Electronic Communications of the EASST
Volume 67 (2014)

Proceedings of the 13th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2014)

Generating Preconditions from Graph Constraints by Higher Order Graph Transformation

Frederik Deckwerth and Gergely Varró

13 pages

Guest Editors: Frank Hermann, Stefan Sauer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Generating Preconditions from Graph Constraints by Higher Order Graph Transformation

Frederik Deckwerth[1]* **and Gergely Varró**[2]†

[1] frederik.deckwerth@es.tu-darmstadt.de, [2] gergely.varro@es.tu-darmstadt.de
Real-Time Systems Lab
Technische Universität Darmstadt, Germany

Abstract: Techniques for the verification of structural invariants in graph transformation systems typically rely on the derivation of negative application conditions that are attached to graph transformation rules in order to avoid the runtime occurrence of forbidden structural patterns in the system model. In this paper, we propose a practical approach for this derivation process, which produces the required negative application conditions by applying higher order graph transformation on the rule specifications themselves. Additionally, we integrate filtering criteria into these higher order constructs to avoid, already at an early stage, the unnecessary construction of invalid and redundant negative application conditions.

Keywords: higher order graph transformation, static analysis, verification

1 Introduction

Graph transformation (GT) [Roz97] as a declarative technique to specify the rule-based manipulation of system models has been successfully employed in many practical, real-world applications [Hec98] including scenarios from the security domain [KMP02], where the formal nature of graph transformation plays an important role.

Role-based access control (RBAC) [SCFY96] is undoubtedly such a well-known security scenario, in which graph transformation can be applied as suggested in [KMP02]. In this setup, an access control policy is defined (i) by modeling authorization settings (e.g., users, roles, and permissions) in the system as graphs, and (ii) by specifying their modifications declaratively by graph transformation rules, whose application actually performs these changes on the graph-based authorization model.

An additional important and challenging task in an RBAC scenario is to ensure that a specified access control policy is always enforced. This can be supported by automated GT verification techniques [HW95], which requires (global) negative constraints representing forbidden structures that are never allowed to occur in any authorization settings.

While runtime verification techniques for graph transformation can efficiently detect the occurrences of these forbidden patterns by monitoring and reacting accordingly, this approach is unsatisfactory, if the policy is to be statically enforced in a proven and certified manner, e.g., in an auditing process carried out by the security team of a company or by external auditors. In

* Supported by CASED (www.cased.de).
† Supported by the DFG funded CRC 1053 MAKI.

such a case, static analysis techniques [HW95], which are used to derive information required to extend the left-hand side (LHS) of the graph transformation rules at design time by attaching new negative application conditions (NAC) [HHT96], are more appropriate. The attached NACs, generated from the given negative constraints, avoid rule applications at runtime that would lead from a policy compliant authorization setting to a policy violating one.

The design-time production of graph transformation rules with a NAC-enriched left-hand side is carried out by a sophisticated algorithm [HW95] in the following three steps. (1) Each subgraph of each negative constraint is glued to the right-hand side (RHS) of the GT rules (in a type-conform manner) as a postcondition NAC. This forbids a situation where the negative constraint is violated *after* the rule application. (2) The postcondition NAC is back-propagated to the LHS as a regular (precondition) NAC by reverting the modifications specified by the GT rule. (3) Finally, the set of GT rules with (precondition) NACs is filtered to discard redundant NACs, where the negative constraint is already violated *before* the rule application.

Although the original algorithm [HW95] and its conceptual derivatives [EEHP04] present many interesting theoretical considerations, the implementation of such a technique either remains an open issue, or requires additional logic or category theory based libraries.

In this paper, we propose a practical approach, which can be implemented by any existing graph transformation tool that considers rule specifications as regular models, to produce graph transformation rules with negative application conditions. The proposed technique derives higher order graph transformation rules from the negative constraints, and applies these higher order constructs to the original GT rule specifications to attach appropriate NACs. Additionally, we identify filtering conditions at an early stage and integrate them into the higher order rules to be able to avoid the construction of redundant (precondition) NACs.

The remainder of the paper is structured as follows: Section 2 introduces basic modeling and graph transformation concepts. The general precondition NAC derivation process is sketched in Sec. 3, while Sec. 4 presents our algorithm using higher order transformations to construct these application conditions. Related work is discussed in Sec. 5, and Sec. 6 concludes our paper.

2 Modeling and Graph Transformation Concepts

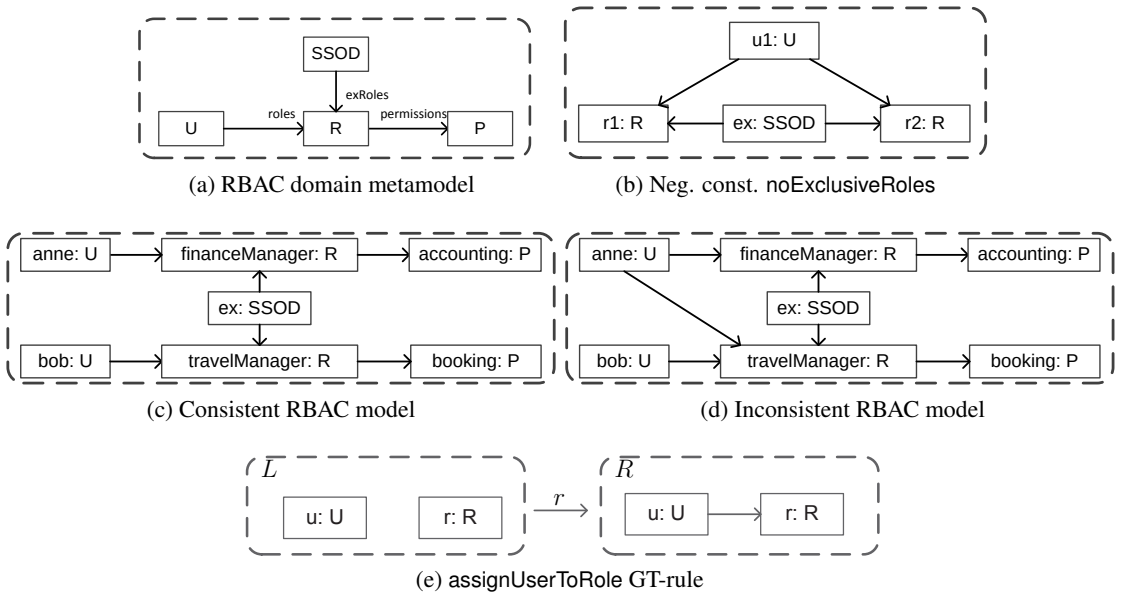
This section introduces the basic concepts of metamodels, models and graph transformation.

