Attribute Handling for Generating Preconditions from Graph Constraints

Frederik Deckwerth* and Gergely Varró**

Technische Universität Darmstadt,
Real-Time Systems Lab,
D-64283 Merckstraße 25, Darmstadt, Germany
{frederik.deckwerth,gergely.varro}@es.tu-darmstadt.de

Abstract. This paper presents a practical attribute handling approach for generating rule preconditions from graph constraints. The proposed technique and the corresponding correctness proof are based on symbolic graphs, which extend the traditional graph-based structural descriptions by logic formulas used for attribute handling. Additionally, fully declarative rule preconditions are derived from symbolic graphs, which enable automated attribute resolution as an integral part of the overall pattern matching process, which carries out the checking of rule preconditions at runtime in unidirectional model transformations.

Keywords: static analysis, rule preconditions, attribute handling

1 Introduction

Graph transformation (GT) [1] as a declarative technique to specify rule-based manipulation of system models has been successfully employed in many practical, real-world application scenarios [2] including ones from the security domain [3], where the formal nature of graph transformation plays an important role.

A recurring important and challenging task is to statically ensure that (global) negative constraints representing forbidden structures are never allowed to occur in any system models that are derived by applying graph transformation rules.

A well-known general solution to this challenge was described as a sophisticated constructive algorithm [4], which generates negative application conditions (NAC) [5] from the negative constraints, and attaches these new NACs to the left-hand side (LHS) of the graph transformation rules at design time. At runtime, these NAC-enriched left-hand sides block exactly those rule applications that would lead to a constraint violating model.

This constructive algorithm is perfectly appropriate from a theoretical aspect for proving the correctness of the approach when system models are graphs without numeric or textual attributes, and negative constraints and graph transformation rules specify only structural restrictions and manipulations, respectively, but in practical scenarios the handling of attributes cannot be ignored at all.

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

A state-of-the-art approach [6] has been recently presented for transforming arbitrary OCL invariants and rule postconditions into preconditions, which implicitly involves the handling of attributes as well. On one hand, the corresponding report lacks formal arguments underpinning the correctness of the suggested algorithm. On the other hand, the proposed transformation manipulates the abstract syntax tree of OCL expressions, consequently, this solution might be negatively affected by the same (performance) issues like any other OCL-based techniques when checking rule preconditions *at runtime*. The main point is that an OCL expression is always evaluated (i) from a single and fix starting point defined explicitly by its context, and (ii) in an imperative manner following exactly the traversal order specified by the user, which is not necessarily suboptimal, but requires algorithmic background from the modeller.

In this paper, we present a practical and provenly correct attribute handling approach for generating preconditions from graph constraints. The proposed technique and the corresponding correctness proof use symbolic graphs [7], which combine graph-based structural descriptions with logic formulas expressing attribute values and restrictions. Additionally, the concept of fully declarative pattern specifications [8, 9] is reused in a novel context, namely, as an intermediate language, to which the generated symbolic graph preconditions are converted. Finally, an attribute evaluation order is automatically derived from these declarative pattern specifications together with a search plan for the graph constraints resulting in a new, integrated pattern matching process, which performs the checking of rule preconditions in unidirectional model transformations.

The remainder of the paper is structured as follows: Section 2 introduces basic logic, modeling and graph transformation concepts. The precondition NAC derivation process and the corresponding correctness proof are presented in Sec. 3, while Sec. 4 describes the automated attribute resolution technique. Related work is discussed in Sec. 5, and Sec. 6 concludes our paper.

2 Basic Concepts

2.1 Formal Concepts

Signature and Σ-algebra. A *signature* Σ consists of sort and attribute value predicate symbols, and associates a sort symbol with each argument of each attribute value predicate symbol. *A Σ-algebra $\mathcal{D}$* defines the symbols in Σ by assigning (i) a carrier set to each sort symbol, and (ii) a relation to each attribute value predicate symbol. The relation is defined on the carrier sets and has to be compatible with respect to the number and sorts of the attribute value predicate arguments. In this paper, we use a signature and a corresponding Σ-algebra that consists of a single sort *Real* that represents the real numbers $\mathbb{R}$ as well as the attribute value predicates symbols *eq*, *gr*, *mult* and *add*. Symbol *eq* is defined by the equality relation on $\mathbb{R}$, symbol *gr* by $gr(x, y) = \{\, x, y \in \mathbb{R} \mid x > y \,\}$, and symbols *mult* and *add* by $mult(x, y, z) = \{\, x, y, z \in \mathbb{R} \mid x = y \cdot z \,\}$ and $add(x, y, z) = \{\, x, y, z \in \mathbb{R} \mid x = y + z \,\}$, respectively.

First-order logic formula. Given a signature Σ and a set of variables X, a *first-order logic formula* is built from the variables in X, the (attribute value) predicate symbols in Σ, the logic operators $\wedge,\vee,\neg,\Rightarrow,\Leftrightarrow$, the constants $\top,\bot$ (meaning true and false) and the quantifiers $\forall$ and $\exists$ in the usual way [10].

Assignment and evaluation of first-order logic formulas. A *variable assignment* $\sigma : X \to \mathcal{D}$ maps the variables $x \in X$ to a value in the corresponding carrier set of $\mathcal{D}$. A first order logic formula Ψ is evaluated for a given assignment σ in a Σ-algebra $\mathcal{D}$ by first replacing all variables in Ψ according to the assignment σ and evaluating the attribute value predicates according to the algebra and the logic operators in the usual way [10]. We write $\mathcal{D}, \sigma \models \Psi$ iff Ψ evaluates to *true* for the assignment σ; and $\mathcal{D} \models \Psi$, iff Ψ evaluates to *true* for all assignments.

E*-graphs and E*-graph morphisms. An *E*-graph*[1] is a tuple $G = (V_G, E_G, V_G^{\mathbb{L}}, E_G^{\mathbb{L}}, s_G, t_G, s_G^{\mathbb{L}}, t_G^{\mathbb{L}})$ consisting of a set of graph nodes V_G, graph edges E_G, label nodes $V_G^{\mathbb{L}}$, label edges $E_G^{\mathbb{L}}$, and four functions $s_G, t_G, s_G^{\mathbb{L}}, t_G^{\mathbb{L}}$. The functions $s_G : E_G \to V_G$ and $t_G : E_G \to V_G$ assign source and target graph nodes to the graph edges. The functions $s_G^{\mathbb{L}} : E_G^{\mathbb{L}} \to V_G$ and $t_G^{\mathbb{L}} : E_G^{\mathbb{L}} \to V_G^{\mathbb{L}}$ map the label edges to the (source) graph nodes and (target) label nodes, respectively.

An *E*-graph morphism* $h : G \to H$ *from E*-graph* G *to an E*-graph* H is a tuple of total functions $\langle h_V : V_G \to V_H, h_E : E_G \to E_H, h_{V_{\mathbb{L}}} : V_G^{\mathbb{L}} \to V_H^{\mathbb{L}}, h_{E_{\mathbb{L}}} : E_G^{\mathbb{L}} \to E_H^{\mathbb{L}} \rangle$ such that h commutes with source and target functions, i.e., $h_V \circ s_G = s_H \circ h_E$, $h_V \circ t_G = t_H \circ h_E$, $h_V \circ s_G^{\mathbb{L}} = s_H^{\mathbb{L}} \circ h_{E_{\mathbb{L}}}$, $h_{V_{\mathbb{L}}} \circ t_G^{\mathbb{L}} = t_H^{\mathbb{L}} \circ h_{E_{\mathbb{L}}}$. E*-graphs together with their morphisms form the category **E*-graphs**.

