Delaying Constraint Solving in Symbolic Graph Transformation

Fernando Orejas ^{*1} and Leen Lambers²

¹ Universitat Politècnica de Catalunya, Spain
orejas@lsi.upc.edu

² Hasso Plattner Institut, Universität Potsdam, Germany
Leen.Lambers@hpi.uni-potsdam.de

Abstract. Applying an attributed graph transformation rule to a given object graph always implies some kind of constraint solving. In many cases, the given constraints are almost trivial to solve. For instance, this is the case when a rule describes a transformation $G \Rightarrow H$, where the attributes of H are obtained by some simple computation from the attributes of G . However there are many other cases where the constraints to solve may be not so trivial and, moreover, may have several answers. This is the case, for instance, when the transformation process includes some kind of searching. In the current approaches to attributed graph transformation these constraints must be completely solved when defining the matching of the given transformation rule. This kind of *early binding* is well-known from other areas of Computer Science to be inadequate. For instance, the solution chosen for the constraints associated to a given transformation step may be not fully adequate, meaning that later, in the search for a better solution, we may need to backtrack this transformation step.

In this paper, based on our previous work on the use of *symbolic graphs* to deal with different aspects related with attributed graphs, including attributed graph transformation, we present a new approach that allows us to delay constraint solving when doing attributed graph transformation. In particular we show that the approach is sound and complete with respect to *standard* attributed graph transformation. A running example, where a graph transformation system describes some basic operations of a travel agency, shows the practical interest of the approach.

Key words: Attributed graph transformation, symbolic graph transformation, lazy transformation

1 Introduction

Attributed graphs and attributed graph transformation play a significant role in most applications of graph transformation. In practice, an attributed graph

^{*} This work has been partially supported by the CICYT project (ref. TIN2007-66523) and by the AGAUR grant to the research group ALBCOM (ref. 00516)

transformation rule is like a normal rule, but some nodes or edges are labelled by expressions over some given variables. Then, defining a match m of a rule to a given object graph, whose attributes are some concrete values, means finding the values that must be assigned to the variables occurring in the rule, so that the value of each expression associated to each node or edge e on the left-hand side of the rule coincides with the value of the corresponding attribute associated to $m(e)$ in the object graph. That is, defining a match of a rule means solving a set of constraints. In many cases, these constraints are trivial. For instance, when a rule describes a transformation $G \Rightarrow H$, where the attributes in H are obtained by some simple computation from the attributes in G , e.g. when the expressions used as attributes in the left-hand side of the rules are just variables, and more general expressions, defined over these variables, only occur in the right-hand side. However there are many other cases where the constraints to solve may be not so trivial and, moreover, may have several answers. For instance, when the transformation process includes some kind of searching (e.g. when the right-hand side of a rule involves a variable which does not occur explicitly on the left-hand side). In existing approaches to attributed graph transformation these constraints must be completely solved when defining the matching of the given transformation rule. Then, finding a match means choosing one specific solution. This kind of *early binding* is well-known from other areas of Computer Science to be inadequate. One problem is that the solution chosen for the constraints associated to a given transformation step may not be fully adequate, meaning that we may need to backtrack this transformation step. The approach taken in areas like Constraint Logic Programming [8], in which our approach is inspired, is to postpone solving the constraints as much as possible, checking meanwhile their satisfiability. Then, not only may we avoid some useless backtracking, but we have other advantages. On the one hand, checking satisfiability may be computationally simpler than solving a set of constraints, meaning that it may also be simpler to apply a transformation step. On the other hand, some constraints which may be difficult to solve at a given moment, may become simpler, even trivial, because of the interaction with constraints defined by later steps.

In [12], when studying the problem of defining graph constraints over attributed graphs, we saw that the existing approaches [11, 1, 7, 3, 14] were not fully adequate for our purposes. These approaches presented different kinds of technical difficulties together with a limited expressive power for defining conditions on the attributes. To avoid these problems we presented a new formal approach which (we believe) is conceptually simpler and more powerful than existing approaches. It is simple because, in our approach, attributed graphs are not defined as some kind of combination of a graph and an algebra, as in [11, 1, 7, 3], nor do we have to establish a difference between transformation rules and rule schemata, as in [14]. At the same time, our approach is expressively more powerful, not only because we can define graph constraints with arbitrary conditions on the attributes as we aimed, but also because we can define transformation rules that cannot be defined in other approaches, as shown in [13]. Graphs in our approach are called *symbolic graphs*, because the attributes in

the graph are represented symbolically by variables whose possible values are specified by a set of formulas.

In this paper, we continue the work in [13], where we studied symbolic graph transformation and we compared it with attributed graph transformation as presented in [4]. In particular, now we present a new approach to symbolic graph transformation that allows us to delay constraint solving when doing attributed graph transformation. In particular we show that the approach is sound and complete with respect to *standard* attributed graph transformation. Moreover, a running example, where a graph transformation system describes some basic operations of a travel agency, shows the practical interest of the approach.