Graphs and Graph Morphisms. A *graph* $G = (V_G, E_G, s_G, t_G)$ consists of a set of vertices V_G and edges E_G, and two functions $s_G, t_G : E_G \to V_G$, which map edges to their source and target vertices, respectively. A *total graph morphism f from graph G to H* is a pair of total functions $f_V : V_G \to V_H$, $f_E : E_G \to E_H$ that map vertices and edges in a structure preserving manner, i.e., $f_V \circ s_G = s_H \circ f_E$ and $f_V \circ t_G = t_H \circ f_E$. A *graph morphism f is injective*, if both f_V and f_E is are injective functions. A *partial graph morphism f from G to H* is a total graph morphism from some subgraph of G.

Metamodels and Models. A *metamodel* defines the core concepts of a domain, while a *model* describes an actual system. Formally, a *metamodel MM* is a graph, whose vertices and edges are *classes* and *associations*, respectively. A *model M typed by the metamodel MM* is also a graph, whose vertices and edges (referred to as *objects* and *links*) can be mapped to classes and associations, respectively, by a total graph morphism $type : M \to MM$. A *typed (graph)*

morphism $f : M \to M'$ *from model* M *to* M' *that are both typed by metamodel* MM is a graph morphism, which preserves type information, i.e., $type \circ f = type'$.

Example. A simplified role-based access control (RBAC) scenario inspired by [KMP02] is used throughout the paper as a running example. Figure 1a shows the RBAC metamodel, which consists of the classes user (U), role (R), permission (P), and the static separation of duty relation (SSOD). The association roles defines the roles belonging to a user and the association permissions specifies the permissions assigned to a role. The association exRoles connects the SSOD relation to roles that can only be granted exclusively. Figure 1c shows an RBAC model, which contains users anne and bob having the roles financeManager and travelManager, respectively. These roles can be granted exclusively, expressed by the SSOD relation ex. The permissions accounting and booking are assigned to the roles financeManager and travelManager, respectively.



(a) RBAC domain metamodel

(b) Neg. const. noExclusiveRoles

(c) Consistent RBAC model

(d) Inconsistent RBAC model

(e) assignUserToRole GT-rule

Figure 1: The role-based access control application scenario

Patterns, Negative Constraints and Model Consistency. A *pattern* P is a graph typed by a metamodel MM. A *pattern* P *matches a model* M if there exists a total injective typed morphism $m : P \to M$ called match morphism. The subgraph $m(P) \subseteq M$ is called match of P in M.

A *negative constraint* NC is a pattern to declaratively define forbidden subgraphs in a model. A model M is consistent with respect to a negative constraint NC, if NC does not match M. For simplicity, we will call a model just consistent if it is clear which constraints are meant.

Example. Figure 1b shows a negative constraint that forbids a user to have two exclusive roles. The model depicted in Figure 1c is consistent, as neither anne, nor bob has two exclusive roles. In contrast, the model in Figure 1d is inconsistent, as anne has both the financeManager and the travelManager roles, which are exclusive according to the SSOD relation ex.

Graph Transformation Rules and Their Application. A *graph transformation rule* $\mathbf{r} = (\mathcal{N}^{\triangleleft} \leftarrow L \to R \to \mathcal{N}^{\triangleright})$ consists of a left hand side (LHS) pattern L and a right hand side (RHS)

pattern R and the (possibly empty) sets of pre- and postcondition negative application conditions (NACs) $\mathcal{N}^{\triangleleft}$ and $\mathcal{N}^{\triangleright}$. The LHS and the RHS are related by the typed partial morphism $r : L \to R$. The pre- and postcondition NACs $N^{\triangleleft} \in \mathcal{N}^{\triangleleft}$ and $N^{\triangleright} \in \mathcal{N}^{\triangleright}$ are related to L and R by typed total injective morphisms $n^{\triangleleft} : L \to N^{\triangleleft}$ and $n^{\triangleright} : R \to N^{\triangleright}$.

A rule $\mathbf{r} = (\mathcal{N}^{\triangleleft} \leftarrow L \to R \to \mathcal{N}^{\triangleright})$ *is applicable* to a model M iff (i) there exists a match $m : L \to M$ of the LHS pattern L of $\mathbf{r}$ in M, and (ii) the precondition NACs $N^{\triangleleft} \in \mathcal{N}^{\triangleleft}$ are satisfied. A precondition NAC $N^{\triangleleft}$ is satisfied if the current match $m(L)$ cannot be extended to $m^{**}(N^{\triangleleft})$ in M, i.e., $\nexists\, m^{**} : N^{\triangleleft} \to M : m = m^{**} \circ n^{\triangleleft}$.

A rule $\mathbf{r} = (\mathcal{N}^{\triangleleft} \leftarrow L \to R \to \mathcal{N}^{\triangleright})$ *is applied* at a match m in model M to obtain the modified model M' (denoted by $M \overset{\mathbf{r@m}}{\Longrightarrow} M'$), by constructing the *pushout* of the match morphism $m : L \to M$ and morphism $r : L \to R$ of rule $\mathbf{r}$ defined (informally) as follows: (i) Remove the vertices and edges $m(L) \subseteq M$ identified by the match m that are not present in the RHS pattern R. (ii) Add the vertices and edges to M that are part of the RHS pattern R, but not present in the LHS pattern L. The vertices and edges that occur in L and R are preserved. Dangling edges in M' are deleted. The violation of a postcondition NAC $N^{\triangleright} \in \mathcal{N}^{\triangleright}$ indicates that the derived model M' is inconsistent. [1]

Example. Figure 1e shows the GT-rule assignUserToRole that assigns a user u to a role r by adding a new roles link. Morphisms r, $n^{\triangleleft}$, and $n^{\triangleright}$ are implicitly specified in all the figures of the running example by matching vertex labels. The result of applying the GT-rule assignUserToRole to user anne and role travelManager in the model of Figure 1c is depicted in Figure 1d.