Symbolic graphs and symbolic graph morphisms. A *symbolic graph* $G^{\psi} = \langle G, \psi_G \rangle$, which was introduced in [7], consists of an E*-graph part G and a first-order logic formula ψ_G over the Σ-algebra $\mathcal{D}$ using the label nodes in $V_G^{\mathbb{L}}$ as variables and elements of the carrier sets of $\mathcal{D}$ as constants.

A *symbolic graph morphism* $h^{\psi} : \langle G, \psi_G \rangle \to \langle H, \psi_H \rangle$ from symbolic graph $\langle G, \psi_G \rangle$ to $\langle H, \psi_H \rangle$ is an E*-graph morphism $h : G \to H$ such that $\mathcal{D} \models \psi_H \Rightarrow \overline{h}_{\psi}(\psi_G)$, where $\overline{h}_{\psi}(\psi_G)$ is the first-order formula obtained when replacing each variable x in formula ψ_G by $h_{V_{\mathbb{L}}}(x)$. Symbolic graphs over a Σ-algebra $\mathcal{D}$ together with their morphisms form the category $\mathbf{SymbGraphs}_{\mathcal{D}}$.

Pushouts in $\mathbf{SymbGraphs}_{\mathcal{D}}$. (1) is a pushout iff it is a pushout in **E*-graphs** and $\mathcal{D} \models \Psi_3 \Leftrightarrow (\overline{g}_1(\Psi_1) \wedge \overline{g}_2(\Psi_2))$.

$$\begin{array}{ccc} \langle G_0, \Psi_0 \rangle & \xrightarrow{h_1^{\Psi}} & \langle G_1, \Psi_1 \rangle \\ {\scriptstyle h_2^{\Psi}}\downarrow & (1) & \downarrow{\scriptstyle g_1^{\Psi}} \\ \langle G_2, \Psi_2 \rangle & \xrightarrow[g_2^{\Psi}]{} & \langle G_3, \Psi_3 \rangle \end{array}$$

For presentation purposes we consider symbolic graphs G^{ϕ} to have a conjunction $\phi = p_1(x_{1,1}, \ldots x_{1,n}) \wedge \ldots \wedge p_m(x_{m,1}, \ldots, x_{m,k})$ of attribute value predicates $p_1, \ldots, p_m$ as logic formula.

2.2 Modeling and Transformation Concepts

In this section, metamodels, models and patterns are defined as symbolic graphs.

Metamodels and models. A *metamodel* is a symbolic graph $MM^{\phi} = \langle MM, \bot \rangle$, where MM is an E*-graph. The graph nodes $v \in V_{MM}$ and graph edges $e \in E_{MM}$ define *classes* and *associations* in a domain, respectively. The set $V_{MM}^{\mathbb{L}}$ contains one label node for each sort in the given signature. A label edge $e^{\mathbb{L}} \in E_{MM}^{\mathbb{L}}$ from a class $v \in V_{MM}$ to a label node $v^{\mathbb{L}} \in V_{MM}^{\mathbb{L}}$ expresses that class v has an *attribute* $e^{\mathbb{L}}$ of sort $v^{\mathbb{L}}$.

[1] In contrast to E-Graphs [1], E*-Graphs do not provide labels for graph edges.

A *symbolic graph* G^ϕ *conforms to a metamodel* MM^ϕ if all graph nodes V_G and graph edges E_G can be mapped to the classes and associations in the metamodel, and the label edges $E_G^{\mathbb{L}}$ and nodes $V_G^{\mathbb{L}}$ can be mapped to the attributes of corresponding sorts by a symbolic graph morphism $type^\phi : G^\phi \to MM^\phi$.

A model M^ϕ *of a metamodel* MM^ϕ is a symbolic graph $M^\phi = \langle M, \phi_M \rangle$ conforming to metamodel MM^ϕ, which has to fulfill the following properties: (i) A model M^ϕ has a label node $x_{val} \in V_M^{\mathbb{L}}$ for each value val in the carrier sets of $\mathcal{D}$. (ii) For each label node x_{val}, the conjunction ϕ_M includes an equality attribute value predicate $eq(x_{val}, val)$ (i.e., $\phi_M = \bigwedge_{val \in \mathcal{D}} eq(x_{val}, val)$). A *model is valid*[2] if each graph node $v_M \in V_M$ has exactly one label edge $e_M^{\mathbb{L}} \in E_M^{\mathbb{L}}$ for each attribute $e_{MM}^{\mathbb{L}} \in E_{MM}^{\mathbb{L}}$ such that $s(e_{MM}^{\mathbb{L}}) = type_V^\phi(v_M)$, $s(e_M^{\mathbb{L}}) = v_M$ and $type_{E_{\mathbb{L}}}^\phi(e_M^{\mathbb{L}}) = e_{MM}^{\mathbb{L}}$.

Graph nodes, graph edges, label nodes and label edges in a model are called *objects*, *links*, *attribute values* and *attribute slots*, respectively.

A *typed symbolic (graph) morphism* $f^\phi : M_1^\phi \to M_2^\phi$ *from model* M_1^ϕ *to* M_2^ϕ, *both conform to metamodel* MM^ϕ, is a symbolic graph morphism that preserves type information, i.e., $type_1^\phi \circ f^\phi = type_2^\phi$.

Example. Figure 1a shows the e-commerce platform metamodel, which consists of the classes Customer, Order, Article and PaymentMethod. A customer has a set of orders (orders) and registered payment methods (paymentMethods) assigned. An order consists of articles and a payment method represented by the associations articles and usedPaymentMethod, respectively. Attributes and their corresponding sorts are represented using the UML class diagram notation. E.g., the class Customer has an attribute reputation of sort `double`. Additionally, an order has the totalCost attribute that corresponds to the accumulated price of all articles in the articles association. The attribute limit assigns the maximal amount of money admissible in a single transaction to a payment method.

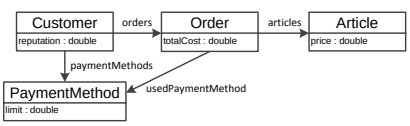
Patterns, negative constraints and model consistency. A *pattern* P^ϕ is a symbolic graph $P^\phi = \langle P, \phi_P \rangle$ that conforms to a metamodel MM^ϕ. Additionally a pattern has no duplicate attributes, i.e., each graph node $v_P \in V_P$ has at most one label edge $e_P^{\mathbb{L}} \in E_P^{\mathbb{L}}$ for each attribute $e_{MM}^{\mathbb{L}} \in E_{MM}^{\mathbb{L}}$ such that $s(e_{MM}^{\mathbb{L}}) = type_V^\phi(v_P)$, $s(e_P^{\mathbb{L}}) = v_P$ and $type_{E_{\mathbb{L}}}^\phi(e_P^{\mathbb{L}}) = e_{MM}^{\mathbb{L}}$.

A *pattern* P^ϕ *matches a model* M^ϕ if there exists a typed symbolic morphism $m^\phi : \langle P, \phi_P \rangle \to \langle M, \phi_M \rangle$ such that functions $m_V : V_P \to V_M, m_E : E_P \to E_M$ and $m_{E_{\mathbb{L}}} : E_P^{\mathbb{L}} \to E_M^{\mathbb{L}}$ are injective and $\mathcal{D} \models \phi_M \Rightarrow \overline{m}(\phi_P)$. The morphism m^ϕ is called *match*. All such morphisms, denoted as $\mathcal{M}'_\phi$, are called match morphisms.