The paper is organized as follows. In Section 2 we provide a reminder of some notions that are used in the rest of the paper. In Section 3 we present the category of symbolic graphs. Section 4 is dedicated to describe how (standard) symbolic graph transformation works, and Section 5 to present the new notion of lazy transformation and to prove its soundness and completeness. In Section 6, we compare our approach with other related work and we draw some conclusions.

2 Preliminaries

We assume that the reader has a basic knowledge on algebraic specification and on graph transformation. For instance, we advise to look at [5] for more detail on algebraic specification or at [4] for more detail on graph transformation.

2.1 Basic algebraic concepts and notation

A signature $\Sigma = (S, \Omega)$ consists of a set of sorts S , and Ω is a family of operation and predicate symbols typed over these sorts. A Σ -algebra A consists of an S -indexed family of sets $\{A_s\}_{s \in S}$ and a function op_A (resp. a relation pr_A) for each operation op (resp. each predicate pr) in the signature. A Σ -homomorphism $h : A \rightarrow A'$ consists of an S -indexed family of functions $\{h_s : A_s \rightarrow A'_s\}_{s \in S}$ commuting with the operations.

Given a signature Σ , we denote by T_Σ the term algebra, consisting of all the possible Σ -(ground) terms. Given any Σ -algebra A there is a unique homomorphism $h_A : T_\Sigma \rightarrow A$. In particular, h_A yields the value of each term in A . Similarly, $T_\Sigma(X)$ denotes the algebra of all Σ -terms with variables in X , and given a variable assignment $\sigma : X \rightarrow A$, this assignment extends to a unique homomorphism $\sigma^\# : T_\Sigma(X) \rightarrow A$ yielding the value of each term after the replacement of each variable x by its value $\sigma(x)$. In particular, when an assignment is defined over the term algebra, i.e. $\sigma : X \rightarrow T_\Sigma$, then $\sigma^\#(t)$ denotes the term obtained by substituting each variable x in t by the term $\sigma(x)$. However, for simplicity, even if it is an abuse of notation, we will write $\sigma(t)$ instead of $\sigma^\#(t)$.

2.2 E-graphs

E-graphs are introduced in [4] as a first step to define attributed graphs. An E-graph is a kind of labelled graph, where nodes and edges may be decorated

with labels from a given set E . The difference with labelled graphs, as commonly understood, is that in labelled graphs it is usually assumed that each node or edge is labelled with a given number of labels, which is fixed a priori. In the case of E-graphs, each node or edge may have any arbitrary (finite) number of labels, which is not fixed a priori. Actually, in the context of graph transformation, the application of a rule may change the number of labels of a node or of an edge.

Formally, in E-graphs labels are considered as a special class of nodes and the labeling relation between a node or an edge and a given label is represented by a special kind of edge. For instance, this means that the labeling of an edge is represented by an edge whose source is an edge and whose target is a label.

Definition 1 (E-Graphs and morphisms). *An E-graph over the set of labels X_G is a tuple $G = (V_G, X_G, E_G, E_{NL}, E_{EL}, \{s_j, t_j\}_{j \in \{G, NL, EL\}})$ consisting of:*

- V_G and X_G , the sets of graph nodes and of label nodes, respectively.
- E_G , E_{NL} , and E_{EL} , the sets of graph edges, node label edges, and edge label edges, respectively.

and the source and target functions:

- $s_G : E_G \rightarrow V_G$ and $t_G : E_G \rightarrow V_G$
- $s_{NL} : E_{NL} \rightarrow V_G$ and $t_{NL} : E_{NL} \rightarrow X_G$
- $s_{EL} : E_{EL} \rightarrow E_G$ and $t_{EL} : E_{EL} \rightarrow X_G$

Given the E-graphs G and G' , an E-graph morphism $f : G \rightarrow G'$ is a tuple, $\langle f_V : V_G \rightarrow V_{G'}, f_X : X_G \rightarrow X_{G'}, f_E : E_G \rightarrow E_{G'}, f_{E_{NL}} : E_{NL} \rightarrow E'_{NL}, f_{E_{EL}} : E_{EL} \rightarrow E'_{EL} \rangle$ such that f commutes with all the source and target functions.

*E-graphs and E-graph morphisms form the category **E – Graphs**.*

The following constructions on E-graphs are needed in the sections below. The first one tells us how to replace the labels of an E-graph:

Definition 2 (Label substitution). *Given a set of labels X' , an E-graph $G = (V_G, X_G, E_G, E_{NL}, E_{EL}, \{s_j, t_j\}_{j \in \{G, NL, EL\}})$, and a function $h : X_G \rightarrow X'$ we define the graph resulting from the substitution of X_G along h , denoted $h(G)$, the E-graph $h(G) = (V'_G, X', E'_G, E'_{NL}, E'_{EL}, \{s'_j, t'_j\}_{j \in \{G, NL, EL\}})$ defined:*

- $V'_G = V_G, E'_G = E_G, E'_{NL} = E_{NL}, E'_{EL} = E_{EL}, \{s'_j = s_j\}_{j \in \{G, NL, EL\}}$, and $t'_G = t_G$
- For every $e \in E'_{NL} : t'_{NL}(e) = h(t_{NL}(e))$
- For every $e \in E'_{EL} : t'_{EL}(e) = h(t_{EL}(e))$