Consistency Preservation. A rule $\mathbf{r}$ is *consistency preserving*, if for any arbitrary model M and all possible applications $M \overset{\mathbf{r}}{\Rightarrow} M'$ of rule $\mathbf{r}$ holds: If M is consistent then M' is also consistent. A set of rules $\mathcal{R}$ is consistency preserving if all rules $\mathbf{r} \in \mathcal{R}$ are consistency preserving.

The idea of consistency preservation is that if a set of rules is consistency preserving and the start graph is consistent, then consistency is preserved for all models derivable by a sequence of rule applications, as none of the rules will transform a consistent to an inconsistent model.

3 Building Consistency Preserving Graph Transformation Rules

This section recapitulates the static analysis technique proposed in [HW95], which extends graph transformation rules with precondition NACs to produce consistency preserving GT-rules. The construction of precondition NACs can be carried out in the following three steps:

1. **Constructing postcondition NACs.** For each non-empty subgraph of a negative constraint that is also a subgraph of the RHS pattern of a GT-rule, a postcondition NAC is constructed by gluing the negative constraint and the RHS together along the common subgraph. The postcondition NACs represent all those situations where the negative constraint is violated after the rule application.
2. **Back-propagating postcondition NACs into precondition NACs.** Each postcondition NAC constructed in the previous step is back-propagated to the LHS pattern as a precondition NAC by reverting the modifications specified by the GT-rule. The precondition NACs can prevent the GT-rule from transforming a consistent model into an inconsistent one.
3. **Filtering precondition NACs.** Redundant or invalid precondition NACs are discarded in

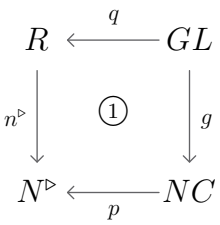
[1] Note that in real applications postcondition NACs are of little use as they cannot prevent the derivation of inconsistent models. However, they are used as an intermediate step for the construction of precondition NACs.

the following two cases. (i) If the complete negative constraint appears in the precondition NAC, then such a construct could block rule application only on inconsistent models, which can never be created by applying consistency preserving rules on a consistent initial model. Consequently, such a precondition NAC can be considered *redundant* (assuming a consistent initial model and a set of consistency preserving rules is used). (ii) If an edge in the postcondition NAC that originates exclusively from the negative constraint is adjacent to a node that is deleted by the back-propagation, then a (structurally) *invalid* precondition NAC with dangling edges is produced.

As [HW95] states, the filtering conditions for precondition NACs in Step 3 can also be formulated on the corresponding postcondition NACs, consequently, an adapted filtering step can already be carried out directly after Step 1 in the previous process. Therefore, in the rest of the paper, we only consider *the first two phases of this* ***alternative*** *approach*, namely, the *construction (Sec. 3.1) and filtering (Sec. 3.2) of postcondition NACs.*

3.1 Construction of Postcondition NACs

A non-empty graph GL is a *gluing graph* of a RHS pattern R and a negative constraint NC if it is a subgraph of both R and NC, i.e., there exists at least a pair of total injective typed morphisms $R \xleftarrow{q} GL \xrightarrow{g} NC$.



The postcondition NAC patterns $N^{\triangleright} \in \mathcal{N}^{\triangleright}$ of R and NC can be constructed as pushouts ① over morphisms $R \xleftarrow{q} GL \xrightarrow{g} NC$ for all gluing graphs GL of R and NC, and all total injective typed morphisms q and g.

Example. The postcondition NACs derived from the negative constraint NoExclusiveRoles (Fig. 1b) and the RHS of rule AssignUserToRole (Fig. 1e) are shown in Figure 2. Vertices and edges are drawn solid if they belong to the RHS. Dashed vertices and edges denote the elements of the negative constraint NoExclusiveRoles. The gluing graph is drawn bold.

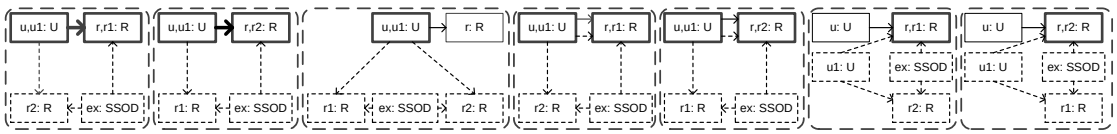


Figure 2: Postcondition NACs from neg. const. noExclusiveRoles and rule assignUserToRole

3.2 Filter Conditions for Postcondition NACs

The back-propagation of postcondition NACs into redundant or invalid precondition NACs can be avoided if the filtering is carried out on the postcondition NACs right after their construction in Step 1. A postcondition NAC $N^{\triangleright}$ of the RHS pattern R of rule $\mathbf{r}$ and the negative constraint NC, is discarded if either of the following conditions is fulfilled:

(i) *Preserving Condition.* All elements of the gluing graph GL are preserved by the reverse application of rule $\mathbf{r}$ that would yield a precondition NAC that contains the complete negative

constraint NC. Formally, $\forall v_{GL} \in V_{GL}, \exists v_L \in V_L : r(v_L) = q(v_{GL})$, and $\forall e_{GL} \in E_{GL}, \exists e_L \in E_L : r(e_L) = q(e_{GL})$.