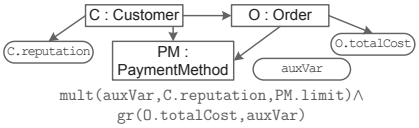
A *negative constraint* NC^ϕ is a pattern to declaratively define forbidden subgraphs in a model. A model M^ϕ is *consistent* with respect to a negative constraint $NC^\phi = \langle NC, \phi_{NC} \rangle$, if there does not exist a match $m^\phi : NC^\phi \to M^\phi$.

Example. Figure 1b shows a global negative constraint limitOrder (NC_{lo}^ϕ) that prohibits a customer (C) to have an order (O) whose totalCost exceeds the

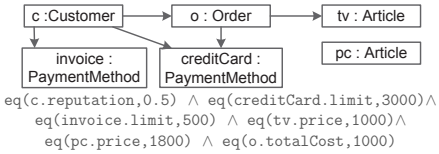
[2] Note that this requirement is only necessary to align the concept of models including attributes with the behaviour of our Eclipse Modeling Framework (EMF) based implementation (Sec. 4). The results of Sec. 3 are not affected by this assumption.



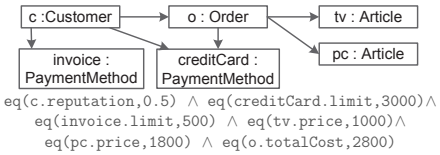
(a) Running example metamodel



(b) Negative constraint limitOrder (NC^{ϕ}_{lo})



(c) Consistent model M^{ϕ}_1



(d) Inconsistent model M^{ϕ}_2

Fig. 1: The e-commerce scenario

product of its reputation and the limit of the used payment method PM. Figures 1c and 1d show the models M^{ϕ}_1 and M^{ϕ}_2, respectively, where label nodes are not explicitly drawn. The model M^{ϕ}_1 is consistent w.r.t. constraint limitOrder. Model M^{ϕ}_2 is inconsistent w.r.t. constraint limitOrder, since the cost (`o.totalCost`) of the order are greater than the product of the payment method limit (`creditCard.limit`) and the customer reputation (`c.reputation`). More specifically, we can find a match $m : NC_{lo} \to M_2$ for the graph part of the constraint NC_{lo} in the model M_2 such that $\mathcal{D} \models \phi_{M_2} \Rightarrow \overline{m}(\phi_{NC_{lo}})$ holds for the formula $\phi_{NC_{lo}}$ of the constraint NC_{lo} after label replacement $\overline{m}(\phi_{NC_{lo}})$.

Symbolic graph transformation. A *graph transformation rule* $\mathbf{r}^{\phi} = \langle L \xleftarrow{l} K \xrightarrow{r} R, \phi \rangle$ consists of a left hand side (LHS) pattern $\langle L, \phi \rangle$, a gluing pattern $\langle K, \phi \rangle$ and a right hand side (RHS) pattern $\langle R, \phi \rangle$ that share the same logic formula ϕ. Morphisms l^{ϕ}, r^{ϕ} are typed symbolic morphisms that are (i) injective for graph nodes and all kinds of edges, (ii) bijective for label nodes, and (iii) $\mathcal{D} \models \phi \Leftrightarrow \overline{l}(\phi) \Leftrightarrow \overline{r}(\phi)$. These morphisms are denoted by $\mathcal{M}_{\phi}$.

The LHS and RHS of graph transformation rule $\mathbf{r}^{\phi}$ can be augmented with *negative application conditions* (NACs) $n^{\phi}_L : L^{\phi} \to N^{\phi}_L \in NAC_L$ (precondition NAC) and $n^{\phi}_R : R^{\phi} \to N^{\phi}_R \in NAC_R$ (postcondition NAC), where n^{ϕ}_L and n^{ϕ}_R are match morphisms.

A rule $\mathbf{r}^{\phi} = \langle L \xleftarrow{l} K \xrightarrow{r} R, \phi \rangle$ with negative precondition NACs NAC_L is *applicable* to a model M^{ϕ} iff (i) there exists a match $m^{\phi} : \langle L, \phi \rangle \to \langle M, \phi_M \rangle$ of the LHS $\langle L, \phi \rangle$ in $\langle M, \phi_M \rangle$, and (ii) the precondition NACs in NAC_L are satisfied by the current match m^{ϕ}. A precondition NAC $n^{\phi}_L : L^{\phi} \to N^{\phi}_L \in NAC_L$ is satisfied by a match m^{ϕ} if there does not exist a match $x^{\phi}_L : \langle N_L, \phi_{N_L} \rangle \to \langle M_L, \phi_{M_L} \rangle$ of the precondition NAC in the model such that $m_L = x_L \circ n_L$.

The application of a graph transformation rule $\mathbf{r}^\phi$ to a model M_L^ϕ resulting in model M_R^ϕ is given by the double pushout diagram, where m_L^ϕ (match), m_K^ϕ and m_R^ϕ (co-match) are match morphisms.

$$\begin{array}{ccccc} L^\phi & \xleftarrow{l^\phi} & K^\phi & \xrightarrow{r^\phi} & R^\phi \\ {\scriptstyle m_L^\phi}\downarrow & & {\scriptstyle m_K^\phi}\downarrow & & {\scriptstyle m_R^\phi}\downarrow \\ M_L^\phi & \xleftarrow[l^\phi_{M_L}]{} & M_K^\phi & \xrightarrow[r^\phi_{M_R}]{} & M_R^\phi \end{array}$$

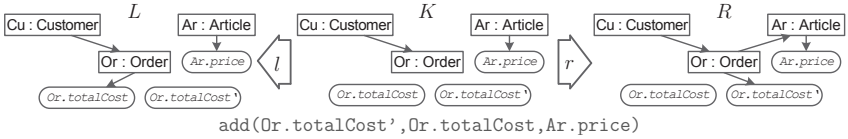


Fig. 2: Graph transformation rule addArticle

Although it seems counterintuitive at a first glance that we require L^ϕ, K^ϕ and R^ϕ to share the same conjunction and label nodes, it does not mean that attribute values cannot be changed by a rule application, since attribute values can be modified by redirecting label edges.

To preserve model validity by a graph transformation rule application we introduce conditions that ensure that rules do not transform valid models into invalid ones.