Moreover h induces the definition of the morphism $h^ : G \rightarrow G'$, with $h^* = \langle id_V, h, id_{E_G}, id_{E_{NL}}, id_{E_{EL}} \rangle$.*

$Reach(G)$ eliminates all the labels in an E-graph which are not bound to a node or an edge and $Restr(G, Y)$ eliminates all the labels which are not in Y (we assume that they are not reachable):

Definition 3 (Reachable subgraph and restriction subgraph). *Given an E-graph $G = (V_G, X_G, E_G, E_{NL}, E_{EL}, \{s_j, t_j\}_{j \in \{G, NL, EL\}})$, we define the reachable subgraph of G , $Reach(G)$ as the largest subgraphs of G where $X_{Reach(G)}$ consists of all labels x such that:*

- *There is a node label edge $nl \in E_{NL}$ such that $x = t_{NL}(nl)$, or*
- *there is an edge label edge $el \in E_{EL}$ such that $x = t_{EL}(el)$.*

Given a morphism $f : G \rightarrow G'$ we denote by $Reach(f) : Reach(G) \rightarrow G'$ the restriction of f to the subgraph $Reach(G)$.

Similarly, we define the restriction of G to a set of labels Y , $Restr(G, Y)$, where $X_{Reach(G)} \subseteq Y$ as the largest subgraph of G where $X_{Restr(G, Y)} = Y$.

3 Symbolic graphs

If we consider that an attributed graph is like an E-graph whose labels are values over the given data algebra, a symbolic graph can be seen as the specification of a class of attributed graphs. In particular, a symbolic graph consists of an E-graph G whose labels are variables, together with a set of formulas Φ that constrain the possible values of these variables. We consider that a symbolic graph denotes the class of all attributed graphs where the variables in the E-graph have been replaced for values that make Φ true in the given data domain. For instance, the symbolic graph below specifies a class of attributed graphs, including distances in the edges, that satisfy the well-known triangle inequality.

Definition 4 (Symbolic graphs and morphisms). *A symbolic graph over the data Σ -algebra D , with $\Sigma = (S, \Omega)$, is a pair $\langle G, \Phi \rangle$, where G is an E-graph over an S -sorted set of variables $X = \{X_s\}_{s \in S}$, i.e. $X_G = \cup_{s \in S} X_s$, and Φ is a set of first-order Σ -formulas with free variables in X and including elements in D as constants.*

Given symbolic graphs $\langle G_1, \Phi_1 \rangle$ and $\langle G_2, \Phi_2 \rangle$ over D , a symbolic graph morphism $h : \langle G_1, \Phi_1 \rangle \rightarrow \langle G_2, \Phi_2 \rangle$ is an E-graph morphism $h : G_1 \rightarrow G_2$ such that $D \models \Phi_2 \Rightarrow h(\Phi_1)$, where $h(\Phi_1)$ is the set of formulas obtained when replacing in Φ_1 every variable x_1 in the set of labels of G_1 by $h_X(x_1)$.

Symbolic graphs over D together with their morphisms form the category $\mathbf{SymbGraphs}_D$.

As said in the definition, we consider that Φ is a set of arbitrary first-order Σ -formulas. However, for practical purposes, we may want to restrict the class of formulas that can occur in this set, since first-order formula satisfiability is an

undecidable problem. In principle, the only condition that we need to ensure that the results presented in this paper hold is that the given class of formulas is closed under conjunction, disjunction and existential quantification, because these are the connectives needed to ensure the existence of pushouts and pullbacks in the category of symbolic graphs.

Remark 1 (Symbolic typed graphs). Even if along the paper, for simplicity, we have only considered untyped graphs, in our running example we consider that our graphs are typed. We can easily extend all our theory to the case of typed graphs using the standard technique. In particular, we just have to see typed graphs as morphisms $SG \rightarrow TG$, where SG is a symbolic graph and TG is the given type graph being a specific symbolic graph (i.e. the category of typed symbolic graphs over the type graph TG is the slice category over TG). Moreover, in this case, type graphs TG must be symbolic graphs $TG = \langle ETG, \mathbf{False} \rangle$, where ETG is an E-graph over a set of labels X_{ETG} consisting of just one variable for each sort in the given data signature, and $\mathbf{False}$ is the set consisting just of the *false* formula. In this way, given a symbolic graph $SG = \langle G, \Phi \rangle$, any (typing) morphism $t : G \rightarrow ETG$ for G is also a (typing) morphism $t : SG \rightarrow TG$.

Definition 5 (Semantics of symbolic graphs). *The semantics of a symbolic graph $\langle G, \Phi \rangle$ over a data algebra D is a class of attributed graphs defined:*

$$Sem(\langle G, \Phi \rangle) = \{ \langle \sigma(G), D \rangle \mid \sigma : X_G \rightarrow D \text{ and } D \models \sigma(\Phi) \}$$

where $\sigma(G)$ denotes the graph obtained according to Def. 2.

It may be noted that the class of attributed graphs denoted by a symbolic graph may be empty if the associated condition is unsatisfiable.