(ii) *Dangling Edge Condition.* There exists an edge e in that part of the postcondition NAC $N^{\triangleright}$, which originates only from the negative constraint NC (formally, $e \in E_{N^{\triangleright}} \setminus n_E^{\triangleright}(R)$), whose source $s_{N^{\triangleright}}(e)$ or target $t_{N^{\triangleright}}(e)$ vertex would be deleted by the reverse application of rule $\mathbf{r}$, making e a dangling edge in the precondition NAC (formally, $\exists v \in (V_R \setminus r_V(V_L)) : s_{N^{\triangleright}}(e) = n_V^{\triangleright}(v) \vee t_{N^{\triangleright}}(e) = n_V^{\triangleright}(v)$).

Let the *dangling point* graph DP consist of all vertices $v \in V_{GL}$ whose image $g_V(V_{DP}) \subseteq V_{NC}$ in the negative constraint is source or target of an edge that is not in the gluing graph (formally, $V_{DP} = \{v \in V_{GL} \mid \exists e \in E_{NC} \setminus g_E(E_{GL}) : s_{NC}(e) = g_V(v) \vee t_{NC}(e) = g_V(v)\}$).

Lemma 1 *A postcondition NAC $N^{\triangleright}$ constructed by the pushout ① over morphisms $R \xleftarrow{q} GL \xrightarrow{g} NC$ for gluing graph GL fulfills the dangling edge condition iff the image $q(DP) \subseteq R$ contains at least one vertex that is deleted during the reverse application of rule* $\mathbf{r}$.

Proof. "⇒": As the dangling edge condition is fulfilled, there exists a vertex $v \in V_R \setminus r_V(V_L)$ s.t. there exists $e \in E_{N^{\triangleright}} \setminus n_E^{\triangleright}(R)$ with $s_{N^{\triangleright}}(e) = n_V^{\triangleright}(v) \vee t_{N^{\triangleright}}(e) = n_V^{\triangleright}(v)$. As $N^{\triangleright}$ is constructed by pushout ① there exists an edge $e' \in E_{NC}$ such that $p_E(e') = e$ and with a source or target vertex $v' \in V_{NC}$ such that $p_V(v') = n_V^{\triangleright}(v)$. Hence, there exists a vertex $v'' \in V_{GL}$ with $g_V(v'') = v'$ and $q_V(v'') = v$ that is in V_{DP}, as from the pushout ① and the injectivity of g follows that $e' \in E_{NC} \setminus g_E(E_{GL})$.

"⇐": There exists a vertex $v \in V_{DP} \subseteq V_{GL}$, whose image $q_V(v) \in V_R \setminus r_V(V_L)$ is deleted by the reverse rule application. From $v \in V_{DP}$, we can conclude that there exists an edge $e \in E_{NC} \setminus g_E(E_{GL})$ such that $s_{NC}(e) = g_V(v) \vee t_{NC}(e) = g_V(v)$. From the injectivity of q we know that there exists an edge $e' \in E_{N^{\triangleright}}$ s.t. $e' = p_E(e)$ whose source or target vertex $v' = q_V \circ n_V^{\triangleright}(v)$. From pushout ① and the injectivity of g follows that e' must be in $E_{N^{\triangleright}} \setminus n_E^{\triangleright}(R)$, as e has no preimage in GL. Thus, the dangling edge condition is fulfilled. □

Example. All but the first two postcondition NACs shown in Figure 2 are discarded due to the preserving condition, as the gluing graph does not contain the edge between user u and role r that is deleted by the reverse application.

4 Deriving Postcondition NACs by Higher Order Transformation

In this section, we present an approach for the automated construction of postcondition NACs using higher order (HO) graph transformation, i.e., transformations that can be applied on graph transformation rules. Our technique first constructs higher order transformation rules (HO-rules) either from the subgraphs of the negative constraints or from the GT-rules, and then these derived HO-rules are applied to the original GT-rules to produce exactly those postcondition NACs that violate either the preserving or the dangling edge condition. Consequently, postcondition NACs that would result later in redundant or invalid precondition NACs are *not produced* in our approach. Moreover, as a result of this early filtering, the approach is basically suitable (i.e., with minor changes) to directly construct precondition NACs, which is not discussed here due to space limitations.

Although the rest of this section deals with theoretical topics presenting the approach and proving its correctness, it should be highly emphasized that HO-rules can be easily specified in a graph transformation tool (like *e*Moflon [ALPS11]), in which GT-rules are represented as regular models. These HO-rules can be applied by the existing machinery facilitating the construction and the filtering of postcondition NACs.

4.1 Higher Order Transformation

A higher order (HO) transformation is a transformation applied on graph transformation (GT) rule specifications. Higher order transformation is now introduced analogously to the concepts of graph transformation.