Model validity preserving graph transformation rules. A graph transformation rule $\mathbf{r}^\phi = \langle L \xleftarrow{l} K \xrightarrow{r} R, \phi \rangle$ typed over metamodel MM^ϕ is *model validity preserving* if: (i) *For each created object all attribute values are initialized.* Formally, for each created graph node $v \in V_R \backslash r_V(V_K)$ there exists exactly one label edge $e^{\mathbb{L}} \in E_R^{\mathbb{L}}$ for each corresponding attribute $e_{MM}^{\mathbb{L}} \in MM^\phi$: $s_{MM}^{\mathbb{L}}(e_{MM}^{\mathbb{L}}) = type_G(v)$ s.t. $type_{\mathbb{L}}(e^{\mathbb{L}}) = e_{MM}^{\mathbb{L}}$ assigning a value to the attribute, i.e., $s_R^{\mathbb{L}}(e^{\mathbb{L}}) = v$. (ii) *For preserved objects, rules can only change attribute values by redirecting label edges.* Formally, for each label edge $e_1^{\mathbb{L}} \in E_L^{\mathbb{L}}$ in the LHS pattern whose source graph node is preserved by the rule application (i.e. $\exists v \in V_K$ s.t. $s_L^{\mathbb{L}}(e_1^{\mathbb{L}}) = l_V(v)$), there exists exactly one label edge $e_2^{\mathbb{L}} \in E_R^{\mathbb{L}}$ of the same type (i.e., $type_{\mathbb{L}}(e_1^{\mathbb{L}}) = type_{\mathbb{L}}(e_2^{\mathbb{L}})$) in the RHS pattern such that $s_R^{\mathbb{L}}(e_2^{\mathbb{L}}) = r_V(v)$. Similarly, for each label edge in the RHS pattern with preserved source graph node, there exists exactly one label edge with similar source and same type in the LHS pattern. Note that for object deletion model validity is preserved by the dangling edge condition for the double pushout approach [1].

Example. Figure 2 shows the rule addArticle that adds an article Ar to the order Or of a customer Cu, and calculates the new total cost `Or.totalCost'` of order Or by adding the price `Ar.price` of the added article Ar to the actual total cost `Or.totalCost` of the order Or. The total cost value is updated by redirecting the label edge from the actual value `Or.totalCost` to the new value `Or.totalCost'`. Morphisms are implicitly specified in all the figures of the running example by matching node identifier. The result of applying the rule to user `u`, order `o`, and article `pc` in the model of Figure 1c is depicted in Figure 1d.

Consistency guaranteeing rules. A rule $\mathbf{r}^\phi$ with a set of precondition NACs NAC_L is *consistency guaranteeing* w.r.t a negative constraint NC^ϕ, iff for any arbitrary model M_L^ϕ and all possible applications of rule $\mathbf{r}^\phi$ that result in model M_R^ϕ it holds that M_R^ϕ is consistent w.r.t. the negative constraint NC^ϕ.

3 Constructing Precondition NACs with Attributes

In this section, we extend the results of constructing precondition NACs from negative constraints presented in [1] to symbolic graph transformation. The construction of precondition NACs are carried out by (i) constructing a postcondition NAC from the negative constraint and the RHS pattern of a GT-rule (Sec. 3.1) and (ii) back-propagating the postcondition NAC into an equivalent precondition NAC (Sec. 3.2). In Section 3.3 we show that the construction ensures consistency guarantee.

3.1 Construction of Postcondition NACs from Negative Constraints

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 graph parts of the negative constraint and the RHS together along the common subgraph. The logic part is obtained as the conjunction of the formulas of the RHS pattern and the negative constraint, where the label nodes that are glued along the common subgraph are replaced in both formulas with a common label.

Formally, the postcondition NACs $n_R^\phi : \langle R, \phi_R\rangle \to \langle N_R, \phi_{N_R}\rangle \in NAC_R$ for the RHS pattern $\langle R, \phi_R\rangle$ of a rule and a negative constraint $\langle NC, \phi_{NC}\rangle$ is derived as the gluings $\langle R, \phi_R\rangle \overset{n_R^\phi}{\to} \langle N_R, \phi_{N_R}\rangle \overset{q^\phi}{\leftarrow} \langle NC, \phi_{NC}\rangle$ such that the pair of match morphisms (n_R^ϕ, q^ϕ) is jointly epimorphic and $\mathcal{D} \models \phi_{N_R} \Leftrightarrow (\overline{n}_R(\phi_R) \wedge \overline{q}(\phi_{NC}))$.

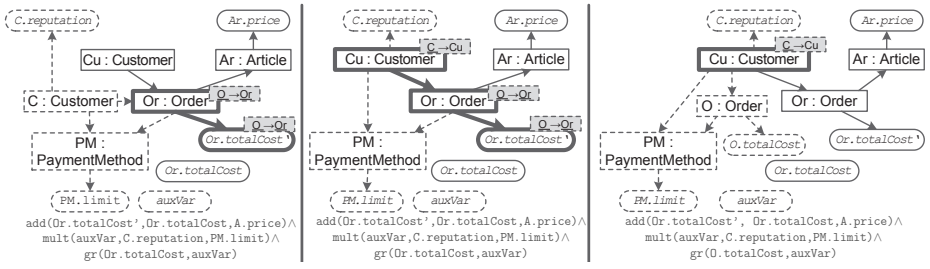


Fig. 3: Postcondition NACs derived for rule addArticle and neg. constr. limitOrder

Example. Figure 3 depicts all postcondition NACs derived from the rule addArticle (Fig. 2) and the negative constraint limitOrder (Fig. 1b). Solid nodes and edges belong to the RHS of rule addArticle. Dashed elements are from the negative constraint limitOrder, and the common subgraph is drawn bold. The

mapping of the RHS pattern of rule addArticle and the constraint limitOrder are implicitly denoted by the mapping of the node identifiers. For the common subgraph, we used the labels from the RHS of rule addArticle and denoted the mapping from the constraint by the grey boxes. E.g., O→Or denotes that node O of the constraint is mapped to node Or in the postcondition NAC.

3.2 Constructing Precondition NAC from Postcondition NAC

Each postcondition NAC constructed in the previous step is back-propagated to the LHS as a precondition NAC by reverting the modifications of the graph part specified by the symbolic GT-rule while preserving the logic formula.

Formally, for a GT-rule $\mathbf{r}^\phi = \langle L \xleftarrow{l} K \xrightarrow{r} R, \phi \rangle$ with a postcondition NAC $n_R^\phi : \langle R, \phi_R \rangle \to \langle N_R, \phi_{N_R} \rangle$, the precondition NAC $n_L^\phi : \langle L, \phi_L \rangle \to \langle N_L, \phi_{N_L} \rangle$ is derived as follows: (i) Construct $n_K : K \to N_K$ by the pushout complement of the pair (r, n_R) in **E*-graphs**. (ii) If (r, n_R) has a pushout complement then $n_L : L \to N_L$ is constructed by the pushout of l and n_K in **E*-graphs**. (iii) The precondition NAC is then defined by $n_L^\phi : \langle L, \phi_L \rangle \to \langle N_L, \phi_{N_L} \rangle$ where ϕ_{N_L} is the same formula as ϕ_{N_R}.

$$\begin{array}{ccccc} L^\phi & \xleftarrow{l^\phi} & K^\phi & \xrightarrow{r^\phi} & R^\phi \\ {\scriptstyle n_L^\phi}\downarrow & & {\scriptstyle n_K^\phi}\downarrow & & \downarrow{\scriptstyle n_R^\phi} \\ N_L^\phi & \xleftarrow[l_{N_L}^\phi]{} & N_K^\phi & \xrightarrow[r_{N_R}^\phi]{} & N_R^\phi \end{array}$$

Note that the label nodes and the logic formula remains invariant after symbolic transformation [11] (i.e., $V_{N_L}^{\mathbb{L}} = V_{N_K}^{\mathbb{L}} = V_{N_R}^{\mathbb{L}}$ and $\mathcal{D} \models \phi_{N_L} \Leftrightarrow \phi_{N_K} \Leftrightarrow \phi_{N_R}$).

