Electronic Communications of the EASST
Volume 30 (2010)

International Colloquium on Graph and Model Transformation On the occasion of the 65th birthday of Hartmut Ehrig (GraMoT 2010)

Symbolic Attributed Graphs for Attributed Graph Transformation

Fernando Orejas and Leen Lambers

25 pages

Guest Editors: Claudia Ermel, Hartmut Ehrig, Fernando Orejas, Gabriele Taentzer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Symbolic Attributed Graphs for Attributed Graph Transformation

Fernando Orejas[1] * **and Leen Lambers**[2]

[1] orejas@lsi.upc.edu
Dpt. de Llenguatges i Sistemes Informátics
Universitat Politècnica de Catalunya, Barcelona, Spain.

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

Abstract: In this paper we present a new approach to deal with attributed graphs and attributed graph transformation. This approach is based on working with what we call symbolic graphs, which are graphs labelled with variables together with a formula that constrains the possible values that we may assign to these variables. In particular, in this paper we will compare in detail this new approach with the standard approach to attributed graph transformation.

Keywords: Graph Transformation, Attributed Graphs, Symbolic Graphs

1 Introduction

The study of graph grammars and graph transformation started 40 years ago. However, the first formal approach to deal with attributed graphs is much more recent [12], even if this kind of graphs are needed in many applications of the field. Actually, the development of the fundamental theory of graph transformation for the case of attributed graphs is quite recent [7]. The reason for this late development is probably that, even if the attributed case may seem to be a straightforward generalization of the standard case, it presents some difficulties which have hampered the development of this fundamental theory. One of these difficulties lies on the complication of putting together two theoretical frameworks, algebraic specification and graph transformation, even if both are algebraic and categorical frameworks. In fact, to avoid this problem, at least to some extent, in [12] graphs are coded as algebras with the aim of having a uniform setting. The problem is that, in general, algebra transformation does not enjoy the right properties to ensure that the basic theory of graph transformation will hold.

The approach studied in [12], based on the approach presented in [10] is, in a sense, the opposite. In this case, the data algebra is embedded in the graph. More, precisely, an attributed graph is seen as a pair formed by an algebra, to define the values of the attributes of the graph, and a graph that includes all the values of the algebra as (a special kind of) nodes. This approach still has some difficulties caused by the fact that, even if the graphs of interest are defined over the same data algebra, we have to consider categories including graphs over different algebras. The reason is that, most often, the algebras in the graphs occurring in the transformation rules are

* 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).

different from the algebras in the graphs to which we apply these rules. In [7] the algebra used in rules is the freely generated term algebra over a given set of variables, i.e. attributes are terms with variables. However, one could use different algebras for defining transformation rules. In particular, for the formalization of attribute conditions, the term algebra over a set of variables is not sufficient. In this sense, in this paper we introduce a specific way to do this by taking the initial algebra associated to a given specification.

In [19], Plump and Steinert present an approach that avoids the complexity of having to deal, in a single concept, with graphs and algebras. This approach is essentially based on two ideas. On the one hand, attributed graphs are seen as labelled graphs, where the labels are defined as elements of an algebra. On the other hand, graph transformations involving computations on the labels are defined by rule *schemas*, which are similar to graph transformation rules, but defined in terms of graphs labelled by terms with variables. Then, to apply a rule schema to a given graph we must first instantiate the schema assigning data values to the variables in the schema. The result of the instantiation is a rule where the terms labeling the graphs have been replaced by the values of these terms. In our opinion, the approach has two main drawbacks. The first one is the fact that rule schemas are not first class citizens, in the sense that they need to be instantiated to define the rules. This causes that one may need to explicitly reformulate in terms of that framework most constructions and results associated to graph transformation. This would be the case, for instance, if we would want to define in that framework notions like graph constraints, typing or borrowed contexts. On the other hand, in that approach, a limitation is imposed on the number of labels that each node or edge can have. In particular, in that paper, at most one label is allowed, though it would not be difficult to fix a different limitation.

The main aim of this paper is to present a new approach to deal with attributed graph transformation, which we believe is conceptually simple but more powerful than previous approaches, as we show. The approach is partially inspired on how the clausal part and the data part are conceptually separated in Constraint Logic Programming [11, 14]. In particular, attributed graphs are presented as *symbolic graphs* consisting of a graph that includes as nodes some variables which represent the values of the attributes, together with a set of formulas that constrain the possible values of these variables. This means that the underlying algebra of values remains only implicit to define the satisfaction of these formulas. The idea underlying this approach was first introduced in [16, 17] to study graph constraints over attributed graphs and, then, used again with a similar aim to specify model transformations by means of patterns [8].

Symbolic graphs can be seen as specifications of attributed graphs. Actually, to compare the standard approach to attributed graph transformation, we define a semantics of symbolic graphs in terms of classes of attributed graphs and we show how attributed graphs can be identified with some specific kind of symbolic graphs, which we call grounded symbolic graphs. Then, to compare the expressive power of the two approaches with respect to attributed graph transformation, we first show that symbolic graphs, as it happens with attributed graphs [10], form an adhesive HLR category [13, 4] to ensure that symbolic graphs inherit the fundamental theory of graph transformation. A variant of this proof is already included in [17]. Finally, we show that attributed graph transformation systems can be coded into symbolic graph transformation systems but that the converse is not true in general.

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 particular, first, we briefly enumerate some notions from

algebraic specification; then, we present E-graphs which are used as the graph part for both attributed graphs and symbolic graphs; finally, we define the category of attributed graphs as presented in [4]. In Section 3 we present the category of symbolic graphs, showing that it is adhesive HLR. Section 4 is dedicated to relate the categories of attributed and symbolic graphs and Section 5 to compare the expressive power of both approaches with respect to attributed graph transformation. In Section 6, we draw some conclusions. Finally, in an appendix some technical details and proofs are provided.

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 [6] for more detail on algebraic specification or at [20, 4] for more detail on graph transformation.

2.1 Basic algebraic concepts and notation

As usual, a signature $\Sigma = (S, \Omega)$ consists of a set of sorts S, and a family of operation symbols of the form $op : s_1 \times \cdots \times s_n \to s$, denoted by Ω, where $n \geq 0$ and $s_1, \ldots, s_n, s \in S$. However, in this paper, signatures include also predicates. We can deal with this extended case in two ways. The first one is to consider that Σ consists, in addition, of a family of predicate symbols. The second one, which we will use, because it is simpler, is based in considering that there is a special sort in S, which we could call *logical*, and that predicate symbols are just operation symbols with profile $s_1 \times \cdots \times s_n \to \mathit{logical}$. In this case, logical connectives can be treated as operation symbols over the logical sort. In addition, the truth values **t** and **f** may be seen as constants in the signature of sort *logical*.

A Σ-algebra A consists of an S-indexed family of sets $\{A_s\}_{s \in S}$ and a function $op_A : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ for each operation $op : s_1 \times \cdots \times s_n \to s$ in the signature. A Σ-homomorphism $h : A \to A'$ consists of an S-indexed family of functions $\{h_s : A_s \to A'_s\}_{s \in S}$ commuting with the operations. Σ-algebras and Σ-homomorphisms form the category $\mathbf{Alg}_\Sigma$.

A congruence $\equiv$ on an algebra A is an S-indexed family of equivalence relations $\{\equiv_s\}_{s \in S}$ which are compatible with the operations. In this case, $A/\equiv$ denotes the quotient algebra whose elements are equivalence classes of values in A. Between A and $A/\equiv$ there is a canonical homomorphism mapping every element in A into its equivalent class.

Given signatures Σ, Σ', with $\Sigma' \subseteq \Sigma$, every Σ-algebra can be seen as a Σ'-algebra, by *forgetting* all the sorts and operations which are not in Σ'. In particular this is called the Σ'-reduct of a Σ-algebra A and is denoted by $A|_{\Sigma'}$.

Given a signature Σ, we denote by T_Σ the term algebra, consisting of all the possible Σ-(ground) terms. T_Σ is initial in $\mathbf{Alg}_\Sigma$, and the unique homomorphism $h_A : T_\Sigma \to A$ 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 $\sigma : X \to A$, this assignment extends to a unique homomorphism $\sigma^\# : T_\Sigma(X) \to A$ 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. $\sigma : X \to T_\Sigma$, then $\sigma^\#(t)$ denotes the term obtained by substituting each variable x in t by the term $\sigma(x)$.

A Σ-algebra A is finitely generated if every element in A is the value of some ground term. It is not difficult to see that if A is finitely generated there is at most one homomorphism between A and any other Σ-algebra A'.

A specification $SP = (\Sigma, Ax)$ consists of a signature Σ and a set of axioms Ax, which may be seen as terms of logical sort. Equational specifications are a special case, where the only predicate symbol is the equality. Similarly, conditional equations may be considered as a special kind of terms. Given SP, $\mathbf{Alg_{SP}}$ denotes the full subcategory of $\mathbf{Alg}_\Sigma$, consisting of all Σ-algebras A satisfying the axioms in the specification, i.e. $A \models Ax$. In the case where SP consists of equations or conditional equations there is an initial algebra in $\mathbf{Alg_{SP}}$, denoted by T_{SP}.

