^p : \alpha^{\Sigma}(p)) \rightarrow (I^* : S)$

where $m_c; m_{RI} = m_{LI}$, $m_{LI}(L^p) = m_{LI^*}(L^p)$ and there does not exist a corresponding match $(\delta, m_{RI^*}) : (R^p : \alpha^{\Sigma}(p)) \rightarrow (I^* : S)$ such that $m_c; m_{RI^*} = m_{LI^*}$. Therefore $\exists\ y \in (m_{RI}(R^p) \setminus m_{LI}(L^p))$ from which we obtain $Y \in \iota_c; \delta(R^p \setminus L^p)$ where Y is the type of y. Hence, r satisfies **Cond 2** and according to Algorithm 1, r must comply with either **P.C2#1** or **P.C2#2**. Pattern in **P.C2#2** has a *NAC* that prevents the existence of pattern that matches with $(L^p : \alpha^{\Sigma}(p))$. Since in case (ii), the matching with the condition pattern remains, the rule r must comply with **P.C2#1**. However, pattern **P.C2#1** makes sure that a corresponding match for the required pattern is produced which contradicts with the second case.

Case (iii): r satisfies **Cond 3** since a new match for the forbidden pattern is produced. According to Algorithm 1, r must comply with **P.C3#1**. Pattern **P.C3#1** has a *NAC* that prevents the existence of pattern that matches with $(R^p : \alpha^{\Sigma}(p))$. Therefore we reach to a contradiction.

In all three cases we have shown that if r is applied on a valid model $I \models \mathfrak{S}$ then the result (I^*, ι_{I^*}) cannot violate the constraints specified in $\mathfrak{S}$.

Complexity of Algorithm 1: The complexity of the algorithm depends on two factors: (i) the size of graph patterns of the graph constraints and (ii) the size of graph patterns in transformation rules. The size of a graph pattern refers to the number of vertices of the graph. The performance of the algorithm depends on injective matching. Finding an injective match from an n-vertex graph (G) to a m-vertex graph (H) has complexity $2^{O(n\ log\ m)}$ as finding all possible vertex subsets of H of size at most n is $m^{O(n)}$ and for each subset we need to try all possible mappings from G. The algorithm avoids processing the models of a system, therefore it is expected to analyze the transformation rules fast because in a typical situation, the size of graph patterns in graph constraints and transformation rules would be very small compared to the size of models.

Theorem 2 *Given a metamodel specification* $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ *and a set of conformance preserving rules* $\mathcal{R} = \{r_1, ...r_n\}$ *w.r.t a set of atomic constraints* $C^{\mathfrak{S}}$. *If the rules are applied on a valid model of* $\mathfrak{S}$ *a finite number of times, the result will be a valid model of* $\mathfrak{S}$.

Proof. The theorem can be proved by induction over the number of application of the transformation rules as the results produced in each step are valid models of $\mathfrak{S}$.

5 Evaluation

We have implemented a proof of concept (PoC) tool [1] for graph transformation, conformance checking, and detecting conformance preserving graph transformation rules. The tool takes a metamodel specification and model transformation rules as input in JSON format and provides a visualization of models and rules using a graph visualization software called 'Graphviz' [11]. The algorithm presented in section 4 has been implemented in the web-based PoC tool. Figure 17 shows a screenshot of the PoC tool where the user can visualize the model, transformation rules and can select predicates from a drop-down list. The mappings of the predicates to the model (i.e., constraint) are visualized by color matching. For example, the node 'W' from the predicate is mapped to the node 'A' in the model which is represented by the same color. The user can check for the conformance from the web-based tool. Note that the example presented in Figure 17 is the same as presented in Figure 8.

To evaluate the performance of our algorithm for checking conformance we performed experiments with models and model transformation rules of various size. Table 3 shows the results of the experiments. The table shows the size of the models and transformation rules being considered for the evaluation, and the time taken to detect conformance. Below is a description of the columns of the table:

- *Index No*: Experiment number;
- *vertex* + *edges*: Number of vertex and edges in the model;
- *LHS*: Total number of elements in the LHS of the rule;
- $X \cap RHS$: Number of common elements in X and RHS of the rule;
- $X \setminus (LHS \cup RHS)$: Number of unique elements in X of the rule;
- $LHS \setminus K$: Number of elements deleted by the rule;
- $RHS \setminus K$: Number of elements created by the rule;
- *time* (*in ms*): Time required to determine the conformance of the rule.