Example. Figure 4 shows the construction of the precondition NAC from the postcondition NAC depicted in the middle of Fig. 3. The precondition NAC prevents the rule addArticle to add an article to an order if the new total cost `Or.totalCost'` exceeds the product of the used payment method limit `PM.limit` and the reputation `C.reputation` of the customer. Note that label node identifier can be chosen arbitrarily, hence label node `C.reputation` refers to the reputation attribute of customer Cu.

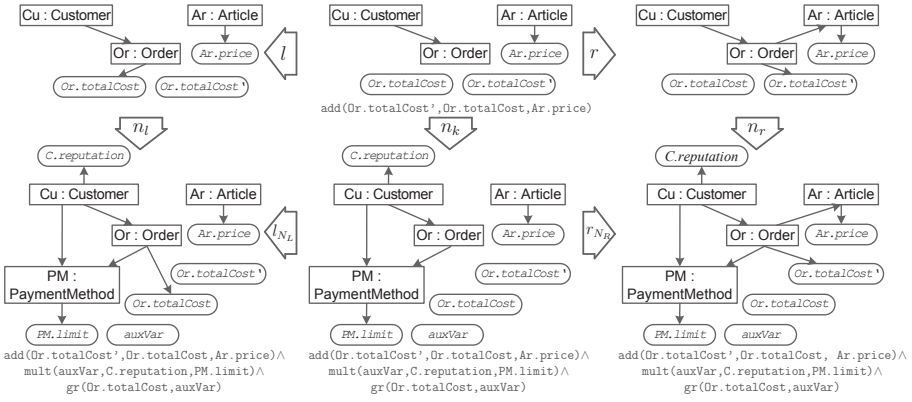


Fig. 4: Constructing a precondition NAC from a postcondition NAC

3.3 Proving the Correctness of the Construction Technique

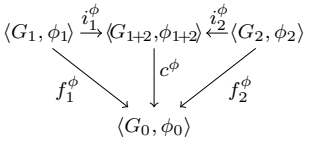
In order to reuse the results from [1] to show that the presented construction is indeed sufficient and necessary to ensure consistency guarantee we have to prove the following properties for symbolic graphs:

1. **SymbGraphs$_\mathcal{D}$** has a generalized disjoint union (binary coproducts).
2. **SymbGraphs$_\mathcal{D}$** has a generalized factorization in surjective and injective parts for each symbolic graph morphism (weak $\mathcal{E}_\phi$-$\mathcal{M}'_\phi$ factorization).
3. Match morphisms $\mathcal{M}'_\phi$ are closed under composition and decomposition.
4. $\mathcal{M}'_\phi$ is closed under pushouts (PO) and pullbacks (PB) along $\mathcal{M}_\phi$-morphisms

Note that although we used typed graphs (i.e. graphs conform to a metamodel) in our running example and formalization we only provide proofs for untyped symbolic graphs as the proofs can be easily extended, since symbolic graphs are an adhesive HLR category [7] and consequently typed symbolic graphs form an adhesive HLR category (slice construction [1]).

Property 1 (SymbGraphs$_\mathcal{D}$ has binary coproducts.) *The diagram on the next page is a binary coproduct in **SymbGraphs$_\mathcal{D}$** if and only if it is a binary coproduct in **E*-graphs** and $D \models \phi_{1+2} \Leftrightarrow (\bar{i}_1(\phi_1) \wedge \bar{i}_2(\phi_2))$.*

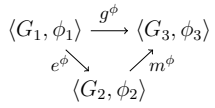
Proof. In **E*-graphs** the coproduct is constructed componentwise as the disjoint union. Consequently, given symbolic graph morphisms f_1^ϕ and f_2^ϕ there exists E*-graph morphisms i_1, i_2, and c such that the diagram below commutes.



The morphisms i_1^ϕ and i_2^ϕ are morphisms in **SymbGraphs$_\mathcal{D}$** since $\mathcal{D} \models (\bar{i}_1(\phi_1) \wedge \bar{i}_2(\phi_2)) \Rightarrow \bar{i}_1(\phi_1)$ and $\mathcal{D} \models (\bar{i}_1(\phi_1) \wedge \bar{i}_2(\phi_2)) \Rightarrow \bar{i}_2(\phi_2)$. Also $c^\phi : \langle G_{1+2}, \bar{i}_1(\phi_1) \wedge \bar{i}_2(\phi_2)\rangle \to \langle G_0, \phi_0\rangle$ is a morphism in **SymbGraphs$_\mathcal{D}$**, as, by definition, $\mathcal{D} \models \phi_0 \Rightarrow \overline{f}_1(\phi_1)$ and $\mathcal{D} \models \phi_0 \Rightarrow \overline{f}_2(\phi_2)$, $f_1 = c \circ i_1$, and $f_2 = c \circ i_2$, so $\mathcal{D} \models \phi_0 \Rightarrow \overline{c}(\bar{i}_1(\phi_1)) \wedge \overline{c}(\bar{i}_2(\phi_2))$ that implies $\mathcal{D} \models \phi_0 \Rightarrow \overline{c}(\bar{i}_1(\phi_1) \wedge \bar{i}_2(\phi_2))$.

Property 2 (SymbGraphs$_\mathcal{D}$ has weak $\mathcal{E}_\phi$-$\mathcal{M}'_\phi$ factorization.) *Given the symbolic morphisms $g^\phi : \langle G_0, \phi_0\rangle \to \langle G_2, \phi_2\rangle$, $e^\phi : \langle G_0, \phi_0\rangle \to \langle G_1, \phi_1\rangle$, and $m^\phi : \langle G_1, \phi_1\rangle \to \langle G_2, \phi_2\rangle$ with $m \circ e = f$, where e is an epimorphism (i.e., surjective on all kinds of nodes and edges) and m of class $\mathcal{M}'$ of E*-graph morphisms, which are injective for graph nodes and all kind of edges. The symbolic morphisms e^ϕ and m^ϕ are the $\mathcal{E}_\phi$-$\mathcal{M}'_\phi$ factorization of g^ϕ if e and m are an epi-$\mathcal{M}'$ factorization of g in* E*-graphs *and $\mathcal{D} \models \phi_2 \Leftrightarrow \overline{e}(\phi_1)$.*

Proof. The category **E*-graphs** has weak epi-$\mathcal{M}'$ factorization [1]. Consequently, given symbolic graph morphism g^ϕ there exists an epimorphism e and morphism $m \in \mathcal{M}'$ in **E*-graphs** such that $g = m \circ e$. Obviously, morphism $e^\phi : \langle G_1, \phi_1\rangle \to \langle G_2, \phi_2\rangle$ is in **SymbGraphs$_\mathcal{D}$** since, by definition, $\mathcal{D} \models \phi_2 \Leftrightarrow \overline{e}(\phi_1)$ implies $\mathcal{D} \models \phi_2 \Rightarrow \overline{e}(\phi_1)$. Morphism $m^\phi : \langle G_1, \phi_1\rangle \to \langle G_2, \phi_2\rangle$ is in **SymbGraphs$_\mathcal{D}$** since $\mathcal{D} \models \phi_2 \Rightarrow \overline{g}(\phi_1)$ and $g = m \circ e$, so $\mathcal{D} \models \phi_3 \Rightarrow \overline{m}(\overline{e}(\phi_1))$.