2.2 E-graphs

E-graphs are introduced in [4] as a first step to define attributed graphs. Intuitively, an E-graph is a kind of labelled graph, where both 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. Notice that, 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 node (a label).

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

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

and the source and target functions:

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

Given the E-graphs G and G', an *E-graph morphism* $f : G \to G'$ is a tuple, $\langle f_{V_G} : V_G \to V'_G, f_L : L \to L', f_{E_G} : E_G \to E'_G, f_{E_{NL}} : E_{NL} \to E'_{NL}, f_{E_{EL}} : E_{EL} \to E'_{EL} \rangle$ such that f commutes with all the source and target functions.

E-graphs and E-graph morphisms form the category $\mathbf{E-Graphs}$.

The following construction, which tells us how we can replace the labels of an E-graph, is used in the sections below.

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

- $V'_G = V_G, E'_G = E_G, E'_{NL} = E_{NL}, E'_{EL} = E_{EL}, \{s'_j = s_j\}_{j\in\{G,NL,EL\}}$, 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 E-graph morphism $h^* : G \to h(G)$, with $h^* = \langle id_V, h, id_{E_G}, id_{E_{NL}}, id_{E_{EL}}\rangle$.

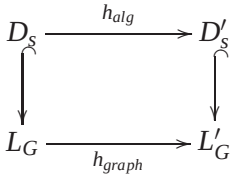
It is routine to see that $h(G)$ is indeed an E-graph and h^* is an E-graph morphism. In addition, it should be obvious that if h is a bijection then h^* is an isomorphism.

2.3 Attributed Graphs

Following [4], an attributed graph is an E-graph whose labels are the values of a given data algebra that is assumed to be included in the graph.

Definition 3 (Attributed graphs and morphisms) Given a signature Σ an *attributed graph* over Σ is a pair $\langle G, D\rangle$, where D is a given Σ-algebra, called the data algebra of the graph, and G is an E-graph such that the set L_G of labels of G consists of all the values in D, i.e. $L_G = \biguplus_{s\in S} D_s$, where s is the set of sorts of the data algebra and $\biguplus$ denotes disjoint union.

Given the attributed graphs over Σ $AG = \langle G, D\rangle$ and $AG' = \langle G', D'\rangle$, an *attributed graph morphism* $h : AG \to AG'$ is a pair $\langle h_{graph}, h_{alg}\rangle$, where h_{graph} is an E-graph morphism, $h_{graph} : G \to G'$ and h_{alg} is a Σ-homomorphism, $h_{alg} : D \to D'$ such that the values in D are mapped consistently by h_{graph} and h_{alg}, i.e. for each sort $s \in S$ the diagram below commutes:

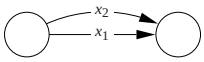


Attributed graphs and attributed graph morphisms form the category **AttGraphs**. Moreover, given a data algebra D we will denote by $\mathbf{AttGraphs}_D$ the full subcategory of **AttGraphs** consisting of attributed graphs over D.

When defining transformation rules over graphs in $\mathbf{AttGraphs}_D$, usually the algebra underlying the graphs in the rules is not D but a term algebra over the signature of D. That is, the attributes in the rules are not values but terms, typically with variables. We call these graphs *term-attributed graphs*.

Definition 4 (Term-attributed graphs) Given a signature $\Sigma = (S, \Omega)$, a *term-attributed graph* over Σ is an attributed graph over the algebra $T_\Sigma(X)$, for some S-sorted set of variables X.

Moreover, as we will see in Section 5, it is also useful to define transformation rules where the underlying algebra has been defined using a specification. In particular this is useful when we want that the match morphism used to apply the given rule satisfies some specific condition. For instance, suppose that the graph below is the left-hand side of a rule.



and suppose that, whenever we apply this rule, we would like that the corresponding match m satisfies that $m(x_1) \leq m(x_2)$. We can do this as follows. First we define a specification SP extending Σ with the variables x_1 and x_2 as constants, and the desired condition as an axiom, i.e.:

$$\begin{array}{rl} SP = \textbf{Sorts} & nat, bool \\ \textbf{Opns} & 0 : nat \\ & x_1, x_2 : nat \\ & suc : nat \to nat \\ & true, false : bool \\ & + : nat \times nat \to nat \\ & \leq : nat \times nat \to bool \\ \textbf{Axms} & (x_1 \leq x_2) = true \end{array}$$

Now, let T_{SP} be the initial algebra associated to SP, and $T_{SP}|_\Sigma$ its Σ-reduct. In T_{SP} the term $x_1 \leq x_2$ and the term $true$ belong to the same congruence class, which means that they denote the same element in $T_{SP}|_\Sigma$. Therefore, any homomorphism m from $T_{SP}|_\Sigma$ into a Σ-algebra D must satisfy that, in this algebra, $m(x_1) \leq m(x_2)$ yields the $true$ value. Hence, we should define the transformation rule over the algebra $T_{SP}|_\Sigma$.

Definition 5 (Term-Attributed graphs over a specification) Given a signature Σ and a specification $SP = (\Sigma', Ax)$, with $\Sigma \subseteq \Sigma'$, a *term-attributed graph over the specification SP extending* Σ is an attributed graph over the algebra $T_{SP}|_\Sigma$.

In [4] it has been proven that **AttGraphs** is an adhesive HLR category for a given class of M-morphisms. Let us first recall this notion [4, 13]:

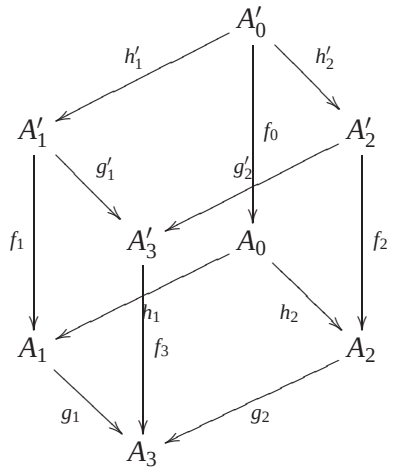
Definition 6 (Adhesive HLR category) A category $\mathbf{C}$ is adhesive HLR with respect to a class M of morphisms if:

1. M is a class of monomorphisms closed under isomorphism, composition (i.e. if $f : A \to B \in M$ and $g : B \to A \in M$ then $g \circ f \in M$), and decomposition (i.e. if $g \circ f \in M$ and $g \in M$ then $f \in M$).

2. $\mathbf{C}$ has pushouts and pullbacks along M-morphisms. Moreover, M-morphisms are closed under pushouts and pullbacks.

3. Pushouts in $\mathbf{C}$ along M-morphisms are van Kampen squares, i.e. for any commutative diagram as the one below, assuming that h_1 and g_2 are M-morphisms, if the bottom diagram

is a pushout and the back faces are pullbacks then the top diagram is a pushout if and only if the front diagrams are pullbacks.

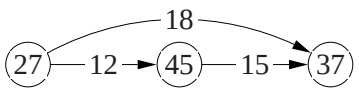


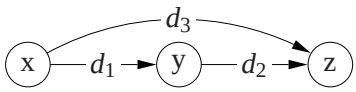
The key idea to show that **AttGraphs** is adhesive HLR is the choice of the right kind of M-morphisms. Actually, **AttGraphs** is not adhesive[1] because it fails to satisfy the van Kampen property for arbitrary monomorphisms.

Theorem 1 **AttGraphs** *is adhesive HLR, with respect to the class of M-morphisms consisting of all monomorphisms* $\langle h_{graph}, h_{alg} \rangle$ *such that* h_{alg} *is an isomorphism.*

3 The category of symbolic graphs

A symbolic graph can be seen as the specification of an attributed graph (or 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. In this sense, 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, below on the right, we can see an example of a very simple symbolic graph and, on the left, the (unique) attributed graph denoted by that symbolic graph.

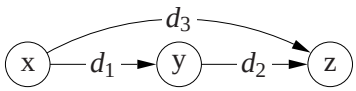




with $(x = 27) \wedge (y = 45) \wedge z = 37$
$\wedge\ (d_1 = 12) \wedge (d_2 = 15) \wedge (d_3 = 18)$

However, as said above, a symbolic graph, in general denotes a class of graphs. For instance, the graph below specifies a class of attributed graphs that includes the graph depicted above on the left, but it also specifies many other graphs.

[1] Roughly speaking, an adhesive category [13] is like an adhesive HLR category, where M is the class of all monomorphisms



with $d_3 \leq d_1 + d_2$

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

Therefore, let us define what is a symbolic graph over a given data algebra.

Definition 7 (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. $L_G = \cup_{s \in S} X_s$, and Φ is a set of first-order Σ-formulas built over the free variables in X and including the elements in D as constants.

Given symbolic graphs $\langle G_1, \Phi_1 \rangle$ and $\langle G_2, \Phi_2 \rangle$ over the same data algebra D, a symbolic graph morphism $h : \langle G_1, \Phi_1 \rangle \to \langle G_2, \Phi_2 \rangle$ is an E-graph morphism $h : G_1 \to G_2$ 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 $h_L(x_1)$.

