Symbolic graphs for attributed graph constraints

Fernando Orejas

Dpt. L.S.I., Universitat Politècnica de Catalunya, Campus Nord, Mòdul Omega, Jordi Girona 1-3, E-08034 Barcelona, Spain

ARTICLE INFO

Article history:
Available online 25 September 2010

Keywords:
Attributed graphs
Symbolic graphs
Graph constraints

ABSTRACT

In this paper we present a new class of graphs, called symbolic graphs, to define a new class of constraints on attributed graphs. In particular, in the first part of the paper, we study the category of symbolic graphs showing that it satisfies some properties, which are the basis for the work that we present in the second part of the paper, where we study how to reason with attributed graph constraints. More precisely, we define a set of inference rules, which are the instantiation of the inference rules defined in a previous paper, for reasoning about constraints on standard graphs, showing their soundness and (weak) completeness. Moreover, the proof of soundness and completeness is also an instantiation of the corresponding proof for standard graph constraints, using the categorical properties studied in the first part of the paper. Finally, we show that adding a new inference rule makes our system sound and strongly complete.

1. Introduction

Research in the area of graph transformation started forty years ago. In these years a rich theory was developed to support an increasing number of applications. For a number of years, this theory was associated to some given classes of graphs, for instance, standard directed graphs or typed directed graphs. This meant that if someone was interested in a slightly different kind of graph, for instance some kind of undirected graphs, then the whole theory would need to be reformulated. Fortunately, Lack and Sobocinski (2005) and Ehrig et al. (2006b) provided the basis for a general theory of transformation of structures (Ehrig et al., 2006b) that would apply to any class of graphical structure, as long as they form an adhesive HLR category.

© 2010 Elsevier Ltd. All rights reserved.
Graph constraints were introduced in the area of graph transformation, in connection with the notion of (negative) application conditions, as a form to limit the applicability of transformation rules (Ehrig and Habel, 1986; Habel et al., 1996; Heckel and Wagner, 1995). As a consequence, most research on these notions has been related to their role in graph transformation. However, less attention has been paid to graph constraints as a modeling or specification technique (Koch et al., 2005; Orejas et al., 2009; de Lara and Guerra, 2008). In this context, in Orejas et al. (2008) and Orejas et al. (2009), we presented inference rules to reason about constraints, for instance for checking the consistency of a given set of graph constraints, proving that these rules are sound and complete.

The results in Orejas et al. (2009) apply, in principle, just to standard graphs. However, trying to be as general as possible, all the constructions and proofs were presented in terms of a small number of categorical properties that are satisfied by the category of graphs and by many other categories. Unfortunately, the category of (typed) attributed graphs, as defined in Ehrig et al. (2006b), does not satisfy these properties in a straightforward way. One of the problems is related with the fact that the values of the underlying data algebra, which are used in the attributes, are seen as (a special kind of) nodes of the graphs. As a consequence, graphs are usually infinite, even if the underlying graph structure is finite. This may cause that the proof rules introduced in Orejas et al. (2009) for reasoning with graph constraints, when dealing with attributed graphs as presented in Ehrig et al. (2006b), may allow us to infer infinitary formulas, making the logic incomplete. These issues are discussed in more detail in Section 5. In addition, using that category of attributed graphs, it was not obvious how one could express conditions on the attributes of a graph.

To avoid these problems, in Orejas (2008) we introduced a new notion of attributed graph constraint in terms of a standard graph constraint, to describe a given condition on the graph part of an attributed graph, together with a formula describing a condition on the values of the attributes. The connection between the graph part of these constraints and the corresponding formulas is given by the free variables in the formulas which are labels in the graph constraint. Then we provided an ad hoc extension of the proof system given in Orejas et al. (2008, 2009) and proved that this extension was also sound and complete for this class of constraints.

However, we were not satisfied by the results in Orejas (2008). On the one hand, the kind of constraints that we were considering in Orejas (2008) were not as general as the constraints considered in Orejas et al. (2008, 2009). The reason was that the general case considered in Orejas et al. (2008, 2009) was already quite involved, so we thought that enriching the constraints with the attribute conditions was a bit too complex. On the other hand, considering that the results in Orejas et al. (2008, 2009) were very generic, the ad hoc nature of the new results was deceiving.