In experiment number $1-7$, we consider the models, graph constraints and transformation rules from Figure 8-11, and 13, 14, and 16, respectively. To evaluate the performance of the algorithm, we used a Java program to produce models and model transformation rules of increasing size. The Java program takes meta information such as the number of nodes and edges in the model, and number of nodes and edges in the LHS, and RHS of the transformation rule. The program also takes input specifying the percentage of nodes and edges that will be removed and produced by the rule. From the meta information, the program produces models and transformation rules in JSON format which is compatible with the PoC tool. Experiment number $8-12$ are performed over transformation rules with a small number of elements and models with increasing size; and experiment number $13-16$ are performed over models with small number of elements and transformation rules with increasing size. These experiments (i.e., $8-16$) are performed with one graph constraint using the predicate <*composite*>. The results indicate that the algorithm takes longer time to detect if we have

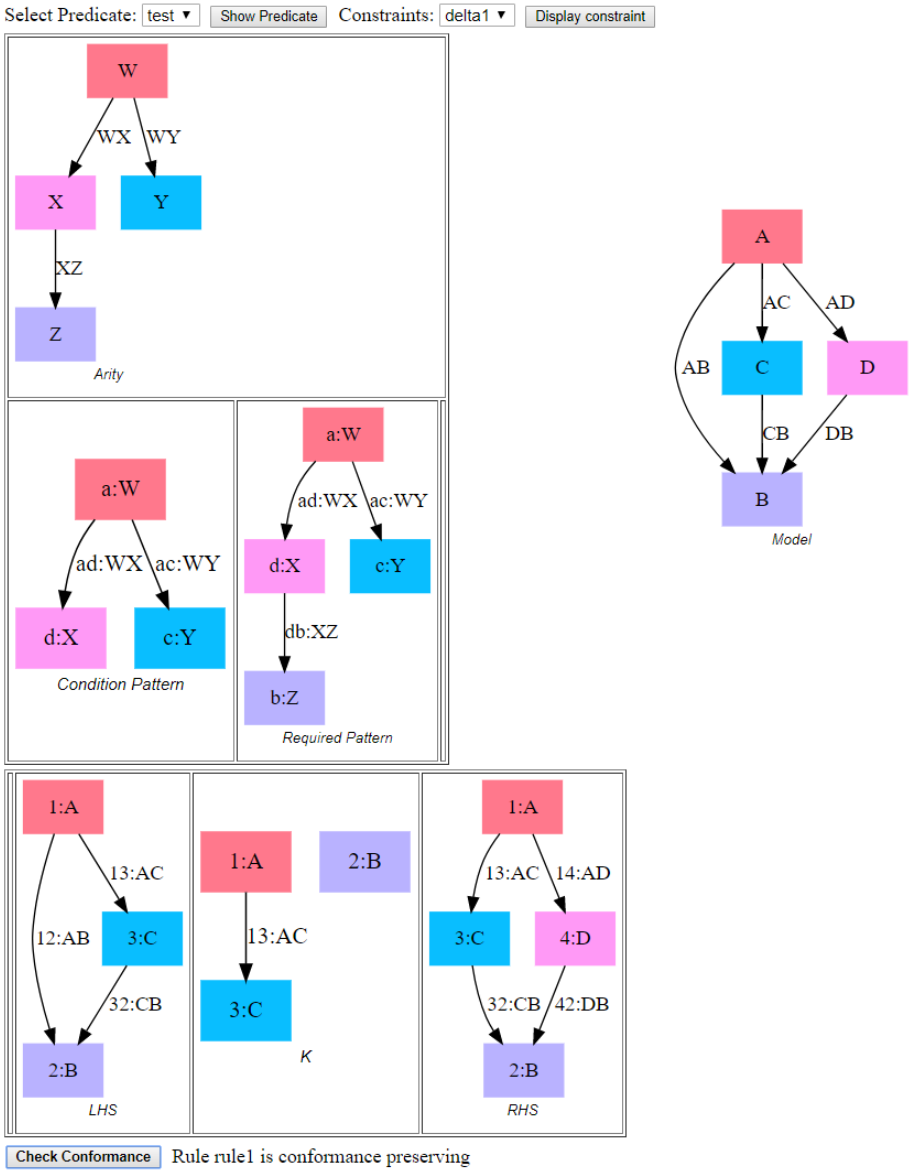


Fig. 17. Screenshot from the PoC tool

transformation rule with larger size. Experiment number 17 has been performed with a graph constraint using a predicate with 23 elements in the condition pattern and 26 elements in the required pattern. The longer time (approx. one minute) required for this experiment is expected as the algorithm need to check