Higher Order Morphism (HO-morphism) and Higher Order Pattern (HO-Pattern). A HO-morphism $\hat{f} : \mathbf{r} \to \mathbf{r}'$ that relates the GT-rules $\mathbf{r} = (\mathcal{N}^{\triangleleft} \xleftarrow{n^{\triangleleft}} L \xrightarrow{r} R \xrightarrow{n^{\triangleright}} \mathcal{N}^{\triangleright})$ and $\mathbf{r}' = (\mathcal{N}^{\triangleleft\prime} \xleftarrow{n^{\triangleleft\prime}} L' \xrightarrow{r'} R' \xrightarrow{n^{\triangleright\prime}} \mathcal{N}^{\triangleright\prime})$ is a tuple $\hat{f} = (\mathbf{f}_{N^{\triangleleft}}, f_L, f_R, \mathbf{f}_{N^{\triangleright}})$ consisting of the typed graph morphisms $f_L : L \to L'$ and $f_R : R \to R'$, and the sets $\mathbf{f}_{N^{\triangleleft}}$ and $\mathbf{f}_{N^{\triangleright}}$ of typed graph morphism of the form $f_{N^{\triangleleft}} : N^{\triangleleft} \to N^{\triangleleft\prime}$ and $f_{N^{\triangleright}} : N^{\triangleright} \to N^{\triangleright\prime}$, respectively. Morphisms f_L and f_R relate the LHS and RHS patterns of the GT-rules $\mathbf{r}$ and $\mathbf{r}'$ such that $f_R \circ r = r' \circ f_L$. The morphisms $f_{N^{\triangleleft}} \in \mathbf{f}_{N^{\triangleleft}}$ and $f_{N^{\triangleright}} \in \mathbf{f}_{N^{\triangleright}}$ relate the pre- and postcondition NACs of $\mathbf{r}$ and $\mathbf{r}'$ such that $f_{N^{\triangleleft}} \circ n^{\triangleleft} = n^{\triangleleft\prime} \circ f_L$ and $f_{N^{\triangleright}} \circ n^{\triangleright} = n^{\triangleright\prime} \circ f_R$ for all $f_{N^{\triangleleft}} \in \mathbf{f}_{N^{\triangleleft}}$ and $f_{N^{\triangleright}} \in \mathbf{f}_{N^{\triangleright}}$. A HO-morphism $\hat{f}$ is total (injective) if all graph morphisms in $\{\mathbf{f}_{N^{\triangleleft}}, f_L, f_r, \mathbf{f}_{N^{\triangleright}}\}$ are total (injective). A HO-morphism $\hat{f}$ is partial if at least one graph morphism in $\{\mathbf{f}_{N^{\triangleleft}}, f_L, f_R, \mathbf{f}_{N^{\triangleright}}\}$ is partial.

A *HO-pattern* $\hat{P} = (\mathcal{N}^{\triangleleft} \leftarrow L \to R \to \mathcal{N}^{\triangleright})$ can be matched to a rule $\mathbf{r}$ if there exist a total injective HO-morphism $\hat{m} : \hat{P} \to \mathbf{r}$.

Higher Order Transformation Rules (HO-rules) and Their Application. A *HO-rule* $\hat{\mathbf{r}} = (\hat{N}^{\triangleleft} \leftarrow \hat{L} \to \hat{R} \to \hat{N}^{\triangleright})$ consists of the left and right hand HO-patterns $\hat{L}$ and $\hat{R}$ and optionally a higher order pre- and postcondition NAC, defined by the HO-patterns $\hat{N}^{\triangleleft}$ and $\hat{N}^{\triangleright}$. The HO-LHS and RHS patterns are related by the partial HO-morphism $\hat{r} : \hat{L} \to \hat{R}$, while the higher order pre- and postcondition NACs are related to $\hat{L}$ and $\hat{R}$ by HO-morphisms $\hat{n}^{\triangleleft} : \hat{L} \to \hat{N}^{\triangleleft}$ and $\hat{n}^{\triangleright} : \hat{R} \to \hat{N}^{\triangleright}$.

A HO-rule $\hat{\mathbf{r}}$ *is applicable to a GT-rule* $\mathbf{r}$ iff (i) there exists a HO-match $\hat{m} : \hat{L} \to \mathbf{r}$ of the HO-LHS pattern $\hat{L}$ in GT-rule $\mathbf{r}$ that (ii) cannot be extended to $\hat{m}^{**} : \hat{N}^{\triangleleft} \to \mathbf{r}$ s.t. $\hat{m} = \hat{m}^{*} \circ \hat{n}^{\triangleleft}$.

A HO-rule $\hat{\mathbf{r}}$ *is applied* at HO-match $\hat{m}$ in GT-rule $\mathbf{r}$ to obtain the modified rule $\mathbf{r}'$ (denoted by $\mathbf{r} \overset{\hat{\mathbf{r}}@\hat{m}}{\Longrightarrow} \mathbf{r}'$) by constructing the *pushout* of the HO-match $\hat{m} : \hat{L} \to r$ and HO-morphism $\hat{r} : \hat{L} \to \hat{R}$ of HO-rule $\hat{\mathbf{r}}$, while dangling edges are deleted.

Example. Figure 3 shows a HO-rule (RHS pattern copying HO-rule) with a HO-LHS pattern $\hat{L}$ and a HO-RHS pattern $\hat{R}$. The HO-rule is applicable to the assignUserToRole GT-rule, as there exists a HO-match $\hat{m}$ that maps the user u and the role r, as well as the edge from LHS HO-pattern to the RHS pattern of the GT-rule. The rule is applied at match $\hat{m}$ by constructing the pushout of the HO-match $\hat{m}$ and the HO-morphism $\hat{r}$, i.e., by adding the elements present in the HO-RHS pattern $\hat{R}$ but not in the HO-LHS pattern $\hat{L}$ (no elements are deleted by the HO-rule). The result is the modified GT-rule assignUserToRole' (shown in the bottom right corner of Figure 3) equipped with a postcondition NAC that is identical to its RHS.

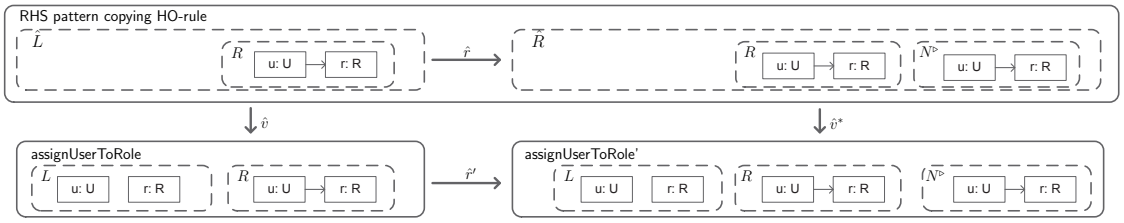


Figure 3: Higher Order Transformation Rule Application

4.2 Derivation of Higher Order Rules

The process of constructing the postcondition NACs is carried out by the following three kinds of HO-rules: (I) The filter criterion checking HO-rules determine the gluing graph and ensure the violation of the preserving and the dangling edge constraints. (II) The RHS pattern copying HO-rules add a postcondition NAC identical to the RHS pattern of the GT-rule. (III) The constraint gluing HO-rules glue the negative constraint to the postcondition NAC (created by the RHS pattern copying rule) along the gluing graph identified by the filter criterion checking rule.