In this paper, using the same intuitions about attributed graph constraints, we follow a more systematic approach. We start by introducing a new formulation for attributed graphs, called symbolic graphs, to be used as a basis for our definition of attributed graph constraints. In particular, we consider that a symbolic graph is an E-graph (Ehrig et al., 2006b), whose labels are variables, together with a set of conditions or formulas that constrain the possible values of these variables. Then, as a consequence of our definition of the category of attributed graphs, constraints in this category almost coincide with the ad hoc constraints defined in Orejas (2008) (actually, the new constraints are more general). Moreover, we prove that our category of attributed graphs satisfies the properties that are needed in Orejas et al. (2009) to prove the soundness and completeness of the given inference rules. As a consequence, all the results in Orejas et al. (2009) can also be applied to this class of attributed graph constraints. This means that the inference rules in Orejas et al. (2009) are also sound and complete for the new class of constraints. Unfortunately, as we see in Section 4.1 the completeness result obtained is not as strong as needed. However, adding a new inference rule provides a complete inference system in the required sense.

The paper is organized as follows. In the second section we present the new category of attributed graphs and we show that it satisfies a number of properties which are important for our purposes. In Section 3 we introduce attributed graph constraints and present a small example to motivate their use in connection with visual modelling or website specification. In Section 4 we instantiate the inference rules in Orejas et al. (2009) to our class of graph constraints and show how we can transfer the results of that paper to prove their soundness and completeness. Moreover, we present the additional rule to show completeness in a strong sense. In Section 5 we review related work. Then, in the conclusion we...
briefly discuss the results presented. Finally, to ease the reading of the paper, the proofs of our results have been moved to an Appendix.

2. The category of symbolic graphs

In this section we present a new class of attributed graphs, which we call symbolic graphs and we show that they form a category that satisfies the properties needed to ensure the soundness and completeness of our logic of constraints. First, we present the notion of E-graphs (Ehrig et al., 2006b), as a kind of labelled graph, which is needed to introduce symbolic graphs, then we present our category of symbolic graphs, and finally we show the properties needed.

2.1. E-graphs

E-graphs are introduced in Ehrig et al. (2006b) also as an intermediate step to 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 from labelled graphs is that, in the latter case, 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 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 $L = (V, L, E, E_{NL}, E_{EL}, \{s_j, t_j\}_{j \in \{G, NL, EL\}})$ consisting of:

- $V$ and $L$, which are the sets of graph nodes and of label nodes, respectively.
- $E, E_{NL}$, and $E_{EL}$, which are the sets of graph edges, node label edges, and edge label edges, respectively.

and the source and target functions:

- $s_G : E \rightarrow V$ and $t_G : E \rightarrow V$
- $s_{NL} : E_{NL} \rightarrow V$ and $t_{NL} : E_{NL} \rightarrow L$
- $s_{EL} : E_{EL} \rightarrow E$ and $t_{EL} : E_{EL} \rightarrow L$.

Given E-graphs $G$ and $G'$, an E-graph morphism $f : G \rightarrow G'$ is a tuple, $(f_V : V \rightarrow V', f_L : L \rightarrow L', f_{E_{NL}} : E_{NL} \rightarrow E'_{NL}, f_{E_{EL}} : E_{EL} \rightarrow E'_{EL})$ such that $f$ commutes with all the source and target functions.

2.2. Symbolic graphs

Usually, an attributed graph is considered to be some kind of labelled graph whose labels are values of a given data domain. The idea of our approach is to generalize this notion by considering that attributed graphs are E-graphs, whose labels are variables, together with a set of conditions or formulas that constrain the possible values of these variables. We call these graphs symbolic graphs. For instance, in Fig. 1 we can see on the left the intuitive presentation of a small attributed graph having some values as attributes on its nodes and edges, and, on the right, the representation of the same graph in our approach.