Property 3 ($\mathcal{M}'_\phi$ is closed under composition.) *If (i) $f^\phi : A^\phi \to B^\phi$ and $g^\phi : B^\phi \to C^\phi$ in $\mathcal{M}'_\phi$ then $g^\phi \circ f^\phi$ is in $\mathcal{M}'_\phi$, and if (ii) $g^\phi \circ f^\phi$ and g^ϕ are in $\mathcal{M}'_\phi$ then f^ϕ is in $\mathcal{M}'_\phi$.*

Proof. The property holds for **E*-graph** morphisms in $\mathcal{M}'$ that are injective for graph nodes and all kinds of edges [1]. Consequently, we have $f : A \to B \in \mathcal{M}'$, $g : B \to C \in \mathcal{M}'$ and $g \circ f \in \mathcal{M}'$. (i) Morphism $g^\phi \circ f^\phi \in \mathcal{M}'_\phi$, since $\mathcal{D} \models \phi_C \Rightarrow \overline{g}(\phi_B)$ and $\mathcal{D} \models \phi_B \Rightarrow \overline{f}(\phi_A)$ implies $\mathcal{D} \models \phi_C \Rightarrow \overline{g}(\overline{f}(\phi_A))$. (ii) Morphism $f^\phi \in \mathcal{M}'_\phi$, as $\mathcal{D} \models \phi_C \Rightarrow \overline{g}(\overline{f}(\phi_A))$ and $\mathcal{D} \models \phi_C \Rightarrow \overline{g}(\phi_B)$ implies $\mathcal{D} \models \phi_B \Rightarrow \overline{f}(\phi_A)$.

Property 4 ($\mathcal{M}'_\phi$ is closed under POs and PBs along $\mathcal{M}_\phi$-morphisms) *$\mathcal{M}'_\phi$ is closed under pushouts and pullbacks along $\mathcal{M}_\phi$ morphisms if the pushout or pullback (1) with $h_1^\phi \in \mathcal{M}_\phi$, $h_2^\phi \in \mathcal{M}'_\phi$ or $g_2^\phi \in \mathcal{M}_\phi$, $g_1^\phi \in \mathcal{M}'_\phi$, respectively, then we also have $g_1^\phi \in \mathcal{M}'_\phi$ or $h_2^\phi \in \mathcal{M}'_\phi$ [1].*

Proof. In **E*-graphs** pushouts and pullbacks can be constructed componentwise [1]. Consequently the property holds for both: (i) choosing $\mathcal{M}$ as the class of morphisms injective for graph nodes and all kinds of edges and bijective for label nodes, and (ii) choosing $\mathcal{M}$ similar to $\mathcal{M}'$ (the class of morphisms injective for graph nodes and all kinds of edges). Since in $\mathbf{SymbGraphs}_\mathcal{D}$ pushouts and pullbacks exist along $\mathcal{M}_\phi$ and $\mathcal{M}'_\phi$ morphisms [11], $\mathcal{M}'_\phi$ is closed under pushouts and pullbacks along $\mathcal{M}_\phi$-morphisms (and $\mathcal{M}'_\phi$-morphisms).

$$\begin{array}{ccc} \langle G_0, \phi_0 \rangle & \xrightarrow{h_1^\phi} & \langle G_1, \phi_1 \rangle \\ {\scriptstyle h_2^\phi}\downarrow & (1) & \downarrow{\scriptstyle g_1^\phi} \\ \langle G_2, \phi_2 \rangle & \xrightarrow[g_2^\phi]{} & \langle G_3, \phi_3 \rangle \end{array}$$

After proving these properties for symbolic graphs, we can now apply results from [1] to show that the given construction ensures consistency guarantee.

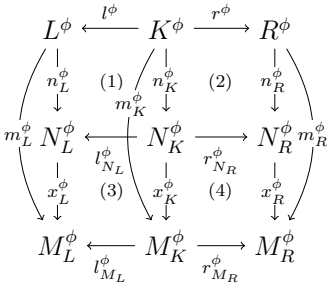
Theorem 1 (Constructing NACs from negative constraints). *Given a symbolic graph transformation rule $\mathbf{r}^\phi = \langle L \xleftarrow{l} K \xrightarrow{r} R, \phi \rangle$ and the set of postcondition NACs NAC_R constructed from the rule $\mathbf{r}^\phi$ and the negative constraint NC^ϕ as defined in Section 3.1. The application of rule $\mathbf{r}^\phi$ satisfies the postcondition NAC iff model M_R^ϕ is consistent w.r.t. the negative constraint NC^ϕ.*

Proof. The proof follows from Theorem 7.13 in [1], and the properties 1–4.

Theorem 2 (Equivalence of the constructed precondition and postcondition NACs). *For each postcondition NAC n_R^ϕ over symbolic GT-rule $\mathbf{r}^\phi$, the precondition NAC n_L^ϕ constructed according to Section 3.2 is satisfied for each application of $\mathbf{r}^\phi$ iff the postcondition NAC n_R^ϕ is satisfied.*

We only provide a proof for the logic component, as the detailed proof of the construction for the category of **E*-graphs** can be found in [1].

Proof. Let the diagram below show the construction of the precondition NAC $n_L^\phi : \langle L, \phi_L \rangle \to \langle N_L, \phi_{N_L} \rangle$ from the postcondition NAC $n_R^\phi : \langle R, \phi_R \rangle \to \langle N_R, \phi_{N_R} \rangle$ for rule $\mathbf{r}^\phi = \langle L \xleftarrow{l} K \xrightarrow{r} R, \phi \rangle$ according to Section 3.2. Assuming the construction is valid for **E*-graphs** (using the $\mathcal{M}$–$\mathcal{M}'$ PO–PB decomposition property [1]) we know that there exists an E*-graph morphism $x_L : N_L \to M_L \in \mathcal{M}'$ iff there exists morphism $x_R : N_R \to M_R \in \mathcal{M}'$ such that $x_R \circ n_R = m_R$ and $x_L \circ n_L = m_L$, and (1), (2), (3), (4) commute.



As the set of label nodes and the logic formula remains invariant after symbolic transformation [7] (i.e., $V^{\mathbb{L}}_{M_L} = V^{\mathbb{L}}_{M_R}$ up to isomorphism and $\mathcal{D} \models \phi_{M_L} \Leftrightarrow \phi_{M_R}$) we may consider $\overline{m}_L = \overline{m}_R$, and ϕ_{M_L} and ϕ_{M_R} to be the same formula abbreviated as ϕ''. Consequently we have to show that if there exists E*-graph morphisms x_R and x_L then $\mathcal{D} \models \phi'' \Rightarrow \overline{x}_L(\phi_{N_L})$ iff $\mathcal{D} \models \phi'' \Rightarrow \overline{x}_R(\phi_{N_R})$. This trivially holds, since the set of label nodes and the logic formulas in the NACs N_L^ϕ and N_R^ϕ are also similar by construction (Sec. 3.2). Hence, we may consider $\overline{n}_L = \overline{n}_R$, and ϕ_{N_L} and ϕ_{N_R} to be the same formula, which implies that $\overline{x}_L = \overline{x}_R$.

4 Attributes in Search Plan Driven Pattern Matching

As demonstrated in Section 3, rule preconditions can be produced as symbolic graphs, whose graph part and logic formula describe structural and attribute restrictions, respectively. This section presents how a generated rule precondition can be actually checked by a tool in a practical setup as a pattern matching process. This paper extends the pattern matching approach for EMF models of [12] by attribute handling. The new process can be summarized as follows:

Section 4.1 A (declarative) pattern specification is derived from the symbolic graph representing the rule precondition. In this phase, the concept of declarative pattern specifications originates from [8], and the idea to describe attribute restrictions by predicates has been first proposed in [9], however, the *complete derivation process* is a novel contribution of this paper.