Table 3. Time required for conformance checking in different settings

Index	*Size of model*	*Size of rule*					*Computation*
No	*vertex + edges*	*LHS*	$X \cap RHS$	$X \setminus (LHS \cup RHS)$	$LHS \setminus K$	$RHS \setminus K$	*time (in ms)*
1	4 + 5 = 9	6	0	0	2	4	12
2	4 + 5 = 9	6	0	0	2	6	14
3	6 + 8 = 14	6	4	2	4	5	11
4	7 + 9 = 16	6	4	4	4	5	12
5	4 + 4 = 8	6	0	0	3	4	12
6	6 + 7 = 13	8	2	2	6	3	11
7	6 + 7 = 13	6	4	2	4	5	12
8	4 + 7 = 11	3	4	2	1	5	11
9	8 + 12 = 20	3	4	2	2	6	12
10	12 + 18 = 30	3	4	2	2	6	11
11	10 + 200 = 210	3	4	3	2	6	11
12	50 + 200 = 250	3	4	3	2	6	12
13	4 + 8 = 12	23	18	16	16	23	23
14	4 + 8 = 12	23	29	16	4	23	31
15	4 + 8 = 12	23	49	36	4	43	145
16	4 + 8 = 12	43	63	36	10	43	250
17	15 + 38 = 53	47	71	36	6	43	61563

for various injective matches from the condition and required pattern of the constraint to the transformation rule.

6 Related work and conclusion

In this paper, we have presented a static analysis technique for checking the conformance property of transformation rules. The static analysis technique processes the semantics of graph constraints and analyzes if a transformation rule exhibits certain structure in order to be conformance preserving rule with respect to a set of constraints. We presented the idea in the context of DPF which provides a formal framework for metamodelling. We performed performance evaluation tests suggesting that the approach is promising for analyzing model transformation rules.

There has been a great deal of research related to the formal analysis of termination, confluence, functional behaviour of model transformation systems [15] [6] [13] [17] and tool support [2] [21]. One important difference between our approach and existing approaches is that our approach rest on diagrammatic logic. Our approach is closely related to the work of Heckel and Wagner [14]. They ensured consistency of graph transformations by automatically adding application conditions to single pushout (SPO) rules. They propose a technique for deriving application conditions from SPO rules of the form $L \xrightarrow{r} R$ and constraints. Constraints are specified in the form $P \xrightarrow{c} Q$ where P and Q are directed

graphs and c is an injective morphism. In their approach, a post-condition (i.e., an application condition over the right hand side of a rule) is constructed as a set of all right-sided constraints by generating all possible gluings of the premise P and the graph R. The post-condition is then used to construct a left-sided constraint (i.e, an application condition over L) by inverse decomposition of pushout diagrams. One issue with this approach is that a post-condition induced by a constraint may include a large number of right-sided constraints. A simple technique was presented in [14] to reduce the number of right-sided constraints from a post-condition. The idea of the reduction is based on the removal of right-sided constraints that is obtained from a gluing $R \xrightarrow{s} S \xleftarrow{p} P$ where the image of P in S does not depend on elements generated by rule r i.e., $p(P) \cap s(R - r(L)) = \emptyset$. This reduction technique however cannot handle situations where a rule deletes an element that matches with the required pattern Q of a constraint $c : P \rightarrow Q$ (see **Cond 2** of Figure 6). To illustrate this issue, consider an input graph G with $c; m_q = m_p$ where $m_p : P \rightarrow G$ and $m_q : Q \rightarrow G$ are two injective morphism. Now consider a rule $L \xrightarrow{r} R$ where $p(P) \cap s(R - r(L)) = \emptyset$ which means that the reduction will disregard the constraint c and no left-sided constraint will be constructed. But it is possible for the rule to remove an element x from $m_q(Q)$ which results in an output not conforming to its metamodel.

Later on, this approach for ensuring consistency was adapted for a double pushout approaches and generalized for high level transformation systems [10]. The approach was further enhanced for nested constraints in [12]. Although the approach presented in [10, 12] can deal with situations where a rule add/delete elements, the construction of application conditions do not include any reduction technique. This results in a large number of application conditions. In our approach, we rely on the modeller to develop transformation rules and automatically check conformance using our algorithm. The proposed algorithm filters out trivially conformance preserving rules as described in section 4.1 before checking the existence of the desired patterns in section 4.2 for optimal performance.