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

In what follows, to simplify notation, even if it may be considered an abuse of notation, we will write $h(\Phi)$ instead of $h^\#(\Phi)$. Moreover, also for simplicity, we may identify the set of formulas Φ with the formula consisting of the conjunction of all the formulas in Φ, even if that formula may be infinitary in the case where Φ is an infinite set.

Notice that, according to the above definition, given any E-graph G, if $D \models \Phi \Leftrightarrow \Phi'$ then $\langle G, \Phi \rangle$ and $\langle G, \Phi' \rangle$ are isomorphic in $\mathbf{SymbGraphs_D}$.

To show that symbolic graphs are an adhesive HLR category, first, we have to define our notion of M-morphism over symbolic graphs. We consider that M-morphisms are monomorphisms where the formulas constraining the source and target graphs are equivalent (in most cases they will just be the same formula). 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.

Definition 8 (M-morphisms) An *M-morphism* $h : \langle G, \Phi \rangle \to \langle G', \Phi' \rangle$ is a monomorphism such that $L_G \cong L_{G'}$, i.e. h_L is a bijection, and $D \models h(\Phi) \Leftrightarrow \Phi'$.

It is not difficult to see that M-morphisms satisfy the required properties. Then, to define pushouts and pullbacks in $\mathbf{SymbGraphs_D}$ we use pushouts and pullbacks in $\mathbf{E-Graphs}$, respectively. More precisely, the pushout of $\langle G_1, \Phi_1 \rangle \stackrel{h_1}{\leftarrow} \langle G_0, \Phi_0 \rangle \stackrel{h_2}{\to} \langle G_2, \Phi_2 \rangle$ is a graph $\langle G_3, \Phi_3 \rangle$, where $G_1 \stackrel{g_1}{\to} G_3 \stackrel{g_2}{\leftarrow} G_2$ is the pushout of $G_1 \stackrel{h_1}{\leftarrow} G_0 \stackrel{h_1}{\to} G_2$ and Φ_3 is the conjunction of $g_1(\Phi_1)$ and $g_2(\Phi_2)$. The case of pullbacks is similar, but the pullback of $\langle G_1, \Phi_1 \rangle \stackrel{g_1}{\to} \langle G_3, \Phi_3 \rangle \stackrel{g_2}{\leftarrow} \langle G_2, \Phi_2 \rangle$ is the graph $\langle G_0, \Phi_0 \rangle$, where $G_1 \stackrel{h_1}{\leftarrow} G_0 \stackrel{h_2}{\to} G_2$ is the pullback of $G_1 \stackrel{g_1}{\to} G_3 \stackrel{g_2}{\leftarrow} G_2$ and Φ_0 is the

disjunction of $h_1(\Phi_1)$ and $h_2(\Phi_2)$. However, since G_0 may include a strict subset of the variables of Φ_1 and Φ_2, in this case Φ_0 is existentially quantified by the variables not in G_0.

Proposition 1 **SymbGraphs$_\mathbf{D}$** *has pushouts and pullbacks.*

To see that pushouts and pullbacks preserve M-morphisms we just have to do some basic logical deduction. If the diagram below is a pushout and h_1 is an M-morphism then we have to prove that $D \models \Phi_2 \Leftrightarrow (g_2(\Phi_2) \wedge g_1(\Phi_1))$. But, since h_1 is an M-morphism we may consider without loss of generality that h_1 is the equality on variables and $\Phi_0 = \Phi_1$. Moreover, we may also consider without loss of generality that g_2 is also the equality on variables and that h_2 and g_1 coincide when restricted to the variables. As a consequence, what we would need to prove is that $D \models \Phi_2 \Leftrightarrow (\Phi_2 \wedge h_2(\Phi_0))$, since $g_2(\Phi_2) = \Phi_2$ and $h_2(\Phi_0) = g_1(\Phi_1)$. But this is obvious, since we know that $D \models \Phi_2 \Leftarrow h_2(\Phi_0)$. The case of pullbacks is slightly more complex because of the existential quantifiers in Φ_0.

$$\begin{array}{ccc} \langle G_0, \Phi_0 \rangle & \xrightarrow{h_1} & \langle G_1, \Phi_1 \rangle \\ {\scriptstyle h_2}\downarrow & & \downarrow{\scriptstyle g_1} \\ \langle G_2, \Phi_2 \rangle & \xrightarrow[g_2]{} & \langle G_3, \Phi_3 \rangle \end{array}$$

Proposition 2 *Pushouts and pullbacks preserve M-morphisms.*

Finally, to prove the van Kampen property we show that a cube in **SymbGraphs$_\mathbf{D}$** is a van Kampen square if and only if the underlying cube in $\mathbf{E}-$**Graphs** is also a van Kampen square. To do this, again we just need to do some basic logical reasoning. As a consequence we have:

Theorem 2 **SymbGraphs$_\mathbf{D}$** *is adhesive HLR.*

4 Symbolic graphs and attributed graphs

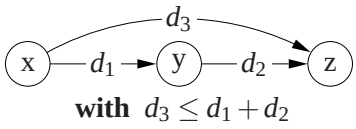
In this section we present the relation between the categories of symbolic and attributed graphs over a given data algebra. On one hand, we will see that every symbolic graph may be seen as denoting a class (a subcategory) of attributed graphs, which may be considered its semantics. On the other hand, we will see that every attributed graph can be represented in a canonical way by a symbolic graph, which means that, for a given data algebra, the category of attributed graphs can be seen as a subcategory of the corresponding category of symbolic graphs.

Definition 9 (Semantics of symbolic graphs) Given a symbolic graph $\langle G, \Phi \rangle$ over a data algebra D, its semantics is a class of attributed graphs defined as follows:

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

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

For example, given the symbolic graph below:



we have that its semantics would include the following attributed graphs:

Conversely, we can identify every attributed graph AG with a *grounded* symbolic graph whose semantics consists only of AG. More precisely a grounded graph is a symbolic graph $\langle G,\Phi\rangle$ that includes a variable x_v for each element v of the data algebra and where the only substitution $\sigma : L_G \to D$ such that $D \models \sigma(\Phi)$ is defined for each variable x_v as $\sigma(x_v) = v$.

Definition 10 (Grounded symbolic graphs) A symbolic graph $\langle G,\Phi\rangle$ over a data algebra D is grounded if

1. L_G includes a variable, which we denote by x_v, for each value $v \in D$, and
2. For every substitution $\sigma : L_G \to D$, such that $D \models \sigma(\Phi)$, we have $\sigma(x_v) = v$, for each variable $x_v \in L_G$.

Moreover, we define $\mathbf{GSymbGraphs_D}$ as the full subcategory of $\mathbf{SymbGraphs_D}$ consisting of all grounded graphs.

Notice that if $\langle G,\Phi\rangle$ is grounded and $\sigma : L_G \to D$ is a substitution such that $D \models \sigma(\Phi)$ then $\sigma^* : G \to \sigma(G)$ is an isomorphism.

It should be obvious that the semantics of a grounded graph includes exactly one attributed graph, and that grounded graphs are closed up to isomorphism. Moreover, we can see that for every attributed graph AG there is a unique grounded symbolic graph (up to isomorphism) $GSG(AG)$ such that $Sem(GSG(AG))$ consists of AG. In particular, the E-graph associated to $GSG(AG)$ is obtained substituting every data value v in a set of labels by a variable x_v, and the set of formulas in the symbolic graph consists of an equation $x_v = v$, for each value v in D.

Definition 11 Given an attributed graph $AG = \langle G,D\rangle$, we define the *grounded symbolic graph associated* to AG, $GSG(AG)$ as the symbolic graph $\langle G',\Phi\rangle$, where:

- The set of labels X of the E-graph G' consists of a variable x_v for each element $v \in D$.
- $G' = f^*(G)$, where $f : D \to X$ is a substitution such that for every $v \in D$, $f(v) = x_v$.
- $\Phi = \{x_v = v \mid v \in D\}$.

Proposition 3

1. *If SG is grounded then Sem(SG) consists exactly of one attributed graph.*

2. *Grounded symbolic graphs are closed up to isomorphism.*

3. *For each attributed graph $AG = \langle G, D \rangle$, $Sem(GSG(AG)) = \{AG\}$.*

Proof.

1. If $SG = \langle G, \Phi \rangle$ is grounded, by definition we know that Sem(SG) is not empty, since Φ is satisfiable in D. Moreover, If $AG1, AG2 \in Sem(\langle G, \Phi \rangle)$ then this means that there are substitutions $\sigma, \sigma' : L_G \to D$ such that $D \models \sigma(\Phi)$ and $D \models \sigma'(\Phi)$. But if $\langle G, \Phi \rangle$ is grounded this means that $L_G = \{x_v \mid v \in D\}$ and for each $v \in D$: $\sigma(x_v) = v$ and $\sigma'(x_v) = v$. But this implies that $\sigma = \sigma'$ and therefore $AG_1 = AG_2$.