Section 4.2 Operations representing atomic steps in the pattern matching process are created from the pattern specification. In this phase, the concept to use operations in pattern matching for structural restrictions originates from [8, 12], while the ideas of *attribute manipulating operations* and *their intertwinement with structure checking operations*, which results in a uniform process for both kinds of operations, are new contributions.

Section 4.3 The operations are filtered and sorted by a search plan generation algorithm [12] to prepare a valid (and efficient) search plan, which is then used, e.g., by a code generator to produce executable code for pattern matching as described in [13].

Due to space restrictions, the current paper only presents the new contributions of Sec. 4.1 and 4.2 in details. The techniques of Sec. 4.3, which have been

described in other papers, are applicable for attributes without any change, consequently, this phase is only demonstrated on the running example.

4.1 Pattern Specification

Definitions in this subsection are from [8, 12]. A *pattern specification* is a set of predicates over a set of variables as arguments. A *variable* is a placeholder for an object or an attribute value in a model. A *predicate* specifies a condition on a set of variables (which are also referred to as *arguments* in this context) that must be fulfilled by the model elements assigned to the arguments.

Four kinds of predicates are used in our approach. An *association predicate* refers to an association in the metamodel and prescribes the existence of a link, which conforms to the referenced association, and connects the source and the target object assigned to the first and second argument, respectively. An *attribute predicate*, whose concept stems from [9], refers to an attribute in the metamodel and ensures that the object assigned to the first argument has an attribute slot with the attribute value assigned to the second argument. An *attribute value predicate* places a restriction on attribute values as already discussed in Sec. 2.1. A *NAC predicate* refers to a NAC and ensures that the NAC is satisfied.

Deriving a pattern specification from a pattern. A pattern specification is derived from a given pattern by the following *new* algorithm:

1. For each graph and label node in the pattern, a variable is introduced.
2. For each graph edge, an association predicate referring to the type of the graph edge is added to the pattern specification. The two arguments are the variables for the source and target graph nodes of the processed graph edge.
3. For each label edge, an attribute predicate of corresponding type is added to the pattern specification. The two arguments are the source graph node and the target label node of the processed label edge, respectively.
4. Each attribute value predicate conjuncted in the logic formula of the pattern is added to the pattern specification.
5. For each precondition NAC in the pattern, a NAC predicate is added to the pattern specification that has an argument for each node in the pattern.

Example. The pattern specification derived from the LHS pattern of rule addArticle (Fig. 4) consists of (i) the association predicate `orders(Cu,Or)` requiring an orders link between customer Cu and order Or, (ii) the attribute predicates `totalCost(Or,Or.totalCost)` and `price(Ar,Ar.price)` for the totalCost and price attributes of order Or and article Ar, respectively, (iii) the attribute value predicate `add(Or.totalCost',Or.totalCost,Ar.price)` (appearing in the logic formula of the LHS pattern), and (iv) the NAC predicate `addArticle-NAC(Cu,Or,Ar,Ar.price,Or.totalCost,Or.totalCost')`.

4.2 Creating Operations

This subsection describes the process of creating operations from the predicates of the pattern specification. The definitions and the production of operations for association predicates are from [8, 12], while the attribute and NAC handling

operations are *novel contributions*. It should be highly emphasized that *the new process does not distinguish between the handling of attribute and structural restrictions any more. Consequently, all these operations are intertwined to an integrated pattern matching process.*

Definitions and operations for association predicates. Let us assume that an (arbitrary) order is fixed for the variables in the pattern specification. An *adornment* represents binding information for *all variables* in the pattern specification by a corresponding character sequence consisting of letters `B` or `F`, which indicate that the variable in that position is *bound* or *free*, respectively. An *operation* represents an atomic step in the pattern matching process. It consists of a predicate, and an operation adornment. An *operation adornment* prescribes which arguments must be bound when the operation is executed.

For each association predicate, two operations are created with the corresponding adornments `BB` and `BF`. The operation adorned with `BB` verifies the existence of a link of corresponding type between the objects bound to the arguments. The operation with the `BF` adornment denotes a forward navigation.

Operations for attribute, attribute value and NAC predicates. For each attribute predicate, two operations are created with the corresponding adornments `BB` and `BF`. The operation adorned with `BB` checks that the (attribute) value of the corresponding attribute of the first argument is equal to the value of the second argument. The operation with adornment `BF` looks up the (attribute) value of the corresponding attribute of the first argument, and assigns this value to the second argument.

For each attribute value predicate, a set of used-defined operations is created. E.g., a user may define four operations for the attribute value predicate $\texttt{add}(\texttt{x}_1, \texttt{x}_2, \texttt{x}_3)$. The operation adorned with `BBB` checks whether the value of variable x_1 equals to the sum of the values of x_2 and x_3. The operation with `FBB` adornment assigns the sum of the values of x_2 and x_3 to variable x_1, while the operations adorned with `BFB` and `BBF` calculate the difference of the first and the other bound argument, and assign this difference to the free argument.

For each NAC predicate, an operation with only bound arguments is created that checks whether the corresponding NAC is satisfied.

Predicate	Op. Adornm.	Predicate	Op. Adornm.
Operations for association predicates		**Operations for attribute value predicates**	
orders(Cu,Or)	BB	add(Or.totalCost',Or.totalCost,Ar.price)	BBB
orders(Cu,Or)	BF	add(Or.totalCost',Or.totalCost,Ar.price)	FBB
Operations for attribute predicates		add(Or.totalCost',Or.totalCost,Ar.price)	BFB
totalcost(Or,Or.totalCost)	BB	add(Or.totalCost',Or.totalCost,Ar.price)	BBF
totalcost(Or,Or.totalCost)	BF		
price(Ar,Ar.price)	BB	**Operations for NAC predicates**	
price(Ar,Ar.price)	BF	addArticleNAC(Cu,Or,Ar,Ar.price,Or.totalCost,Or.totalCost')	BBBBBB

Fig. 5: Created operations for the LHS pattern of the addArticle rule

Example. Fig. 5 lists the operations derived from the LHS of rule addArticle.

4.3 Search Plan and Code Generation

The search plan and code generation techniques described in this subsection originate from [12] and [13], respectively. When pattern matching is invoked, variables can already be bound to restrict the search. The corresponding binding information of all variables is called *initial adornment* a_0. By using the initial adornment, a search plan generation algorithm [12] filters and sorts the operations to prepare a search plan, which is then processed by a code generator to produce executable program code.

A *search plan* is a sequence of operations, which handles each predicate of the pattern specification *exactly once*, and terminates in an adornment with only `B` characters, which means that all the variables are bound in the end.