Every attributed graph may be seen as a symbolic graph by just replacing all its values by variables and by including an equation $x_v = v$, for each value v in the data algebra, into the corresponding set of formulas, where x_v is the variable that has replaced the value v . We call these kind of symbolic graphs *grounded symbolic graphs*. For instance, below on the right, we can see the symbolic representation of the attributed graph on the left.

In [13] we showed that the subcategories of attributed graphs and grounded symbolic graphs are equivalent, provided that the given data algebra is finitely generated. As a consequence, we may identify them. For instance, we may consider that, if $h : SG \rightarrow SG'$ is a symbolic graph morphism and SG' is grounded then we may say that, for a given variable x , $h(x) = v$, with $v \in D$, instead of saying that $h(x) = x_v$. In addition, in this case, the condition $D \models \Phi' \Rightarrow h(\Phi)$, that the symbolic morphism h must satisfy, can be simplified:

Fact 1 *Given a symbolic graph $SG = \langle G, \Phi \rangle$ and a grounded symbolic graph $SG' = \langle G', \Phi' \rangle$, an E-graph morphism $h : G \rightarrow G'$ is a symbolic graph morphism if and only if $D \models h'(\Phi)$, where $h' : X_G \rightarrow D$ is the mapping defined $\forall x \in X_G : h'(x) = v$ if $h(x) = x_v$.*

The reason is quite obvious. If $X_G = \{x_1, \dots, x_n\}$ and $h(x_1) = x_{v_1}, \dots, h(x_n) = x_{v_n}$, then $D \models \{x_{v_1} = v_1, \dots, x_{v_n} = v_n\} \Rightarrow \Phi[x_1/x_{v_1}, \dots, x_n/x_{v_n}]$ if and only if $D \models \Phi[x_1/v_1, \dots, x_n/v_n]$, where $\Phi[x/a]$ denotes the set of formulas obtained replacing in Φ all occurrences of x by a . This means that defining a morphism from $SG = \langle G, \Phi \rangle$ into a grounded symbolic graph is (partially) a constraint satisfaction problem. In particular we need to find the values that must be assigned to the variables in Φ so that it is satisfied.

Sometimes, as we will see in Section 5, we want to work only with graphs where no variable is bound to two different elements of the graph (i.e. every variable occurs at most once in the graph). We call these graphs in *normal form*. It should be clear that every symbolic graph G is equivalent to some symbolic graph G' in normal form, in the sense that $Sem(G) = Sem(G')$. It is enough to replace each repeated occurrence of a variable x by a fresh variable y , and to include the equality $x = y$ in $\Phi_{G'}$.

In [13], we also showed that symbolic graphs are an adhesive HLR category [10, 6] taking as M-morphisms all injective graph morphisms where the formulas constraining the source and target graphs are equivalent (in most cases they will just be the same formula).

Definition 6 (M-morphisms). *An M-morphism $h : \langle G, \Phi \rangle \rightarrow \langle G', \Phi' \rangle$ is a monomorphism such that $X_G \cong X_{G'}$, i.e. h_X is a bijection, and $D \models h(\Phi) \Leftrightarrow \Phi'$*

The intuition of this definition is based on the use of our category of symbolic graphs to define graph transformation. More precisely, we think that the most reasonable formulation of graph transformation rules in our context is based on defining a graph transformation rule as an E-graph transformation rule, together with a set of formulas that globally constrain and relate all the variables in the rule. This is equivalent to consider that the left and right-hand sides (and also the interface) of a rule are constrained by the same set of formulas.

We will not study in detail all the constructions and results that are needed to prove that **SymbGraphs_D** is an adhesive HLR category (the interested reader is addressed to [13]). However, it is important to see how pushouts work in order to understand graph transformation:

Proposition 1. [13] *Diagram (1) below is a pushout if and only if diagram (2) is also a pushout and $D \models \Phi_3 \Leftrightarrow (g_1(\Phi_1) \cup g_2(\Phi_2))$.*

$$\begin{array}{ccc}
 \langle G_0, \Phi_0 \rangle & \xrightarrow{h_1} & \langle G_1, \Phi_1 \rangle \\
 h_2 \downarrow & (1) & \downarrow g_1 \\
 \langle G_2, \Phi_2 \rangle & \xrightarrow{g_2} & \langle G_3, \Phi_3 \rangle
 \end{array}
 \qquad
 \begin{array}{ccc}
 G_0 & \xrightarrow{h_1} & G_1 \\
 h_2 \downarrow & (2) & \downarrow g_1 \\
 G_2 & \xrightarrow{g_2} & G_3
 \end{array}$$

Therefore, as said above, we have:

Theorem 1. [13] **SymbGraphs_D** is adhesive HLR.