2. Let $SG = \langle G, \Phi \rangle$ be a grounded graph and let $SG' = \langle G', \Phi' \rangle$ be isomorphic to SG. This means that there is an E-graph isomorphism $h : G \to G'$ such that $D \models \Phi \Leftrightarrow h(\Phi')$. But this implies that $h_L : L_G \to L_{G'}$ is a bijection and if $\sigma' : L_{G'} \to D$ is a substitution such that $D \models \sigma'(\Phi')$ then $\sigma' \circ h : L_G \to D$ is a substitution such that $D \models \sigma' \circ h(\Phi)$, and this means that for every $v \in D$ $\sigma' \circ h(x_v) = v$. Therefore, if for every $v \in D$ we call y_v the variable $h(x_v)$ then we have that, for each $v \in D$, $\sigma'(y_v) = v$, which means that SG' is grounded.

3. It should be obvious that, by construction, $GSG(AG)$ is grounded and, moreover, $AG \in Sem(GSG(AG))$.

   Now, suppose that $SG_0 = \langle G_0, \Phi_0 \rangle$ is a symbolic graph such that $AG = \langle G, D \rangle \in Sem(SG)$. Let us prove that SG and $GSG(AG) = \langle G', \Phi_{AG} \rangle$ are isomorphic. First of all, we know that $G = \sigma_0^*(G_0)$ for a substitution σ_0 such that $D \models \sigma_0(\Phi_0)$. But, since SG_0 is grounded, σ_0^* is an isomorphism. For similar reasons, we know that $f^* : G \to G'$ is also an isomorphism therefore $f^* \circ \sigma_0^* : G_0 \to G'$ is an E-graph isomorphism. Finally, it is easy to see that $D \models \Phi \Leftrightarrow f \circ \sigma_0(\Phi_0)$. In particular, if σ is a substitution such that $D \models \sigma(\Phi)$ we have to prove that $D \models \sigma \circ f \circ \sigma_0(\Phi_0)$ or, equivalently, that $\sigma \circ f \circ \sigma_0 = \sigma_0$. But this is obvious since, on one hand, by construction, $v \in D$: $f(v) = x_v$, and, on the other hand, we know that for every $v \in D$: $\sigma(x_v) = v$, which means that $f = \sigma^{-1}$. Conversely, if σ is a substitution such that $D \models \sigma \circ f \circ \sigma_0(\Phi_0)$ we can prove similarly that this implies that $D \models \sigma(\Phi)$.

□

It should also be obvious that the encoding of attributed graphs in terms of symbolic graphs defined by *GSG* can be applied to all kinds of attributed graphs, i.e. not only to attributed graphs defined over a data algebra D, but also to term-attributed graphs or to term-attributed graphs defined over a specification *SP*. However, in the latter case, we prefer to define a different encoding which may actually be seen as a variation of *GSG*, that we call its *symbolic representation*, denoted *SR*, and which will be used in the following section. In particular, if G is a term-attributed graph defined over a specification *SP*, we define $SR(G)$ as follows. First, we assume that we have a function that *chooses* a term from every congruence class of terms in T_{SP}. We call this function a *choice function*. Then, the symbolic representation of G would be $\langle G', \Phi' \rangle$, where G' is obtained replacing each label a in G (i.e. each congruence class in T_{SP}) by the variable x_t, where t is the term chosen by the choice function when applied to a, and where Φ' consists of all the equations in *SP* and all the equations $x_t = t$ for each term t returned by the choice function.

Definition 12 Given a specification $SP = (\Sigma \cup X, \Phi)$, such that there is an initial algebra T_{SP} in the category of SP-algebras, we say that $ch : T_{SP} \to T_{\Sigma \cup X}$ is a *choice function* for T_{SP} if for every element $|t'| \in T_{SP}$ if $ch(t) = t'$ then $T_{SP} \models t = t'$, where $|t'|$ denotes the congruence class of t' with respect to the congruence defined by SP.

Given SP, an attributed graph $AG = \langle G, T_{SP}|_\Sigma \rangle$, and a choice function ch for T_{SP} we define the symbolic representation of AG with respect to ch, $SR_{ch}(AG)$ as the symbolic graph $\langle G', \Phi' \rangle$, where:

- The set of labels of the E-graph G' is $X \cup Y$, where Y is disjoint with X and it consists of a variable $y_{ch(a)}$ for each element $a \in T_{SP}$ such that $a \notin \{|x| \mid x \in X\}$.
- $G' = f^*(G)$, where $f : T_{SP} \to Y$ is a substitution such that for every $a \in T_{SP}$, if $a \notin \{|x| \mid x \in X\}$ then $f(a) = y_{ch(a)}$. Otherwise, $f(|x|) = x$.
- $\Phi' = \Phi \cup \{y_{ch(a)} = t \mid y_{ch(a)} \in Y \wedge ch(a) = t\}$.

This means that G' includes as labels the variables in X and a variable y_a for every element a in T_{SP} which is not the congruence class of a variable in X. This means that the substitution f is a bijection. As a consequence, for every attributed graph $AG = \langle G, T_{SP}|_\Sigma \rangle$, if $SR_{ch}(AG) = \langle G', \Phi' \rangle$ then G and G' are isomorphic E-graphs.

It may be noted that we have not stated over which algebra D the symbolic graph $SR_{ch}(AG)$ is defined. The reason is that we may consider that $SR_{ch}(AG)$ is a symbolic graph over any Σ-algebra D. Anyhow, if we consider that $SR_{ch}(AG)$ is defined over $T_{SP}|_\Sigma$ then $SR_{ch}(AG)$ is a grounded graph in $\mathbf{SymbGraphs}_{T_{SP}|_\Sigma}$, i.e. $SR_{ch}(AG)$ is an object in $\mathbf{GSymbGraphs}_{T_{SP}|_\Sigma}$. This means that, following Proposition 3, $SR_{ch}(AG)$ and GSG(AG) are isomorphic graphs in $\mathbf{SymbGraphs}_{T_{SP}|_\Sigma}$.

According to Proposition 3, we can identify each attributed graph with a grounded symbolic graph, and vice versa. Therefore, we may ask whether $\mathbf{AttGraphs}_D$ is isomorphic to $\mathbf{GSymbGraphs}_\mathbf{D}$. The answer is negative since GSG cannot be made injective on morphisms as the following counter-example shows.

Example 1 *Let D be a data algebra consisting of two values of the same sort, which we call a and b. Let AG be an attributed graph having no graph nodes and no graph edges (i.e. the graph structure of AG is empty, which means that it consists only of the label nodes a and b). As a consequence, $GSG(AG) = \langle G, x_a = a \wedge x_b = b \rangle$, where G is an E-graph consisting only of the label nodes x_a and x_b. Now, there are four morphisms, f_1, f_2, f_3 and f_4, from AG to itself:*

- $f_1(a) = a$, $f_1(b) = b$.
- $f_2(a) = a$, $f_2(b) = a$.
- $f_3(a) = b$, $f_3(b) = b$.
- $f_4(a) = b$, $f_4(b) = a$.

However the only morphism from $GSG(AG)$ to itself is the identity. For example we may see that the mapping $g : \{x_a, x_b\} \to \{x_a, x_b\}$, defined $g(x_a) = x_a$, $g(x_b) = x_a$, does not define a

symbolic graph morphism. In particular, if g is a morphism it should hold that $D \models (x_a = a \wedge x_b = b) \Rightarrow g(x_a = a \wedge x_b = b)$. *But this is equivalent to* $D \models (x_a = a \wedge x_b = b) \Rightarrow (x_a = a \wedge x_a = b)$, *which is obviously false.*

The problem in the above counter-example is that we assume that we can define any mapping between the elements of the algebra, while for the variables of the grounded graphs we are forced to map the variable x_v associated to a value to the corresponding variable associated to the same value. This problem disappears if the value algebra is finitely generated. In that case, we know that the only homomorphism of an algebra into itself is the identity causing that morphisms on attributed graphs should be the identity on data values. This means that, if D is finitely generated then the categories $\mathbf{AttGraphs}_D$ and $\mathbf{GSymbGraphs}_\mathbf{D}$ are equivalent. Moreover, this kind of restriction is quite reasonable since, otherwise, the algebra would include values which we cannot refer to. Nevertheless, as we will see in the following section, attributed graph transformation rules are usually defined over non-finitely generated algebras.

Proposition 4 *If* D *is finitely generated then* $\mathbf{AttGraphs}_D$ *and* $\mathbf{GSymbGraphs}_\mathbf{D}$ *are equivalent.*

Proof. First, we will show that GSG can be extended to a functor and, then, that GSG is full, faithful and essentially surjective. Let $f : \langle G_1, D\rangle \to \langle G_2, D\rangle$ be an attributed graph morphism. Since D is finitely generated, f_{alg} is the identity and f_{graph} is an E-graph morphism. Hence, if $f : D \to \mathcal{X}_D$ is a substitution defined for every $v \in D$ as $f(v) = x_v$, and $\Phi = \{x_v = v \mid v \in D\}$ we know that $GSG(\langle G_1, D\rangle) = \langle f(G_1), \Phi\rangle$ to $GSG(\langle G_2, D\rangle) = \langle f(G_2), \Phi\rangle$. We define $GSG(f)$ as follows:

- For every $x \in \{V_G, E_G, E_{NL}, E_{EL}\}$: $GSG(f)_x = f_x$.
- $GSG(f)_L$ is the identity.