(I + III) Deriving filter criterion checking and constraint gluing HO-rules from negative constraints. For each negative constraint NC, all its subgraphs $GL^* \subseteq NC$ are derived as *gluing graph candidates*. Note that gluing graph candidates GL^* are not necessarily subgraphs of the RHS pattern. Consequently, there exists an injective typed morphism $g : GL^* \to NC$, but there need not be an injective mapping $q : GL^* \to R$.

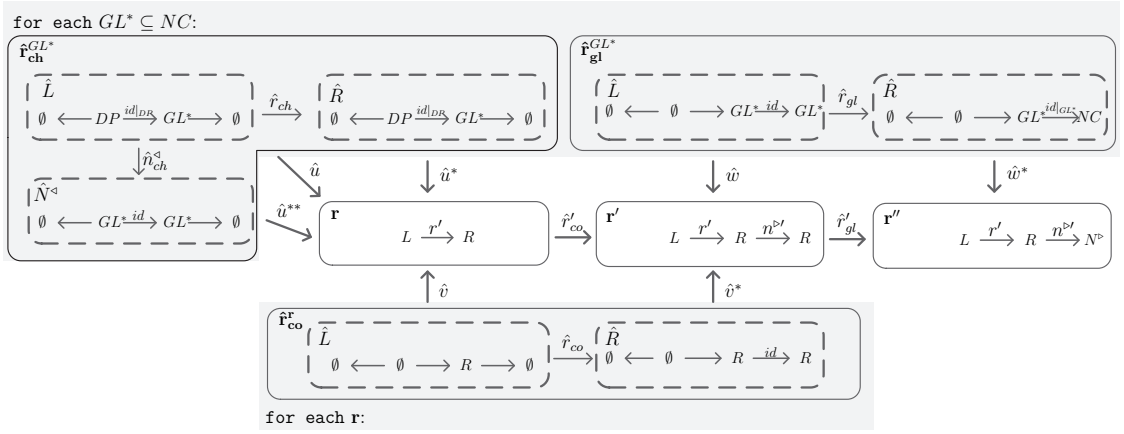


Figure 4: Derivation and application of HO-rules for constructing the postcondition NACs

For each gluing graph candidate $GL^* \subseteq NC$, a filter criterion checking HO-rule $\hat{\mathbf{r}}_{\mathbf{ch}}^{GL^*}$ is derived (top left corner of Figure 4), which consists of a HO-LHS, a HO-RHS and HO-precondition NAC patterns $\hat{L} = \hat{R} = (\emptyset \leftarrow DP \xrightarrow{id|_{DP}} GL^* \to \emptyset)$ and $\hat{N}^{\triangleleft} = (\emptyset \leftarrow GL^* \xrightarrow{id} GL^* \to \emptyset)$, respectively. The HO-LHS $\hat{L}$ ensures that a filter criterion checking HO-rule $\hat{\mathbf{r}}_{\mathbf{ch}}^{GL^*}$ is only applicable to GT-rule $\mathbf{r}$ if the gluing graph GL^* is contained in its RHS and the dangling points $DP \subseteq GL^*$ are preserved during reverse rule application to prevent the occurrence of dangling edges (see dangling edge

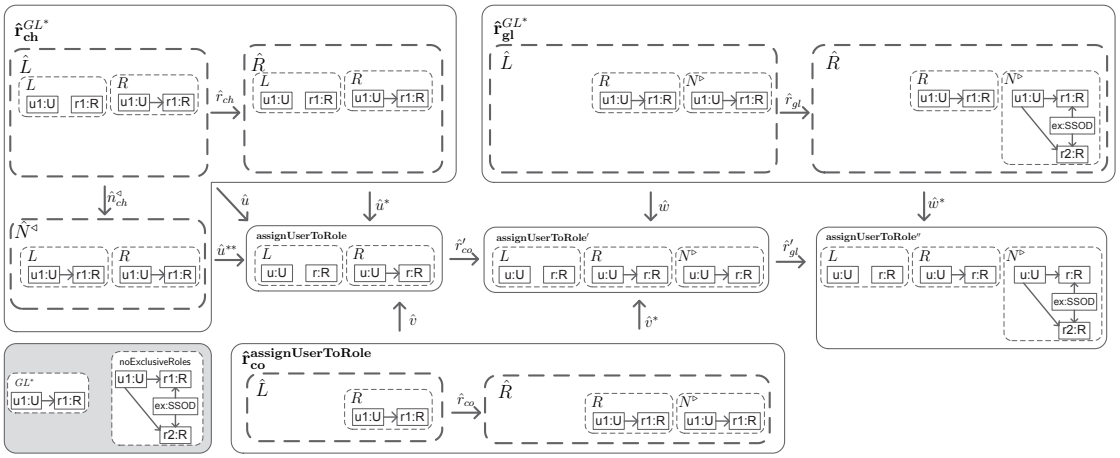


Figure 5: HO-rule Derivation and Application Example

condition and Lemma 1). The HO-precondition NAC $\hat{N}^{\triangleleft}$ prevents the application of HO-rule $\hat{\mathbf{r}}_{\mathbf{ch}}^{GL^*}$ if all elements identified by the gluing graph are preserved by the reverse application of GT-rule r, which is fulfilled as they appear on the LHS and RHS of GT-rule $\mathbf{r}$ (see preserving condition). Morphism $id|_{DP}$ is the identity morphism of GL^* restricted to DP, i.e. $id|_{DP}(v) = v$ for all vertices $v \in DP$.

For each gluing graph candidate $GL^* \subseteq NC$, a constraint gluing HO-rule $\hat{\mathbf{r}}_{\mathbf{gl}}^{GL^*}$ is derived (top right corner of Figure 4), which consists of the HO-RHS and HO-postcondition NAC patterns $\hat{R} = (\emptyset \leftarrow \emptyset \rightarrow GL^* \xrightarrow{id} GL^*)$ and $\hat{N}^{\triangleright} = (\emptyset \leftarrow \emptyset \rightarrow GL^* \xrightarrow{id|_{GL^*}} NC)$, respectively.