Example 1 (Symbolic graphs). In our running example we specify a travel agency in terms of typed symbolic graphs and symbolic graph transformation rules. In particular, the data signature of our example is:

Sorts *int, bool, city, hotel, flight*
Opns *price : hotel → int*
 location : hotel → city
 price : flight → int
 departure : flight → city
 destination : flight → city

and the data algebra, depicted by means of two tables, is:

hotel	price	city
h1	100	München
h2	90	Berlin
h3	85	Berlin
h4	70	Köln
h5	200	Berlin

flight	price	departure	destination
f1	175	Berlin	Barcelona
f2	115	Berlin	Barcelona
f3	150	Barcelona	Berlin
f4	85	Barcelona	Berlin

The type graph consists of a *Customer* that can request a hotel reservation *HotelRequest* or a flight reservation *FlightRequest*. These requests can be responded by a *FlightReservation* or a *HotelReservation*, respectively. A *Customer* has the labels *name* and *bud*, describing the total budget that the customer is willing to spend for his reservations. A *HotelRequest* has as labels the number of *nights* and *loc*, describing the city where the hotel is located. A *FlightRequest* has as labels *dep* and *dest*, the departure and destination city of the flight, respectively. The *HotelReservation* and *FlightReservation* have only one label *h* and *f* for the hotel and flight that are reserved, respectively.

Finally, a possible start graph describing the needs of a customer may be a grounded graph (represented as an attributed graph, including directly the values in the nodes) describing a customer with a budget of 450, requesting a hotel for two nights in Berlin, a flight from Barcelona to Berlin and a flight from Berlin to Barcelona. We have not assigned a specific value to the name of the customer, because it is not relevant for the example:

4 Symbolic graph transformation

As usual, symbolic graph transformation rules are spans of M-morphisms. This means, as explained in the previous section, that we may denote *symbolic graph transformation rules* as pairs $\langle L \hookleftarrow K \hookrightarrow R, \Phi \rangle$, where L, K and R are E-graphs over the same set of labels X and Φ is a set of formulas over X and over the values in D . Intuitively, Φ relates the attributes in the left and right-hand side of the rule and it may also impose some constraints on the matchings.

Definition 7 (Symbolic graph transformation rule). A symbolic graph transformation rule is a pair $\langle L \hookleftarrow K \hookrightarrow R, \Phi \rangle$, where L, K, R are E-graphs over the same set of labels X , $L \hookleftarrow K \hookrightarrow R$ is a span of E-graph inclusions³, and Φ is a set of formulas over X and over the values in the given data algebra D .

Example 2 (Symbolic graph transformation rules). Below we can see the rules that describe some operations of our travel agency. The rule *reserveFlight* connects a *FlightReservation* with a *FlightRequest* for some *Customer*. Identical

³ or, in general, M-morphisms

names in rules specify that the corresponding nodes are preserved. Edges between nodes with identical names are preserved as well. Its formula Φ , expresses that the departure city dep and destination city $dest$ of the *FlightRequest* should be equal to the departure city $departure(f)$ and destination city $destination(f)$ of the reserved flight f , respectively. Moreover the total budget bud of the customer is diminished by the price of the reserved flight $price(f)$ and as an extra constraint it is required that the new budget bud' is still bigger or equal to zero. The rule *reserveHotel* connects a *HotelReservation* with a *HotelRequest* for some *Customer*. It holds a formula Φ , expressing that the location loc of the *HotelRequest* should be identical to the location of the reserved hotel $location(h)$. The total budget bud of the customer is diminished by the price of the reserved hotel multiplied with the number of nights $price(h) * nights$ and as an extra constraint it is required again that the new budget bud' is still greater or equal to zero.

As usual, the application of a graph transformation rule $\langle L \hookrightarrow K \hookrightarrow R, \Phi \rangle$ can be defined by a double pushout in the category of symbolic graphs. However, since M-morphisms are preserved by pushouts, the application of a symbolic transformation rule can be defined in terms of a transformation of E-graphs.

Fact 2

1. Given a symbolic graph transformation rule $\langle L \hookrightarrow K \hookrightarrow R, \Phi \rangle$ over a given data algebra D and an E-graph morphism $m : L \rightarrow G$, $\langle G, \Phi' \rangle \Rightarrow_{p,m} \langle H, \Phi' \rangle$ if and only if the diagram below is a double pushout in **E-Graphs** and $D \models \Phi' \Rightarrow m(\Phi)$.

$$\begin{array}{ccccc}
 L & \xleftarrow{\quad} & K & \xrightarrow{\quad} & R \\
 m \downarrow & (1) & \downarrow & (2) & \downarrow \\
 G & \xleftarrow{\quad} & F & \xrightarrow{\quad} & H
 \end{array}$$

2. If $\langle G, \Phi' \rangle \Rightarrow_{p,m} \langle H, \Phi' \rangle$ and $\langle G, \Phi' \rangle$ is grounded then $\langle H, \Phi' \rangle$ is grounded too, since $X_H = X_G$ and Φ' defines the values of all the variables in X_G .

According to Fact 1, if we apply the rule $\langle L \leftarrow K \hookrightarrow R, \Phi \rangle$ to a grounded symbolic graph $SG = \langle G, \Phi' \rangle$, the condition $D \models \Phi' \Rightarrow m(\Phi)$ can be seen as the constraint satisfaction problem where we find the values to assign to the variables in Φ , so that this set of formulas is satisfied in D .

Remark 2. The fact that the formula constraining the variables of the E-graph H after transformation is the same formula for the graph G before transformation does not mean that the values in the graphs do not change after transformation. The reason is that the transformation may change the bindings of variables to nodes and edges, as the example below shows.