Since we allow any arbitrary formula to constrain the values of the graph, according to our notion, a symbolic graph can be seen as representing a class of attributed graphs. For instance, the graph in Fig. 2 represents a class including a graph for each possible solution of $D_3 \leq D_1 + D_2$, where $+$ and $\leq$ are assumed to be, respectively, an operation and a predicate in the given data domain. In particular, this class may be empty, if the associated condition is unsatisfiable.
In this paper, we use first-order logic with equality to express the conditions on the attributes of a given graph. However, we could also restrict or extend the kind of formulas considered.

There are two standard ways of dealing with the data domain. Following a semantic approach, we may consider that the data domain is given by some predefined algebra over a given data signature. In this context, our main concern is to know if some formulas are satisfied or not by that algebra. The other possibility is to follow a syntactic approach, considering that the data domain is defined by some specification consisting of a signature (the signature of the data domain) and a set of axioms. In this other context, our main concern is to know if some formulas are logical consequences of the specification or if they are satisfiable. In this paper we have chosen the latter approach. The reason for this choice is the kind of results that we want to obtain. More precisely, if a logic is defined as an extension of a given model, trying to prove its completeness may be hopeless. The Gödel incompleteness theorem, or the non-compactness of the set of formulas satisfied by the given model\(^2\) typically make it impossible to prove the completeness of this kind of logic when considering an arbitrary model. On the other hand, proving completeness in the context of the syntactic approach shows that our method may be considered complete when reasoning about the graph part of our logic, independently of the difficulties posed by the data domain.

**Definition 2 (Specification of the Data Domain).** The specification of the data domain, \(\mathcal{S}_\mathcal{D}\), consists of a signature, \(\Sigma_\mathcal{D}\), and a set of first-order axioms \(\mathcal{Ax}_\mathcal{D}\).

Then, a symbolic graph over \(\mathcal{S}_\mathcal{D}\) is an E-graph over a given set of variables \(X\) together with a set of first-order formulas.

**Definition 3 (Symbolic Graphs and Morphisms).** A symbolic graph over \(\mathcal{S}_\mathcal{D}\) is a pair \(\langle G, \Phi \rangle\), where \(G\) is an E-graph over a set of variables \(X\), i.e. \(L_G = X\), and \(\Phi\) is a set of first-order formulas with free variables in \(X\).

Given \(\langle G_1, \Phi_1 \rangle\) and \(\langle G_2, \Phi_2 \rangle\), a 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 \(\mathcal{S}_\mathcal{D} \models \Phi_2 \Rightarrow h^* (\Phi_1)\), where \(h^* (\Phi_1)\) is the set of formulas obtained when replacing in \(\Phi_1\) every variable \(x_1\) in the set of labels of \(G_1\) by \(h_L(x_1)\).

Symbolic graphs over \(\mathcal{S}_\mathcal{D}\) together with their morphisms form the category \(\text{SymbGraphs}_\mathcal{D}\).

The motivation for our definition of symbolic graph morphisms is based on their use in our context. Roughly speaking, we see symbolic graphs as patterns that may match an attributed graph or, in general, another symbolic graph, and we see (mono)morphisms as the corresponding matchings. In this context, if \(h : AG_1 \rightarrow AG_2\) is a symbolic graph monomorphism this should mean that \(h\) embeds the graph part of \(AG_1\) in the graph part of \(AG_2\), i.e. \(h\) is an E-graph monomorphism, and the formulas in \(AG_2\) “satisfy” the formulas in \(AG_1\) (up to the variable renaming defined by \(h\)). For instance, the identity E-graph morphism from the graph \(AG_1\) in Fig. 2 to the graph on the right of Fig. 1 is a symbolic graph morphism, because \(D_1 = 12, D_2 = 15, D_3 = 18\) “satisfy” that \(D_3 \leq D_1 + D_2\). But, formally, this means\(^2\) an algebra may not satisfy an infinite set of formulas, but it may satisfy all its finite subsets.
saying that \( SP_D \models \Phi_2 \Rightarrow h^\#(\Phi_1) \). Similarly, let us suppose that \( AG_3 = (G_3, \Phi_3) \) is a symbolic graph, where \( G_3 = G_1 \) and \( \Phi_3 = \{D_3 < D_1 + D_2\} \). Then the identity morphism from the graph \( G_1 \) to \( G_3 \) is a symbolic morphism from \( AG_1 \) to \( AG_3 \), since \( D_3 < D_1 + D_2 \) “satisfies” \( D_3 \leq D_1 + D_2 \) or, more formally, \( D_3 < D_1 + D_2 \) implies \( D_3 \leq D_1 + D_2 \).