Then, it is routine to prove that $GSG(f)$ is indeed a symbolic graph morphism.

To prove that GSG is full, we have to show that if $AG_1 = \langle G_1, D\rangle$ and $AG_2 = \langle G_2, D\rangle$ are two attributed graphs and $h : GSG(AG_1) \to GSG(AG_2)$ is a symbolic graph morphism then there exists an attributed graph morphism $f : AG_1 \to AG_2$ such that $GSG(f) = h$. But it is enough to define f as follows:

- For every $x \in \{V_G, E_G, E_{NL}, E_{EL}\}$: $f_x = h_x$.
- f_{alg} (and, therefore, f_L) is the identity.

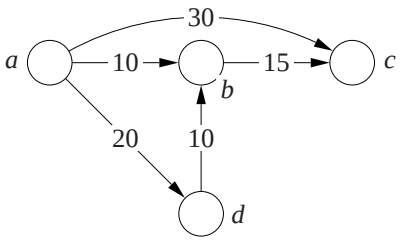
To prove that GSG is faithful we have to show that GSG is injective on morphisms, but this is straightforward by construction. Finally, to prove that GSG is essentially surjective, we have to show that for every grounded graph SG there is another grounded graph SG', which is isomorphic to SG, and an attributed graph AG such that $GSG(AG) = SG'$. But by Prop 3 we know that $Sem(SG)$ is not empty and that if $AG \in Sem(SG)$ satisfies that $GSG(AG) = SG$. □

5 Symbolic graph transformation and attributed graph transformation

In this section we compare attributed graph transformation with symbolic graph transformation. This comparison may seem trivial: if attributed graphs may be seen as a special case of symbolic graphs then we can conclude that attributed graph transformation is a special case of symbolic graph transformation. However things are not so obvious. As we have seen, if the given data algebra D is finitely generated, we can identify attributed graphs over D with grounded symbolic graphs over D. This means, in that case, that if transformation rules are spans of M-morphisms in $\mathbf{AttGraphs}_D$ then these transformation rules can be considered equivalent to spans of M-morphisms in $\mathbf{GSymbGraphs}_\mathbf{D}$, and the application of these rules to a graph AG in $\mathbf{AttGraphs}_D$ is equivalent to the transformation of $GSG(AG)$ by the corresponding rules in $\mathbf{GSymbGraphs}_\mathbf{D}$. The problem is that if the graphs that we want to transform are in $\mathbf{AttGraphs}_D$, usually, the transformation rules will not be spans of M-morphisms in $\mathbf{AttGraphs}_D$, but in $\mathbf{AttGraphs}_{D'}$, where D' is some free algebra over D and, hence, different from D. That is, typically, transformation rules over attributed graphs are defined using term attributed graphs.

In order to compare attributed and symbolic graph transformation we start giving an example of symbolic graph transformation rules and symbolic graph transformation.

Example 2 *Let us suppose that we are dealing with a class of graphs whose edges have an attribute that represents the distance between the source and target nodes. For instance, the graph G_0 below may be an example of a graph in this class:*



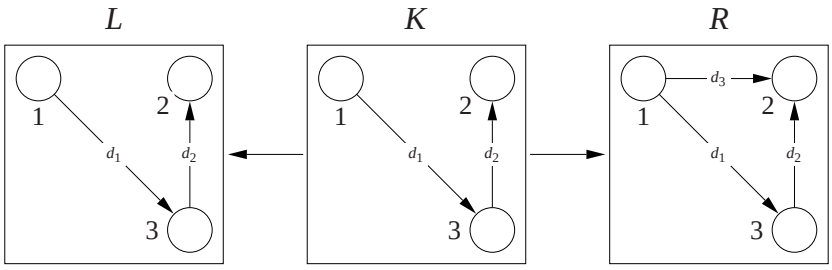
More precisely, let us consider that the underlying algebra in this class of graphs is the algebra D of natural numbers, defined over the signature Σ:

Sorts	$nat, bool$
Opns	$0 : nat$
	$suc : nat \to nat$
	$true, false : bool$
	$+ : nat \times nat \to nat$
	$\leq : nat \times nat \to bool$

We can see the above graph as a grounded symbolic graph. In this case we would need to replace the values which are bound to the edges of the graph by the variables x_{10}, x_{15}, x_{20}, and x_{30}, respectively, and we would need to include the formula $(x_0 = 0) \wedge (x_1 = 1) \wedge (x_2 = 2) \wedge \cdots \wedge (x_{10} = 10) \wedge \cdots \wedge (x_{15} = 15) \wedge \cdots \wedge (x_{20} = 20) \wedge \cdots \wedge (x_{30} = 30) \wedge \ldots$. However, since

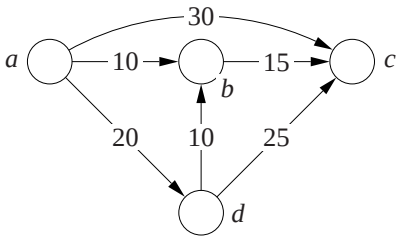
we know that grounded symbolic graphs and attributed graphs are equivalent, for readability, we will directly use the attributed graph representation.

Let us also suppose that we want to compute the distance of the shortest paths between any two nodes. The symbolic graph transformation rule p, depicted below, describes how a new distance can be computed:

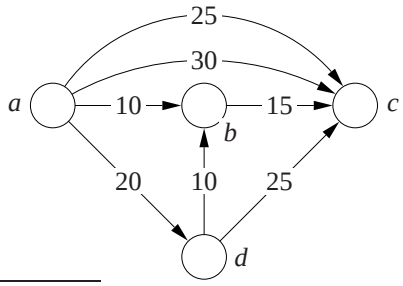


with $d_3 = d_1 + d_2$

If we match 1 *with d,* 2 *with c, and* 3 *with b, and the variables* d_1, d_2*, and* d_3 *with* $10, 15$*, and* 25*, respectively*[2]*, then we can apply this rule to the above graph, because* $25 = 10 + 15$ *holds in D*[3]*, which is the translation of the rule condition, when* d_1, d_2*, and* d_3 *are replaced by their corresponding matches. Therefore, we would transform* G_0 *into* G_1*:*



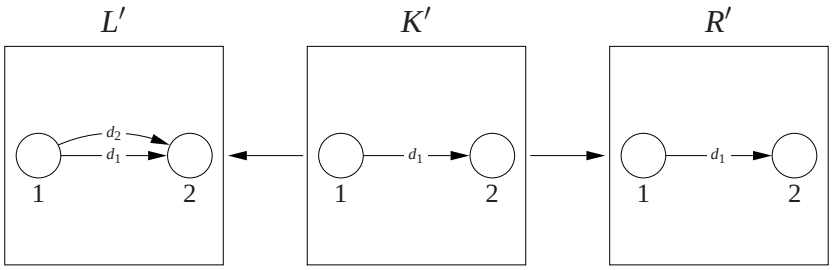
Similarly, matching 1 *with a,* 3 *with b, and* 2 *with c, and the variables* d_1, d_2*, and* d_3 *with* $10, 15$*, and* 25*, respectively, as before, it would be possible to apply the above transformation rule, getting the graph* G_2*:*



[2] To be more precise, we would match d_1, d_2, and d_3 with the variables x_{10}, x_{15}, and x_{25}, respectively.