Example 3. Below we can see how the rule **reserveHotel** presented in Example 2 is applied to a given start graph as presented in Example 1, describing a customer that would like to reserve a hotel for 2 days in Berlin and, in addition, would also like to reserve a flight from Barcelona to Berlin and return.

In this transformation, a hotel in Berlin is chosen non-deterministically. This choice is made when defining the matching between the left-hand side of the rule and the given graph. Even if it is not explicitly depicted, the left-hand side of **ReserveHotel** includes the variable h . Then, when defining the match morphism we must bind that variable to some hotel hi in D_{hotel} such that the formula

$$\text{Berlin} = \text{location}(hi) \ \&\& \ \text{bud}' = 450 - (\text{price}(hi) * 2) \ \&\& \ \text{bud}' \geq 0$$

is satisfied. Here, the chosen hotel, $h5$, satisfies the above formula for $h = h5$ and $\text{bud}' = 50$. For this reason, the value of the attribute bud in the resulting graph is set to 50. This kind of early binding may not be very efficient. For instance, after the transformation defined above no flight reservation is possible, because the budget of the user would not be sufficient for reserving any flight connecting Barcelona and Berlin. Hence, we would need to backtrack this transformation and to try a different hotel. This situation also occurs when working with attributed graphs and attributed graph transformation rules as in [4].

5 Lazy symbolic graph transformation

The problem shown in Example 3 is that, when defining a match for a rule $p = \langle L \leftarrow K \hookrightarrow R, \Phi_p \rangle$, this match must also be defined on variables that are bound to a node or edge in R but not bound to any node or edge in L (i.e.

intuitively, these variables are not part of L). The basic idea of our approach is that, when applying a rule, we do not need to match these variables if we do not want to have early binding. This means that the match morphism for p is not defined on L , but on $Reach(L)$ (cf. Definition 3) and, then, when we apply the rule to a graph SG , we add to the result of the transformation the formulas in Φ_p , since these formulas define the possible values of the non-matched variables with respect to the values of the rest of the variables.

Definition 8 (Lazy graph transformation). *Given a symbolic graph $SG = \langle G, \Phi_G \rangle$, a transformation rule $p = \langle L \hookleftarrow K \hookrightarrow R, \Phi_p \rangle$, where L, K and R are in normal form and where the sets of labels in SG and in p are disjoint⁴, and given a morphism $m : Reach(L) \rightarrow G$, we define the lazy transformation of SG by p via m , denoted $SG \Rightarrow_{p,m} SH$, as the symbolic graph $SH = \langle H, \Phi_H \rangle$, where H is obtained by the double pushout below in the category of E-graphs:*

$$\begin{array}{ccccc}
 Reach(L) & \xleftarrow{\quad} & Restr(K, X_{Reach(L)}) & \xrightarrow{\quad} & R \\
 m \downarrow & & \downarrow & & \downarrow \\
 G & \xleftarrow{\quad} & D & \xrightarrow{\quad} & H
 \end{array}
 \quad \begin{array}{c} (1) \end{array} \quad \begin{array}{c} (2)$$

and where $\Phi_H = \Phi_G \cup m(\Phi_p)$.

As usual, we write $SG \Rightarrow SH$ if the given rule and match can be left implicit, and we write $\Rightarrow^*$ to denote the reflexive and transitive closure of $\Rightarrow$.

Intuitively, H is similar to the E-graph obtained by a standard symbolic transformation, except that it would include all the variables in $X_R \setminus X_{Reach(L)}$ instead of their corresponding matchings.

In general, m is not defined on all the variables in Φ_p , but only on the variables in $Reach(L)$ and, therefore, we could consider that the definition of Φ_H is incorrect. However, we will consider that $m(\Phi_p)$ denotes the set of formulas obtained replacing in every formula $\alpha \in \Phi_p$ every variable x in $X_{Reach(L)}$ by $m(x)$, i.e. the variables which are not in $X_{Reach(L)}$ are not replaced.

It may also be noted that the resulting condition Φ_H may be unsatisfiable for the given data algebra D . This would mean that $Sem(SH)$ is empty. In general, we probably want to avoid this kind of situation, which means that we assume that the lazy application of a rule preserves satisfiability.

Fact 3 *If $SG \Rightarrow_{p,m} SH$, we can obtain SH from SG by applying a double pushout transformation in the category of symbolic graphs using the span $lazy(p) = \langle \langle Reach(L), Restr(\Phi_p, X_{Reach(L)}) \rangle, \hookleftarrow \langle K, Restr(\Phi_p, X_{Reach(L)}) \rangle \hookrightarrow \langle R, \Phi_p \rangle \rangle$ and the match m , i.e. $SG \Rightarrow_{lazy(p),m} SH$, where $Restr(\Phi, X)$ is the formula obtained quantifying existentially all free variables in Φ which are not in X , i.e. $\exists(\text{freevar}(\Phi) \setminus X)\Phi$. However, we may note that $lazy(p)$ is not a symbolic graph transformation rule, since the embeddings involved in the span in general are not M-morphisms.*

⁴ If needed we can rename the labels apart and we can transform L, K and R into normal form, as described in Section 3.