(II) Deriving RHS pattern copying HO-rules from GT-rules. For each GT-rule $\mathbf{r}$, a RHS pattern copying HO-rule $\hat{\mathbf{r}}_{\mathbf{co}}^{\mathbf{r}}$ is derived (bottom of Figure 4), which consists of HO-LHS and HO-RHS patterns $\hat{L} = (\emptyset \leftarrow \emptyset \rightarrow R \rightarrow \emptyset)$ and $\hat{R} = (\emptyset \leftarrow \emptyset \rightarrow R \xrightarrow{id} R)$, respectively.

Example. The top part of Fig. 5 presents the filter criterion checking and the constraint gluing HO-rules that were derived from the gluing graph candidate and the negative constraint noExclusiveRoles depicted in the grey box at the bottom left corner. The RHS pattern copying HO-rule derived from the GT-rule assignUserToRole is shown at the bottom of Figure 5.

4.3 Application of Higher Order Rules

For each pair of a gluing graph candidate GL^* and a GT-rule $\mathbf{r}$, the following steps are carried out (as shown in Figure 4) to construct postcondition NACs from the GT-rules:

(1) The applicability of the filter criterion checking HO-rule $\hat{\mathbf{r}}_{\mathbf{ch}}^{GL^*}$ to GT-rule $\mathbf{r}$ is checked by looking for a HO-match $\hat{u}$ of $DP \xrightarrow{r} GL^*$ in the GT-rule $\mathbf{r}$ that cannot be extended to $GL^* \xrightarrow{r} GL^*$ in $\mathbf{r}$ such that $\hat{u}^{**} \circ \hat{n}_{ch}^{\triangleleft} = \hat{u}$. This ensures that the gluing graph candidate is really a gluing graph of the RHS pattern of GT-rule $\mathbf{r}$ and the postcondition NAC, and it will not be discarded due to violation of the dangling edge or preserving condition (shown later in the proof of Theorem 1).

For each HO-match $\hat{u}$ identified in the previous step, to which $\hat{\mathbf{r}}_{\mathbf{ch}}^{GL^*}$ is applicable :

(2) The RHS pattern copying HO-rule $\hat{\mathbf{r}}^{\mathbf{r}}_{\mathbf{co}}$ is applied to GT-rule $\mathbf{r}$, which copies the RHS pattern of rule $\mathbf{r}$ as a postcondition NAC into the produced rule $\mathbf{r}'$.

(3) The constraint gluing HO-rule $\hat{\mathbf{r}}^{GL^*}_{\mathbf{gl}}$ is applied to the GT-rule $\mathbf{r}'$, which glues the negative constraint to the postcondition NAC created in step (2) along the gluing graph identified in step (1). This is achieved by choosing typed morphism components w_R and $w_{N^\triangleright}$ of the HO-match $\hat{w}$ such that $w_R(GL^*) = u_R(GL^*)$ and $w_{N^\triangleright}(GL^*) \subseteq v^*_{N^\triangleright}(R)$, where $u_R(GL^*)$ is the part of the RHS pattern of GT-rule $\mathbf{r}'$ identified by match $u_R(GL^*)$ in step (1), and $v^*_{N^\triangleright}(R)$ identifies the elements of the postcondition NAC created in step (2).

Example. The filter criterion checking HO-rule $\hat{r}_{ch}$ (top left corner of Fig. 5) is applicable to the GT-rule assignUserToRole (in the middle) as there exists a match for the HO-LHS pattern of $\hat{r}_{ch}$ in the GT-rule (mapping u1 and r1 to u and r, respectively), but no match for the HO-precondition NAC $\hat{N}^\triangleleft$. Consequently, the RHS pattern copying HO-rule adds a copy of the RHS pattern as postcondition NAC to the assignUserToRole GT-rule. Finally, the constraint gluing HO-rule is applied adding the elements in the negative constraint but not in the gluing graph.

Correctness of the approach. The overall correctness of the approach (including the filtering conditions on and the back-propagation of postcondition NACs) has been shown in [HW95]. Hence, we limit ourselves to argue that our higher order transformation constructs postcondition NACs as described in Sec. 3.1, and produces neither redundant nor invalid postcondition NACs.

It is easy to see that the application of the RHS pattern copying and the constraint gluing HO-rules constructs the correct postcondition NAC if the gluing graph has already been identified. Thus, it is enough to prove that the filter criterion checking HO-rule determines the gluing graph correctly and ensures that neither redundant nor invalid postcondition NACs will be produced.

Theorem 1 *A filter criterion checking HO-rule is applicable iff (i) the gluing graph candidate is really a gluing graph of the GT-rule and the negative constraint, and the postcondition NAC (going to be constructed for the gluing graph) will be preserved as it fulfills (ii) neither the preserving condition, (iii) nor the dangling edge condition.*

Proof. The HO-rule $\hat{\mathbf{r}}_{\mathbf{ch}}$ is applicable to a GT rule $\mathbf{r}$ iff a HO-match morphism $\hat{u}$ exists that cannot be extended to the HO-match $\hat{u}^{**}(\hat{N}^\triangleleft)$ of the HO-precondition NAC.

(i) As $\hat{u} = (u_L, u_R)$ is total and injective, its typed morphism component $u_R : GL^* \to R$ is also total and injective. This means that the gluing graph candidate GL^* is a subgraph of the RHS pattern R of GT-rule $\mathbf{r}$ and, therefore, a gluing graph of R and NC iff u_R exists.

(ii) The gluing graph identified by $u_R(GL^*)$ appears completely in both the LHS and the RHS pattern of GT-rule $\mathbf{r}$ iff HO-precondition NAC is violated. Consequently, the preserving condition is fulfilled iff the HO-precondition NAC $N^\triangleleft$ is violated.