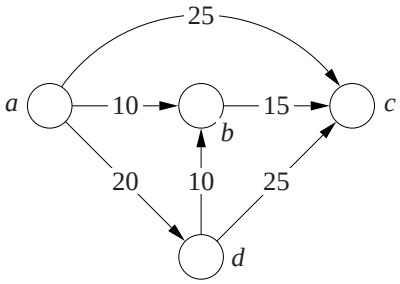
[3] Again, to be more precise, the condition that holds is $(x_{10} = 10) \wedge (x_{15} = 15) \wedge (x_{25} = 25) \wedge \ldots$ implies $(x_{25} = x_{10} + x_{25})$

Now, if we want to get rid of all the edges between two given nodes, except of the one labelled with the smallest distance, we could use the rule p' depicted below[4]*:*

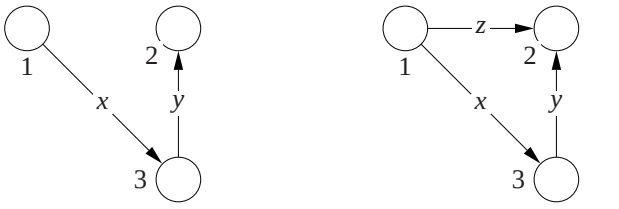


with $(x_1 \leq x_2) = true$

We can apply the above rule to the graph G_2, matching nodes 1 *and* 2 *to nodes a and c, and variables d_1 and d_2 to* 25 *and* 30*, respectively, and also matching the edges bound to the former variables to the corresponding edges in G_2 bound to* 25 *and* 30*. The result would be G_3:*



Remark 1 *Obviously, symbolic graph transformation rules may be applied not only to grounded symbolic graphs, but to arbitrary symbolic graphs. Actually, the fact that the category of symbolic graphs is adhesive HLR ensures that the fundamental theory of graph transformation [4] applies to symbolic graph transformation. However, in practice, it may be impossible to apply a graph transformation rule to an arbitrary symbolic graph, even if seems very reasonable. For instance, we may expect that it should be possible to apply the rule p, depicted above, to the symbolic graph G depicted below on the left, yielding the graph on the right:*



with $x < y$ **with** $x < y \wedge z = x + y$

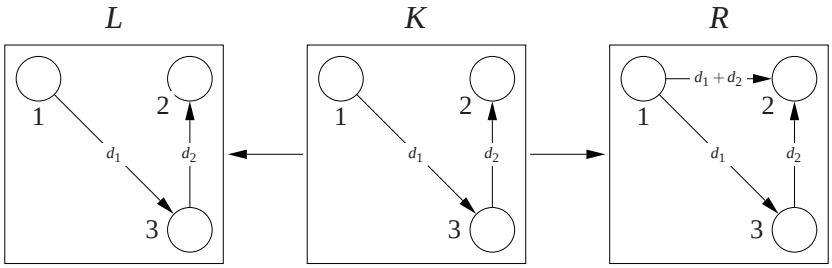
*However, this is not possible. The first problem we find to apply the rule p to G is that if G only includes the variables x and y then the variable d*3 *in the rule could only be matched*

[4] A different alternative for the same problem would be to use some NACs in the first rule to avoid creating more than one edge between any two nodes

to x or y. Moreover, it would be unclear where do the variable z in G' comes from. We can overcome this problem by assuming that G already included the variable z, although it was not explicitly depicted because it was not bounded to any node or edge in G. Actually, we could think that a symbolic graph is supplied with an unlimited number of variables. However this is not the main problem. We cannot apply p to G because $x < y$ does not imply $z = x + y$ in D. This problem is solved in [18], where a new form of symbolic graph transformation, called lazy graph transformation, is studied. More precisely, using lazy graph transformation, G' would be obtained applying p to G in the obvious way.

Now, let us describe how we can define attributed graph transformation rules having a similar effect to the symbolic rules in Example 2.

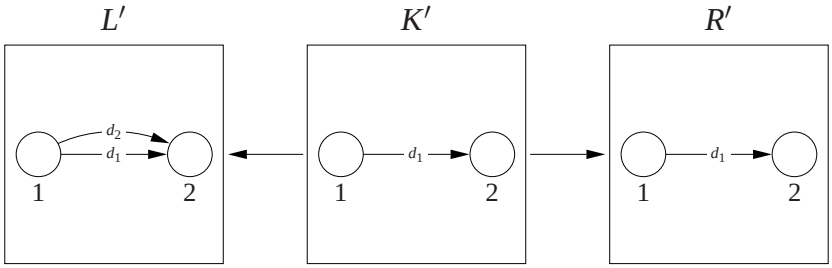
Example 3 *Let us suppose that we want to describe the same procedure as in Example 2 for computing shortest paths, but now using attributed graph transformation rules. In this case, rule p for computing a new distance between two nodes could be:*



The first thing that we should note is that the graphs G_0, G_1, G_2 and G_3 in Example 2 are defined over the algebra D of natural numbers, which means that they include the natural numbers and the booleans as label nodes, even if they are not depicted in the above figures. But the graphs $L, K,$ and R in the transformation rule p are not defined over the same algebra. The reason is that the labels $d_1, d_2,$ or $d_1 + d_2$ are not in D, since they are not natural numbers. The simplest solution that we can use here, is to consider that $L, K,$ and R are term-attributed graphs over the algebra $T_\Sigma(\{d_1, d_2\})$, i.e. the term algebra over the variables d_1 and d_2. These graphs would include as label nodes all the possible Σ-terms over these two variables, even if they are not depicted explicitly. Now, according to the example, when we apply p to G_0 in Example 2 we define a morphism m from L into G matching 1 with d, 2 with c, and 3 with b. Obviously, m would also match the edges in L with the edges in G in the expected way and it would also match d_1 with 10 and d_2 with 15. But this is not all. The match m includes a Σ-homomorphism m_{alg} from $T_\Sigma(\{d_1, d_2\})$ to D matching not only d_1 with 10 and d_2 with 15, but also each possible term over d_1 and d_2 with its corresponding value, after assigning to d_1 and d_2 the values 10 and 15, respectively. This means that, for instance, m would also match $suc(d_1)$ with 11 or $suc(d_1) \leq suc(suc(d_2))$ to **t**. *In particular, m would also match $d_1 + d_2$ with 25, even if $d_1 + d_2$ is not explicitly depicted in L, i.e. we need to compute the resulting value of $d_1 + d_2$ when defining the match, before computing the transformation.*

The fact that the values of the underlying algebra are considered (label) nodes of the attributed

graphs, together with the fact that match morphisms must be homomorphisms for the algebra part, allow us to do some kind of conditional graph transformation without using a negative application condition (NAC) [3, 9], but using term-attributed graphs over a given specification, as discussed in Section 2.3. For example, we can have an attributed graph transformation rule p'', similar to rule p' in Example 2, for deleting all edges between two nodes except the one labelled with the shortest distance. In particular, p'', could be the rule below:



when defined over the algebra $T_{SP}|_\Sigma$, where SP is defined:

$$
\begin{array}{rl}
SP = \textbf{Sorts} & nat, bool \\
\textbf{Opns} & 0 : nat \\
 & d_1, d_2 : nat \\
 & suc : nat \to nat \\
 & true, false : bool \\
 & + : nat \times nat \to nat \\
 & \leq : nat \times nat \to bool \\
\textbf{Axms} & (d_1 \leq d_2) = true
\end{array}
$$

For instance the application of p'' to the graph G_2 in Example 2 matching node 1 to node a and node 2 to node c would necessarily match d_1 to 25 and d_2 to 30 yielding the graph G_3, also in Example 2.

Therefore, we can consider that, in a transformation system for attributed graphs over a Σ-algebra D, each rule r is a span on the category $\textbf{AttGraphs}_{D_r}$, where $D_r = T_{SP_r}|_\Sigma$ and SP_r is a specification $SP_r = (\Sigma_r, \Phi_r)$, such that $\Sigma_r = \Sigma \cup X$ and Φ_r is a set of Σ_r-equations or conditional equations, since this ensures the existence of initial algebras. Under this assumption, we may see that attributed graph transformation systems can be seen as a special case of symbolic graph transformation system. The idea is that every rule r as above can be represented by a symbolic transformation rule r', using the symbolic representation of the graphs in r. More precisely:

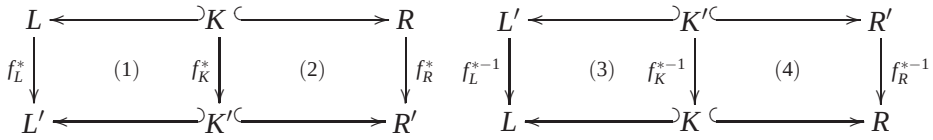
Definition 13 Given a specification $SP = (\Sigma \cup X, \Phi)$, such that there is an initial algebra T_{SP}, a choice function ch for T_{SP}, and an attributed graph transformation rule $r = (\langle L, T_{SP}|_\Sigma\rangle \hookleftarrow \langle K, T_{SP}|_\Sigma\rangle \hookrightarrow \langle R, T_{SP}|_\Sigma\rangle)$, we define the symbolic representation of r with respect to ch, $SR_{ch}(r)$ as the symbolic transformation rule $r' = \langle L' \hookleftarrow K' \hookrightarrow R', \Phi'\rangle$ where:

- $\langle L', \Phi'\rangle = SR_{ch}(\langle L, T_{SP}|_\Sigma\rangle)$,

- $\langle K', \Phi' \rangle = SR_{ch}(\langle K, T_{SP}|_\Sigma \rangle)$,
- $\langle R', \Phi' \rangle = SR_{ch}(\langle R, T_{SP}|_\Sigma \rangle)$.