Example. Let us suppose that customer Cu, order Or and article Ar are bound in the initial adornment, while the three attribute variables are free, and the search plan shown as comments on the right side of Fig. 6 has been generated. As both variables Cu and Or are initially bound, the operation $\texttt{orders}_{\texttt{BB}}\texttt{(Cu,Or)}$ can be applied, which does not change the adornment. The second operation looks up the value of the totalCost attribute of the order stored in variable Or, and assigns this value to variable Or.totalCost, which gets bound by this act. Similarly, the third operation looks up the value of the price attribute of the article stored in variable Ar, and assigns this value to variable Ar.price. At this point, variables Or.totalCost and Ar.price are already bound, so their sum can be calculated and assigned to variable Or.totalCost' by the fourth operation. Finally, the NAC predicate is checked by the last operation. Note that each predicate is represented exactly once in the search plan and all variables are bound in the end, which means that the presented operation sequence is a search plan.

```
public Match addArticle_LHS(Customer Cu, Order Or, Article Ar){
   if(Cu.getOrders().contains(Or)){                  // orders_BB(Cu,Or)
     double Or_totalCost=Or.getTotalCost();          // totalCost_BF(Or,Or.totalCost)
     double Ar_price=Ar.getPrice();                  // price_BF(Ar,Ar.price)
     double Or_totalCost_p=Or_totalCost + Ar_price;  // add_FBB(Or.totalcost',Or.totalCost,Ar.price)
     if(!addArticleNAC(Cu,Or,Ar,                     // addArticleNAC_BBBBBB(Cu,Or,Ar,
         Ar_price,Or_totalCost,Or_totalCost_p)){     //      Ar.price,Or.totalCost,Or.totalCost')
       return new Match(Cu,Or,Ar,Ar_price,Or_totalCost,Or_totalCost_p);
     }
   }
   return null;
 }
```

Fig. 6: Pattern matching code and the corresponding search plan

5 Related Work

The idea of constructing precondition application conditions for GT-rules from graph constraints was originally proposed in [4]. The expressiveness of constraints was extended in [14] that allows arbitrary nesting of constraints. In [15] the approach was generalized to the generic notion of high-level replacement systems.

Including attributes in the theory of graph transformation has been proposed in [16], where attributed graphs are specified by assigning to the label nodes terms of a freely generated term algebra over a set of variables. Although this approach can generate application conditions from attributed graph constraints, it comes with some technical difficulties (arising from the conceptual complexity of combining graphs with algebras) and it has limitations regarding expressiveness compared to symbolic graphs introduced in [11]. Compared to the original notion of symbolic graphs, which allows first order formulas expressing arbitrary constraint satisfaction problems (CSP), we can only handle CSPs for which we can generate valid search plans, which are basically those that have a *unique* solution. However, we can solve these CSPs in linear time in the number of predicates as every predicate is evaluated only once in a valid search plan. Despite this limitation, our approach remains still more expressive than attributed graphs, as these are restricted to (conditional) equations [11]. In [6] OCL preconditions for graph transformation rules are derived from graph constraints with OCL expressions. Consequently, complex expressions including cardinality constraints on collections are allowed. However, different concepts like graphs for expressing structural restrictions and OCL expression for attribute conditions are used that might complicate an efficient evaluation of the preconditions if different engines for the evaluation of graph conditions and OCL expressions are used. In our proposal, restrictions on graphs and attributes can be evaluated arbitrarily intertwined using a single engine. Moreover, as shown in [12] cost values can be assigned to (all) operations guiding the search plan generation process in optimizing the order of operations. A correctness proof is also not given in [6]. [17] suggested an approach based on Hoare-calculus for transforming postconditions to preconditions, which involved the handling of simple attribute conditions. However, implementation issues were not discussed in [17].

6 Conclusion

In this paper, we proposed an attribute handling approach for generating preconditions from graph constraints, whose correctness has been proven using the formalism of symbolic graphs. The presented technique generates preconditions that are transformed to pattern specifications, which are then processed by advanced optimization algorithms [12] to automatically derive search plans, in which the evaluation of attribute and structural restrictions can be intertwined.

One open issue is to analyze the generated NACs and to keep only the weakest preconditions, which could accelerate rule applications at runtime. Another interesting topic could be to determine whether symbolic graphs provide the right properties to construct precondition NACs from more complex constraints (e.g., nested constraints), however, we intentionally left this analysis for future work in favor for an implementation.

References

1. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer Verlag (2006)
2. Heckel, R.: Compositional verification of reactive systems specified by graph transformation. In: In FASE 1998. Volume 1382 of LNCS., Springer (1998) 138–153
3. Koch, M., Mancini, L.V., Parisi-Presicce, F.: A graph-based formalism for RBAC. ACM Trans. Inf. Syst. Secur. **5**(3) (August 2002) 332–365
4. Heckel, R., Wagner, A.: Ensuring consistency of conditional graph rewriting – a constructive approach. In Corradini, A., Montanari, U., eds.: Proc. of Joint COMPUGRAPH/SEMAGRAPH Workshop. Volume 2 of ENTCS., Volterra, Pisa, Italy, Elsevier (August 1995) 118–126
5. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae **26**(3/4) (1996) 287–313
6. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Synthesis of OCL pre-conditions for graph transformation rules. In Tratt, L., Gogolla, M., eds.: Proc. of the ICMT. Volume 6142 of LNCS., Springer (2010) 45–60
7. Orejas, F., Lambers, L.: Delaying constraint solving in symbolic graph transformation. In Ehrig, H., Rensink, A., Rozenberg, G., Schrr, A., eds.: Graph Transformations. Volume 6372 of LNCS. Springer (2010) 43–58
8. Horváth, Á., Varró, G., Varró, D.: Generic search plans for matching advanced graph patterns. In Ehrig, K., Giese, H., eds.: Proc. of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques. Volume 6 of Electronic Communications of the EASST., Braga, Portugal (March 2007)
9. Anjorin, A., Varró, G., Schürr, A.: Complex attribute manipulation in TGGs with constraint-based programming techniques. In Hermann, F., Voigtländer, J., eds.: Proc. of the 1st Int. Workshop on Bidirectional Transformations. Volume 49 of ECEASST. (2012)
10. Shoenfield, J.R.: Mathematical logic. Volume 21. Addison-Wesley Reading (1967)
11. Orejas, F., Lambers, L.: Symbolic attributed graphs for attributed graph transformation. In Ermel, C., Ehrig, H., Orejas, F., Taentzer, G., eds.: Proc. of the ICGT. Volume 30 of Electronic Communications of the EASST. (2010)
12. Varró, G., Deckwerth, F., Wieber, M., Schürr, A.: An algorithm for generating model-sensitive search plans for pattern matching on EMF models. Software and Systems Modeling (2013) Accepted paper.
13. Varró, G., Anjorin, A., Schürr, A.: Unification of compiled and interpreter-based pattern matching techniques. In Tolvanen, J.P., Vallecillo, A., eds.: Proc. of the 8th ECMFA. Volume 7349 of LNCS., Springer (2012) 368–383
14. Habel, A., Pennemann, K.H.: Nested constraints and application conditions for high-level structures. In Kreowski, H.J., Montanari, U., Orejas, F., Rozenberg, G., Taentzer, G., eds.: Formal Methods in Software and Systems Modeling. Volume 3393 of LNCS. Springer (2005) 293–308
15. Ehrig, H., Ehrig, K., Habel, A., Pennemann, K.H.: Constraints and application conditions: From graphs to high-level structures. In Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G., eds.: Graph Transformations. Volume 3256 of LNCS. Springer (2004) 287–303
16. Ehrig, H., Prange, U., Taentzer, G.: Fundamental theory for typed attributed graph transformation. In Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G., eds.: Proc. ICGT 2004. Volume 3256 of LNCS., Springer (2004) 161–177
17. Poskitt, C.M., Plump, D.: Hoare-style verification of graph programs. Fundamenta Informaticae **118**(1) (2012) 135–175