(iii) From Lemma 1 follows that the dangling edge condition is not fulfilled iff the image of the dangling point graph in the RHS is preserved during reverse application of the GT-rule. The vertices $v \in V_{DP}$ are preserved during the reverse application iff they are in the LHS and RHS pattern of GT-rule $\mathbf{r}$, which is ensured iff there exists a HO-match $\hat{u} = (u_L, u_R)$ with morphism $u_L : DP \to L$ and $u_R : GL^* \to R$.

□

5 Related Work

Static Analysis of Graph Transformation Systems. In the context of graph transformation, the notion of patterns to express global constraints was first proposed in [HW95] and extended in [EEPT06] to the more general framework of high-level replacement systems. Both approaches propose a mathematical construction method for application conditions from constraints founded in the abstract frameworks of category theory. Although they provide the formal foundations, a direct implementation of the approaches is only feasible for transformation tools that are built on top of libraries for categorical constructions, as for example AGG [Tae00]. In contrast, our construction procedure can be implemented using any existing graph transformation tool that considers rule specifications as regular models.

A similar idea, namely to reformulate the constructive approach to a domain supported by standard technologies, is presented in [BBG+06]. The authors propose a translation of patterns and constraints into logic formulas performing the computations symbolically using binary decision diagram based solvers. While this requires a complex translation into the logic domain and an additional solver, our approach uses the GT-engine, which already exists in GT-tools.

Higher-Order Transformation. The majority of higher order transformation approaches do not address static verification. For example, [VP04] proposes a special kind higher order transformation rules (called metarules) to transform generic rewrite rules to model level rules that can be carried out by the transformation engine. In [VP04] it is shown that under suitable assumption the rules for metamodel refractoring can also be used to adapt transformation rules to preserve compatibility with the refactorings.

The approach of [BGL08] uses higher order graph transformation to translate negative constraints into postcondition NACs. Instead of creating the postcondition NACs for all possible gluing graphs, only the maximal gluing graphs are considered. This avoids the construction of postcondition NACs that are subsumed by others. In contrast to our approach, the challenges regarding a practical implementation are not considered. It is also not discussed how to avoid the construction of redundant and invalid postcondition NACs using higher order transformations, which is crucial for using the approach in real world applications.

6 Conclusion

In this paper, we proposed an approach to produce negative application conditions at compile time by applying higher order graph transformation on rule specifications to prevent the runtime occurrence of forbidden patterns. We integrated filtering criteria into the higher order constructs to avoid the unnecessary construction of invalid and redundant negative application conditions and presented formal arguments for the correctness of this technique.

As GT-rules are represented as regular models in many state-of-the-art graph transformation tools, higher order transformations can be carried out easily by the already available transformation machinery of these tools, which can be considered as a clear advantage over a handcrafted implementation.

As the validity of the negative application conditions is checked before their construction, only minor extensions are required to use the approach to directly construct precondition NACs. That

is, simply modifying the presented algorithm such that it creates the same negative application condition but now as precondition NAC, and adding an extra step that applies an additional HO-rule to add (and remove) the elements in the previously created precondition NAC as specified by reverse application of the GT-rule. Although, this is not discussed in the paper (due to space limitations), it is included in our graph transformation tool eMoflon [ALPS11]. Additionally, we will also support positive (if then) application conditions, as well as application condition including attributes and inheritance in an upcoming release.

A further task is to investigate how to use higher order transformation for static analysis of higher order transformation. For instance, the precondition NAC in the filter criterion checking HO-rule can be derived using a higher order negative constraint.

Bibliography

[ALPS11] A. Anjorin, M. Lauder, S. Patzina, A. Schürr. eMoflon: Leveraging EMF and Professional CASE Tools. In *INFORMATIK 2011*. LNI 192, p. 281. Gesellschaft für Informatik, Bonn, October 2011. extended abstract.

[BBG+06] B. Becker, D. Beyer, H. Giese, F. Klein, D. Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *Proceedings of the 28th ICSE*. ICSE '06, pp. 72–81. ACM, New York, NY, USA, 2006.

[BGL08] P. Bottoni, E. Guerra, J. de Lara. Enforced generative patterns for the specification of the syntax and semantics of visual languages. *Journal of Visual Languages and Computing* 19(4):429 – 455, 2008.

[EEHP04] H. Ehrig, K. Ehrig, A. Habel, K.-H. Pennemann. Constraints and Application Conditions: From Graphs to High-Level Structures. In Ehrig et al. (eds.), *Graph Transformations*. LNCS 3256, pp. 287–303. Springer Berlin Heidelberg, 2004.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer Verlag, 2006.

[Hec98] R. Heckel. Compositional Verification of Reactive Systems Specified by Graph Transformation. In *In FASE 1998*. LNCS 1382, pp. 138–153. Springer, 1998.

[HHT96] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae* 26(3/4):287–313, 1996.

[HW95] R. Heckel, A. Wagner. Ensuring Consistency of Conditional Graph Rewriting – A Constructive Approach. In Corradini and Montanari (eds.), *Proc. of Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation*. ENTCS 2, pp. 118–126. Elsevier, Volterra, Pisa, Italy, August 1995.

[KMP02] M. Koch, L. V. Mancini, F. Parisi-Presicce. A Graph-based Formalism for RBAC. *ACM Trans. Inf. Syst. Secur.* 5(3):332–365, Aug. 2002.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation*. Volume 1: Foundations. World Scientific, 1997.

[SCFY96] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman. Role-based access control models. *Computer* 29(2):38 –47, 1996.

[Tae00] G. Taentzer. AGG: A Tool Environment for Algebraic Graph Transformation. In Nagl et al. (eds.), *Proc. of the AGTIVE'99*. LNCS 1779, pp. 481–490. Springer-Verlag, Kerkrade, The Netherlands, 2000.

[VP04] D. Varró, A. Pataricza. Generic and Meta-transformations for Model Transformation Engineering. In Baar et al. (eds.), *UML 2004 - The Unified Modeling Language. Modelling Languages and Applications*. LNCS 3273, pp. 290–304. Springer, 2004.