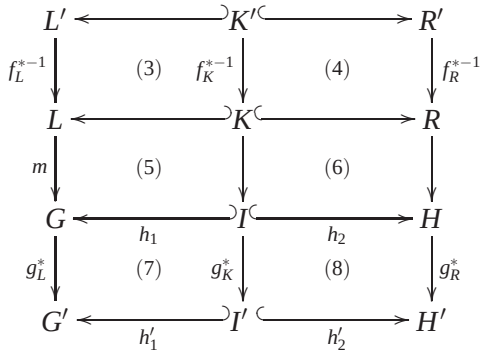
Remark 2 *The inclusions $K' \subseteq L'$ and $K' \subseteq R'$ are a consequence, first, of the fact that we assume that $K \subseteq L$ and $K \subseteq R$[5]; second, of the definition of how a label substitution is applied to an E-graph; and third of the fact that the use of the choice function ensures that the substitution of values in $T_{SP}|_\Sigma$ by variables in Y is the same on the three graphs L', R', and K'.*

Moreover, it may be noticed that, by definition of the choice functions, diagrams (1), (2), (3), and (4) below are pushouts, where f_L^*, f_K^*, and f_R^* are, respectively, the isomorphisms relating L, K, and R with L', K', and R', and where f_L^{*-1}, f_K^{*-1}, and f_R^{*-1} are their inverses:



Theorem 3 *Let $r = (AL \leftarrow AK \rightarrow AR)$ be an attributed graph transformation rule, where AL, AK, and AR are attributed graphs over $T_{SP}|_\Sigma$ and $SP = (\Sigma \cup X, \Phi)$, let ch be a choice function for T_{SP}, and let $r' = SR_{ch}(r)$, then for every attributed graph $AG = \langle G, D \rangle$ and every morphism $m : AL \rightarrow AG$ there is a morphism $m' : \langle SR_{ch}(AL), \Phi' \rangle \rightarrow GSG(AG)$ such that AG is transformed into AH by r with match m, i.e. $AG \Rightarrow_r^m AH$, if and only if $GSG(AG) \Rightarrow_{r'}^{m'} GSG(AH)$. Conversely, for every morphism $m' : \langle SR_{ch}(AL), \Phi' \rangle \rightarrow GSG(AG)$ there is a morphism $m : AL \rightarrow AG$ such that $GSG(AG) \Rightarrow_{r'}^{m'} GSG(AH)$ if and only if $AG \Rightarrow_r^m AH$.*

Proof. Let us assume that $AL = \langle L, T_{SP}|_\Sigma \rangle, AK = \langle K, T_{SP}|_\Sigma \rangle, AR = \langle R, T_{SP}|_\Sigma \rangle$, and $r' = \langle L' \hookleftarrow K' \hookrightarrow R', \Phi' \rangle$, and let us consider the following diagram in $\mathbf{E} - \mathbf{Graphs}$:



where (5) and (6) are the pushouts defining the application of r to AG with match m, $\langle G', \Psi \rangle = GSG(AG), \langle I', \Psi \rangle = GSG(\langle I, T_{SP}|_\Sigma \rangle)$, $\langle H', \Psi \rangle = GSG(\langle H, T_{SP}|_\Sigma \rangle)$, g_L^*, g_K^*, g_R^* are, respectively, the isomorphisms relating G, I, H with G', I', H', and finally the morphisms h_i', for $i = 1, 2$ are

[5] Since the morphisms relating L and R are M-morphisms, without loss of generality, we may assume that they are the identity on the algebra part and an inclusion on the graph part

defined as follows. For every element $e \in I'$ which is not a label, $h'_i(e) = h_i(e)$, and for every label e, $h'_i(e) = e$. It is routine to see that, by definition of GSG and as a consequence of the fact that h_1 and h_2 are M-morphisms, h'_1 and h'_2 are morphisms and, moreover, diagrams (7) and (8) not only commute but are pushouts. Therefore, since we know that diagrams (3) and (4) are pushouts, then diagrams (3)+(5)+(7) and (4)+(6)+(8) are also pushouts. Therefore, if we define $m' = g^*_L \circ m \circ f^{*-1}_L$ and we show that m' is a morphism in $\mathbf{SymbGraphs_D}$ the first part of the theorem will be proved. Therefore, we have to prove that $D \models \Psi \Rightarrow m'(\Phi')$.

We now that

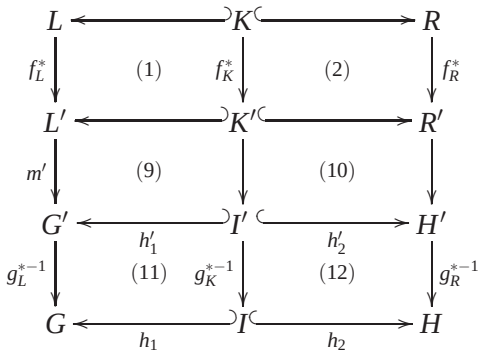
$$\Phi' = \Phi \cup \{y_a = t \mid y_a \in Y \wedge ch(a) = t\} \text{ and } \Psi = \{x_v = v \mid v \in D\}$$

We also know that the only substitution σ such that $D \models \sigma(\Psi)$ is defined $\forall v \in D : \sigma(x_v) = v$. Therefore, we have to show that $D \models \sigma(m'(\Phi'))$ or, equivalently, $D \models \sigma(m'(\Phi))$ and $D \models \sigma(m'(\{y_e = t \mid y_e \in Y \wedge ch(e) = t\}))$. Finally, by definition, on the one hand, we have that for every $a \in T_{SP}|_\Sigma$ we have $m'(y_a) = x_v$, where $m(a) = v$, which means that $\sigma(m'(y_a)) = m(a)$, and on the other hand, for each $x \in X$, $\sigma(m'(x)) = m(x)$.

Now, let $t_1 = t_2$ be an equation in Φ. Since m_{alg} is a Σ-homomorphism and T_{SP} satisfies this equation, we have that $D \models m(t_1) = m(t_2)$, implying $D \models \sigma(m'(t_1)) = \sigma(m'(t_2))$.

Let $y_a = t$ and $ch(a) = t$. Then, on the one hand, we have that $\sigma(m'(y_a)) = m(a)$ and, on the other, since $ch(a) = t$ we have that in $a = |t|$, implying $\sigma(m'(t)) = m(a)$. Therefore, $\sigma(m'(y_a)) = \sigma(m'(t))$.

The proof of the second part of the theorem is similar to the proof of the first part. We only have to consider the diagram below:



□

The theorem above shows that attributed graph transformation can be seen as a special case of symbolic graph transformation. One may wonder whether both kind of transformations can be considered equivalent in the sense that every symbolic graph transformation rule r can be coded into an attributed graph transformation rule r' such that the application of r to a grounded graph produces the same effect as the application of r' to the corresponding attributed graph. The answer is negative as the counter-example below shows, which means that symbolic transformation rules have more definitional power than attributed graph transformation rules.

Example 4 *Let us suppose that the following symbolic graph SG is the left-hand side of a symbolic graph transformation rule r:*

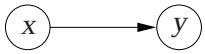
with $(x = 0) \vee (y = 0) \vee (x = y)$

where the signature of the data domain is:

Sorts *nat*
Opns $0 : nat$
$suc : nat \rightarrow nat$

and where the given data algebra D is the algebra of natural numbers. This means that, if r could be represented by an attributed graph transformation rule r', then r' would include as a left-hand side an attributed graph AG like:

where a_1 and a_2 are elements of some Σ-algebra A. Moreover, there should exist a match m from SG into any grounded symbolic graph SG' if and only if there exists an equivalent match m' from AG into the corresponding attributed graph. In particular, given the symbolic graph:



with $(x = n_1) \wedge (y = n_2)$

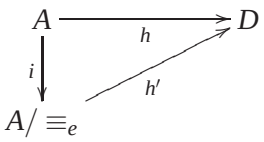
where n_1 and n_2 are two natural numbers, there should exist a homomorphism from A to D mapping a_1 to n_1 and a_2 to n_2 if and only if $n_1 = 0$ or $n_2 = 0$ or $n_1 = n_2$. Let us see that this is impossible.

In particular, first, we will see that a_1 and a_2 cannot be the value of a ground term (i.e. they cannot be obtained applying the suc operation some number of times to 0. Then, we will see that neither a_1 nor a_2 can be obtained applying any number of times the suc operation to some other value in the algebra. But, then this means that we can match a_1 and a_2 to any pair of natural numbers, which implies that for instance we can match a_1 to 2 and a_2 to 1, violating the condition in the symbolic graph.

First, we may notice that we may assume without loss of generality that A satisfies the axiom:

$$e : suc(x) = suc(y) \Rightarrow x = y$$

since for every homomorphism $h : A \rightarrow D$ there is a unique homomorphism $h' : A/\equiv_e \rightarrow D$ such that the diagram below commutes:



where $\equiv_e$ is the congruence on A defined by the axiom e and i is the canonical homomorphism from A into its quotient, mapping every element from A into its congruence class. Vice versa, for every homomorphism $h' : A/\equiv_e \to D$ there is a unique homomorphism $h : A \to D$ such that the diagram above commutes. Finally, we know that $h(a_1) = 0$ or $h(a_2) = 0$ or $h(a_1) = h(a_2)$ if and only if $h'(|a_1|) = 0$ or $h'(|a_2|) = 0$ or $h'(|a_1|) = h'(|a_2|)$.