Becker et al. [5] developed a verification technique for structural safety property of a transformation system which is very similar to our approach in the sense that their technique is based on checking the locality of transformation rules against a set of safety properties. In their approach, the authors checked if the application of transformation rules can violate any safety property given as a set of forbidden graph patterns. Dyck and Giese [9] improved the technique for the automated verification of structural invariants for graph transformation systems by extending the expressive power. They provided support for negative application conditions in constraints and support for application conditions in transformation rules. However, both techniques only check against forbidden patterns while in our approach we support checking the conformance property of transformation rules against both required and forbidden patterns. Making sure that the application of a transformation rule does not violate any required pattern is more complex than checking against a set of forbidden patterns as it involves more scenarios to cover for the checking algorithm.

In future, we plan to adapt the algorithm to more expressive constraint language such as nested graph constraints. We also plan to enhance the proof of concept presented in this paper with more user interaction and visualization support.

Acknowledgement

The authors would like to thank the reviewers for their constructive comments on the earlier version of this paper.

References

1. Analysis of conformance preserving transformation rule, 2018. https://github.com/fazlRabbi/ConformanceTxRule.
2. T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. Henshin: Advanced Concepts and Tools for In-place EMF Model Transformations. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, MODELS'10, pages 121–135. Springer-Verlag, 2010.
3. A. K. Artemiadis, A. A. Vervainioti, E. Alexopoulos, A. Rombos, M. C. Anagnostouli, and C. Darviri. Stress management and multiple sclerosis: a randomized controlled trial. *Archives of clinical neuropsychology : the official journal of the National Academy of Neuropsychologists*, 27 4:406–16, 2012.
4. L. Baresi and P. Spoletini. *On the Use of Alloy to Analyze Graph Transformation Systems*, pages 306–320. Springer, 2006.
5. 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*, ICSE '06, pages 72–81, New York, NY, USA, 2006. ACM.
6. H. J. S. Bruggink, B. König, and H. Zantema. *Termination Analysis for Graph Transformation Systems*, pages 179–194. Springer, 2014.
7. S. A. da Costa and L. Ribeiro. Verification of graph grammars using a logical approach. *Science of Computer Programming*, 77(4):480 – 504, 2012.
8. Z. Diskin and U. Wolter. A diagrammatic logic for object-oriented visual modeling. *Electronic Notes in Theoretical Computer Science*, 203(6):19 – 41, 2008. Proceedings of the 2nd Workshop on Applied and Computational Category Theory (ACCAT 2007).
9. J. Dyck and H. Giese. *Inductive Invariant Checking with Partial Negative Application Conditions*, pages 237–253. Springer International Publishing, Cham, 2015.
10. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.
11. Graphviz. Graph visualization software, 2018. https://www.graphviz.org/.
12. A. Habel and K.-h. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical. Structures in Comp. Sci.*, 19(2):245–296, Apr. 2009.
13. R. Heckel, J. M. Küster, and G. Taentzer. *Confluence of Typed Attributed Graph Transformation Systems*, pages 161–176. Springer, 2002.
14. R. Heckel and A. Wagner. Ensuring Consistency of Conditional Graph Grammars - A Constructive Approach -. *ENTCS*, 2(C):118–126, 1995.

15. F. Hermann, H. Ehrig, F. Orejas, and U. Golas. *Formal Analysis of Functional Behaviour for Model Transformations Based on Triple Graph Grammars*, pages 155–170. Springer, 2010.
16. L. Lambers, H. Ehrig, U. Prange, and F. Orejas. *Embedding and Confluence of Graph Transformations with Negative Application Conditions*, pages 162–177. Springer, 2008.
17. D. Plump. Checking graph-transformation systems for confluence. *ECEASST*, 26, 2010.
18. L. Ribeiro, F. L. Dotti, S. A. da Costa, and F. C. Dillenburg. Towards theorem proving graph grammars using event-b. *ECEASST*, 30, 2010.
19. A. Rutle. *Diagram Predicate Framework: A Formal Approach to MDE*. PhD thesis, Department of Informatics, University of Bergen, Norway, 2010.
20. A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A formal approach to the specification and transformation of constraints in mde. *Journal of Logic and Algebraic Programming*, 81(4):422–457, 2012.
21. G. Taentzer. AGG: A graph transformation environment for modeling and validation of software. In J. L. Pfaltz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, Charlottesville, VA, USA, September 27 - October 1, 2003, Revised Selected and Invited Papers*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer, 2003.
22. J. Troya and A. Vallecillo. *Towards a Rewriting Logic Semantics for ATL*, pages 230–244. Springer, 2010.
23. D. Varró, G. Varró, and A. Pataricza. Designing the automatic transformation of visual languages. *Sci. Comput. Program.*, 44(2):205–227, Aug. 2002.
24. X. Wang, F. Büttner, and Y. Lamo. Verification of graph-based model transformations using alloy. *ECEASST*, 67, 2014.