Example 4. For example, if we apply the rule `ReserveHotel` as in Example 3 without binding the variable h to some concrete value hi in D_{hotel} . Then we obtain a lazy graph transformation step as depicted in the first step of the lazy graph transformation in the figure in Example 5. After this first step the formula

`Berlin = location(h) && bud=450-(price(h) * 2) && bud>=0`

should be satisfied. Given the data algebra D as presented in Example 1, it is obvious that this formula is satisfiable.

Let us now study the relation between symbolic graph transformation and lazy transformation. The first result tells us that lazy transformation may be seen as a sound and complete implementation of symbolic graph transformation over grounded symbolic graphs (i.e. attributed graphs). More precisely, in the case of grounded symbolic graphs, for every transformation $SG \xRightarrow{*} SH$, if AH is the attributed graph denoted by SH then there is a lazy transformation $SG \xRightarrow{*} SH'$ such that AH is in the semantics of SH' , and conversely if there is a lazy transformation $SG \xRightarrow{*} SH'$ then every attributed graph AH in $Sem(SH')$ can be obtained by a symbolic graph transformation from AG . This means that, semantically, it does not matter if, each time we apply a transformation rule, we solve the constraints associated to that application, or if we apply lazy transformation and, at the end, we solve the constraints of the resulting graph.

Theorem 2 (Soundness and completeness of lazy graph transformation). *Given a grounded symbolic graph $SG = \langle G, \Phi_G \rangle$, then:*

1. **Completeness** *For every symbolic transformation $SG \xRightarrow{*} SH$ there is a lazy transformation $SG \xRightarrow{*} SH'$ such that $Sem(SH) \subseteq Sem(SH')$.*
2. **Soundness** *For every lazy transformation $SG \xRightarrow{*} SH'$ and every attributed graph $AH \in Sem(SH')$ there is a symbolic transformation $SG \xRightarrow{*} SH$ such that $AH \in Sem(SH)$.*

If we consider the general case, where the transformations are applied to arbitrary symbolic graphs, lazy transformation is still complete in the above sense, but it may be considered, technically, not sound. In particular, it may be possible to apply a rule $p = \langle L \hookrightarrow K \hookrightarrow R, \Phi_p \rangle$ to a graph $SG = \langle G, \Phi_G \rangle$ to apply lazy transformation, but it may be impossible to apply the same rule to SG to do symbolic transformation. The reason is that for symbolic transformation we require that $D \models \Phi_G \Rightarrow m(\Phi_p)$, for a given match m , while in the case of lazy transformation it may be enough that $\Phi_G \cup m(\Phi_p)$ is satisfiable in D . This may happen, for instance, if the condition on G is $0 < X < 5$ and the condition on the rule is $3 < X < 7$. In that case, we would be unable to apply symbolic graph transformation, since $0 < X < 5$ does not imply $3 < X < 7$. However, it could be possible to apply lazy transformation and, then, the resulting condition would be $0 < X < 5 \wedge 3 < X < 7$, or equivalently $3 < X < 5$. Actually, we may consider that what happens in the case of lazy transformation is that, at each

transformation step, we narrow the conditions of the given graphs, while in the case of symbolic transformations the conditions remain invariant. In this sense, we can state an extended soundness result for lazy transformation:

Theorem 3 (Extended soundness and completeness of lazy transformation). *Given a symbolic graph $SG = \langle G, \Phi_G \rangle$, then:*

1. **Completeness** *For every symbolic transformation $SG \xRightarrow{*} SH$ there is a lazy transformation $SG \xRightarrow{*} SH'$ such that $Sem(SH) \subseteq Sem(SH')$.*
2. **Extended soundness** *For every lazy transformation $SG \xRightarrow{*} SH'$, with $SG = \langle G, \Phi_G \rangle$ and $SH' = \langle H', \Phi_{H'} \rangle$ there is a symbolic transformation $\langle G, \Phi_{H'} \rangle \xRightarrow{*} SH$ such that $SH \cong SH'$.*

Example 5. Consider the lazy graph transformation as presented below, where a hotel is reserved for two nights in Berlin and a flight from Barcelona to Berlin and return is reserved. The labels h for the hotel, $fr1.f$ and $fr2.f$ for the flights are not bound to a concrete hotel or concrete flights yet. Note that we use the $.*$ notation to distinguish labels having the same name in different nodes. At the end of this transformation a formula equivalent to:

```
Berlin = location(h) && Barcelona = departure(fr1.f) &&
Berlin = destination(fr1.f) && Berlin = departure(fr2.f) &&
Barcelona = destination(fr2.f) && bud = 450 - (price(h) * 2)
- price(fr1.f) - price(fr2.f) && bud >= 0
```

should be satisfied⁵. Solving this formula for the data algebra D given in Example 1, we obtain the following solutions for the labels $(h, fr1.f, fr2.f, bud)$: $(h2, f4, f1, 10)$, $(h3, f4, f1, 20)$, $(h2, f3, f2, 5)$, $(h3, f3, f2, 15)$, $(h2, f4, f2, 70)$ and $(h3, f4, f2, 80)$. In particular, for the solution $(h2, f4, f1, 10)$ a corresponding symbolic transformation exists, where h , $fr1.f$ and $fr2.f$ are bound to $h2$, $f4$ and $f1$, respectively. Moreover, in the end the total budget bud of the customer is 10.