Therefore, let us assume that A satisfies the above axiom. Now, let us notice that neither a_1 nor a_2 can be the value of some ground term $suc^n(0)$, for $0 \leq n$. The reason is that, otherwise, if $n_1 = n_2 \neq n$ the match would be impossible. We can also see that it is not possible that a_1 is the value of some term $suc^n(a_0)$, for $1 \leq n$ and any $a_0 \in A$. Otherwise, if $n_1 = 0$ the match would be impossible, against the assumption, since if the match m' satisfies $m'(a_0) = n_0$ then $m'(a_1)$ would be $n + n_0$. For similar reasons, we know that it is not possible that a_2 is the value of some term $suc^n(a_0)$, for $1 \leq n$ and any $a_0 \in A$. As a consequence, we can see that

$$A' = A \setminus \{a \mid (a = suc_A^n(a_1)) \vee (a = suc_A^n(a_2)) \text{ for some } n \geq 0\}$$

is a subalgebra of A. Suppose, otherwise, that A' is not a subalgebra of A. This would mean that there is an element $a' \in A'$ such that $suc_A(a') \in A \setminus A'$. But this would mean that $suc_A(a') = suc_A^n(a_1)$ or $suc_A(a') = suc_A^n(a_2)$. But this would imply one of the following cases:

1. *$suc_A(a') = a_1$ or $suc_A(a') = a_2$. These two cases are impossible according to what we have proved above.*

2. *$suc_A(a') = suc_A^n(a_1)$ or $suc_A(a') = suc_A^n(a_2)$ for $n \geq 1$. However, since A is assumed to satisfy the axiom e, this means that $a' = suc_A^{n-1}(a_1)$ or $a' = suc_A^{n-1}(a_2)$, implying that $a' \in A \setminus A'$, against the hypothesis.*

As a consequence of the previous facts we know that every homomorphism $h : A \to D$ is uniquely determined by a homomorphism $h' : A' \to D$ and by the values of $h(a_1)$ and $h(a_2)$, in the sense that given h', there is a unique h extending h' satisfying $h(a_1) = n_1$ and $h(a_2) = n_2$, for any $n_1, n_2 \in D$, and vice versa. But this implies that there is a morphism $m' : A \to D$ satisfying $m'(a_1) \neq m'(a_2)$.

In general, a symbolic transformation rule $r' = \langle L' \hookleftarrow K' \hookrightarrow R', \Phi' \rangle$ over a Σ-algebra D can be simulated by an attributed graph transformation rule $r = (AL \leftarrow AK \rightarrow AR)$ over a Σ-algebra A, if the specification SP, whose signature is Σ plus the labels in r' (considered as constants), and whose set of axioms is Φ', has an initial algebra T_{SP}. The problem in the previous counter-example is that the associated specification has no initial algebra. In particular, to ensure the existence of initial algebras Φ' should include only equations and conditional equations.

6 Related work and conclusion

In this paper we have presented a new approach to deal with attributed graphs based on the new notion of symbolic graphs, showing that the new category is adhesive HLR, which means that it is adequate to define graph transformation.

As far as we know, there are essentially three kinds of approaches to define attributed graphs and attributed graph transformation. First, we have the approaches [10, 7] where an attributed graph is a pair (G,D) consisting of a graph G and a data algebra D whose values are nodes in G. Second, we have the approaches [12, 1] where attributed graphs are seen as algebras over a given signature *ASIG*, where *ASIG* is the union of two signatures *ASSIG*, the graph signature and *DSIG*, the data signature, that overlap in the value sorts. In particular, *ASSIG* may be seen as a representation of the graph part of an attributed graph. In [2] these two approaches are compared showing that they are, up to a certain point, equivalent. Finally, we have the approach [19] based on the use of labelled graphs to represent attributed graphs, and of rule schemas 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. However, that approach has also some drawbacks that are briefly discussed in the introduction.

However, a fundamental theory of graph transformation has been formulated only for [7], as a consequence of its characterization as an adhesive HLR category (for more detail see [4]). For this reason, in this paper we have essentially used that approach to study it in connection with our approach based on symbolic graphs.

As we have seen, our approach can be considered an abstract version of [7], since we work at the specification level, rather than dealing directly with algebras to define the attributes. However, as we have shown, it has more expressive power than [7] for the definition of graph transformation rules. In addition to the expressive power, using symbolic attributed graphs has some other advantages. For instance, in [15] working with symbolic attributed graphs simplifies certain kinds of operations defined on transformation rules. For example, this is the case of the operation that, given two transformation rules r_1 and r_2, where r_1 is a subrule of r_2, yields a rule r_3 that computes the remainder of r_2 with respect to r_1, i.e. what has not been computed by r_1 but is computed by r_2. In particular, when working with symbolic graphs the attribute conditions of r_3 are just a simple combination of the attribute conditions of r_1 and r_2. However, if we would have worked with attributed graphs, computing the attributes for r_3 may involve some complex equation solving.

Moreover, we think that there are further aspects related to symbolic graph transformations that deserve some further study. In particular, using logical conditions to specify the attributes of a graph may allow us to postpone finding the solution to attribute constraints when performing graph transformation. This can make attributed graph transformation more efficient. In addition, a generalization of this idea would allow us to define a certain form of narrowing that may be useful in connection to several kind of problems.

Bibliography

[1] M. Berthold, I. Fischer, M.Koch: Attributed Graph Transformation with Partial Attribution. Technical Report 2000-2 (2000).

[2] H. Ehrig: Attributed Graphs and Typing: Relationship between Different Representations. *Bulletin of the EATCS* 82: 175–190 (2004)

[3] H. Ehrig, A. Habel: Graph Grammars with Application Conditions. In *The Book of L* (Grzegorz Rozenberg and Arto Salomaa, Eds.), Springer (1986), 87–100.

[4] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, Gabriele Taentzer: *Fundamentals of Algebraic Graph Transformation*, Springer (2006).

[5] Hartmut Ehrig, Annegret Habel, Julia Padberg, Ulrike Prange: Adhesive High-Level Replacement Categories and Systems. In *Graph Transformations, Second International Conference, ICGT 2004*. Springer Lecture Notes in Computer Science 3256 (2004),144–160.

[6] H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specifications 1: Equations and Initial Semantics*. Vol. 6 of EATCS Monographs on Theoretical Computer Science. Springer, 1985.

[7] H. Ehrig, U. Prange, G. Taentzer: Fundamental Theory for Typed Attributed Graph Transformation. In *Graph Transformations, Second International Conference, ICGT 2004*. Springer Lecture Notes in Computer Science 3256 (2004), 161–177.

[8] Esther Guerra, Juan de Lara, Fernando Orejas: Pattern-Based Model-to-Model Transformation: Handling Attribute Conditions. In *Theory and Practice of Model Transformations, Second International Conference, ICMT 2009*, Richard F. Paige (Ed.), Springer Lecture Notes in Computer Science 5563 (2009), pp. 83–99.

[9] A. Habel, R. Heckel, G. Taentzer: Graph Grammars with Negative Application Conditions. Fundam. Inform. 26(3/4): 287–313 (1996).

[10] R. Heckel, J. Küster, G. Taentzer: Towards Automatic Translation of UML Models into Semantic Domains. In Proc. *APPLIGRAPH Workshop on Applied Graph Transformation* 2002, pp. 11–22.

[11] J. Jaffar, M. Maher, K. Marriot, and P. Stukey. The semantics of constraint logic programs. *The Journal of Logic Programming*, (37):1–46, 1998.

[12] M. Löwe, M. Korff, A. Wagner: An Algebraic Framework for the Transformation of Attributed Graphs. In *Term Graph Rewriting: Theory and Practice*. John Wiley and Sons Ltd. (1993) 185–199.

[13] S. Lack, P. Sobocinski: Adhesive Categories. In *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004* (Igor Walukiewicz, Ed.), Springer Lecture Notes in Computer Science 2987 (2004), 273–288.

[14] P. Lucio, F. Orejas, E. Pasarella and E. Pino. A Functorial Framework for Constraint Normal Logic Programming. In *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, Kokichi Futatsugi, Jean-Pierre Jouannaud, José Meseguer (Eds.), Springer-Verlag Lecture Notes in Computer Science 4060 (2006), 555–577.

[15] M. Naeem, R. Heckel, F. Orejas, F. Hermann, Incremental Service Composition based on Partial Matching of Visual Contracts. In *Fundamental Approaches to Software Engineering*,

13th International Conference, FASE 2010, Springer Lecture Notes in Computer Science 6013 (2010), 123–138.

[16] F. Orejas: Attributed Graph Constraints. In *Graph Transformations, 4th International Conference, ICGT 2008* (Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, Gabriele TaentzerEds.), Springer Lecture Notes in Computer Science 5214 (2008): 274–288.

[17] F. Orejas: Attributed Graph Constraints for Attributed Graph Transformation. *Journal of Symbolic Computation*. To appear.

[18] F. Orejas, L. Lambers: Delaying Constraint Solving in Symbolic Graph Transformation. To appear in *ICGT 2010*.

[19] D. Plump, S. Steinert: Towards Graph Programs for Graph Algorithms. In *Graph Transformations, Second International Conference, ICGT 2004*. Springer Lecture Notes in Computer Science 3256 (2004), 128–143.

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