To solve the resulting constraints after a sequence of transformations we may need to use a specialized constraint solver. If that solver gives the possibility of defining some kind of optimality criteria (e.g. the overall price of the reservations) then it would return us the best solution according to these criteria. However, if the data algebra D can be considered to be implemented by a database, as it may happen in this example, then the resulting constraints after a sequence of transformations could be compiled into a query to that database, meaning that all the possible solutions would just be obtained as a result of that query.

⁵ Actually, the formula that we would obtain is a bit more complex, including several (renamed) copies of the variable bud , since each application of a rule would add a new variable bud and a constraint defining it. For readability, we have preferred to use this simpler version. Anyhow, if we would allow to simplify the given formula after each transformation step, as it is sometimes done in Constraint Logic Programming, we could indeed obtain the above formula.

6 Conclusion and related work

In this paper we have seen how we can delay constraint solving in the framework of attributed graph transformation by means of symbolic graphs. The idea of the technique comes from constraint logic programming [8]. Actually, the idea underlying symbolic graphs also comes from constraint logic programming, where the domain of values, and the conditions on them are neatly separated from symbolic clauses, in a similar way we separate in symbolic graphs, the graph part from the conditions defining the value part. Symbolic graph transformation can be seen as a generalization of attribute grammars [9]. In particular, attribute grammars can be considered to provide a method for computing attributes in trees. Moreover, these attributes must be computed in a fixed way: either top-down (inherited attributes) or bottom-up (synthesized attributes).

There are three kinds of approaches to define attributed graphs and attributed graph transformation. In [11, 1] attributed graphs are seen as algebras. In particular, the graph part of an attributed graph is encoded as an algebra extending the given data algebra. In [7, 3] an attributed graph is a pair (G, D) consisting of a graph G and a data algebra D whose values are nodes in G . In [2] these two approaches are compared showing that they are, up to a certain point, equivalent. Finally, [14] is based on the use of labelled graphs to represent attributed graphs, and of rule schemata to define graph transformations involving computations on the labels. That approach has some similarities with our approach, including the simplicity provided by the separation of the algebra and the graph part of attributed graphs. We do not think that there is a direct way

of defining lazy graph transformations for those approaches, since there is no specific way of handling constraints explicitly.

There are further aspects related to symbolic graph transformations that deserve some future work. Addressing the implementation of these techniques could be an obvious goal. However, there are other fundamental aspects which may be interesting to study, like developing specific techniques for checking confluence for symbolic transformation or the definition of symbolic graph transformation with borrowing contexts, avoiding some existing technical difficulties.

References

1. Berthold, M., Fischer, I., Koch, M.: Attributed graph transformation with partial attribution. In: GRATRA 2000. Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems. pp. 171–178 (2000)
2. Ehrig, H.: Attributed graphs and typing: Relationship between different representations. *Bulletin of the EATCS* 82, 175–190 (2004)
3. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamental theory of typed attributed graph transformation based on adhesive HLR-categories. *Fundamenta Informaticae* 74(1), 31–61 (2006)
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs of Theoretical Computer Science, Springer (2006)
5. Ehrig, H., Mahr, B.: *Fundamentals of Algebraic Specifications 1: Equations and Initial Semantics*. EATCS Monographs of Theoretical Computer Science, Springer (1985)
6. Ehrig, H., Padberg, J., Prange, U., Habel, A.: Adhesive high-level replacement systems: A new categorical framework for graph transformation. *Fundamenta Informaticae* 74(1), 1–29 (2006)
7. Heckel, R., Küster, J., Taentzer, G.: Towards automatic translation of uml models into semantic domains. In: *APPLIGRAPH Workshop on Applied Graph Transformation*. pp. 11–22 (2002)
8. Jaffar, J., Maher, M., Marriot, K., Stuckey, P.: The semantics of constraint logic programs. *The Journal of Logic Programming* 37, 1–46 (1998)
9. Knuth, D.: Semantics of context-free languages. *Mathematical Systems Theory* 2, 127–145 (1968)
10. Lack, S., Sobocinski, P.: Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications* 39, 511–545 (2005)
11. Löwe, M., Korff, M., Wagner, A.: An algebraic framework for the transformation of attributed graphs. In: *Term Graph Rewriting: Theory and Practice*, pp. 185–199. John Wiley, New York (1993)
12. Orejas, F.: Attributed graph constraints. In: *Graph Transformations (ICGT 2008)*. LNCS, vol. 5214. Springer (2008)
13. Orejas, F., Lambers, L.: Symbolic attributed graphs for attributed graph transformation. In: *Int. Coll. on Graph and Model Transformation On the occasion of the 65th birthday of Hartmut Ehrig* (2010)
14. Plump, D., Steinert, S.: Towards graph programs for graph algorithms. In: *Graph Transformations (ICGT’04)*. LNCS, vol. 3256, pp. 128–143. Springer (2004)