In the following, we write \( h(\Phi) \) as short for \( h^\#(\Phi) \). Moreover, one may consider that the formula \( \Phi_2 \Rightarrow h(\Phi_1) \) is not a proper formula, since \( h(\Phi_1) \) and \( \Phi_2 \) are not formulas, but sets of formulas. However, we may identify any set of formulas \( \Psi \) with the formula \( \text{Conj}(\Psi) = \bigwedge_{\alpha \in \Psi} \alpha \) and with the singleton set including that formula, since they are all logically equivalent. In the following, even if it may be considered an abuse of notation, we will make further use of this implicit equivalence.

Notice that, according to the above definition, given any E-graph \( G \), if \( SP_D \models \Phi \Leftrightarrow \Phi' \) then \( (G, \Phi) \) and \( (G, \Phi') \) are isomorphic in \( \text{SymbGraphs}_D \).

**Remark 1.** We can use \( \text{SymbGraphs}_D \) for doing attributed or symbolic graph transformation. In particular, in [Orejas and Lambers (2010b)](https://doi.org/10.1016/j.jsc.2009.04.008) we define symbolic graph transformation rules as the pairs \((L \leftarrow K \rightarrow R, \Phi)\), where \( L, K, \) and \( R \) are E-graphs over the same set of labels and \( l \) and \( r \) are E-graph monomorphisms. This is equivalent to considering that a symbolic transformation rule is a span \((L, \Phi) \leftarrow (K, \Phi) \rightarrow (R, \Phi)\) in \( \text{SymbGraphs}_D \). Then, these rules can be used for transformation in two different ways. First, we can do graph transformation using the standard double pushout approach in \( \text{SymbGraphs}_D \). In this case, given a symbolic graph \( AG = (G, \Psi) \) and a match morphism \( h : (L, \Phi) \rightarrow AG \) the result of the transformation is the symbolic graph \( AH = (H, \Psi') \), where \( H \) is the result of the double pushout transformation in the category of E-graphs, when applying the rule \( L \leftarrow K \rightarrow R \) to \( G \) with match \( h \). This kind of symbolic graph transformation can be used to represent attributed graph transformation, when the object graphs \( AG \) are the symbolic representations of an attributed graph. Actually, in [Orejas and Lambers (2010b)](https://doi.org/10.1016/j.jsc.2009.04.008) it is proved that this kind of symbolic graph transformation is more general than attributed graph transformation. However, this kind of symbolic graph transformation may not be adequate when the object graphs are arbitrary symbolic graphs. For this reason, in [Orejas and Lambers (2010a)](https://doi.org/10.1016/j.jsc.2009.07.009) we defined a new form of symbolic graph transformation which, roughly speaking, is based on the following ideas. First, the match morphism \( h \) does not have to map all the labels in \( L \) into labels in \( G \), but only the labels which are actually bound to a node or edge in \( L \). Second, \( h \) does not need to satisfy that \( SP_D \models \Psi \Rightarrow h(\Phi) \), but only that \( \Psi \cup h(\Phi) \) is satisfiable. Finally, the result of the transformation is a symbolic graph \( AH = (H, \Psi') \), where \( H \) is obtained also by a double pushout in the category of E-graphs, \( \Psi' = \Psi \cup h'(\Phi) \), and \( h' \) is the comatch in the double pushout. This kind of transformation not only solves the limitations of the former one, but it also allows us to do a useful kind of lazy binding in the transformation process.

### 2.3. Some properties of the category \( \text{SymbGraphs}_D \)

In this section we show that the category \( \text{SymbGraphs}_D \) satisfies some conditions that, following [Orejas et al. (2009)](https://doi.org/10.1201/9781466582586-9), are needed to show the soundness and completeness of the proof system for graph constraints presented in Section 4. The proofs are easy extensions of the corresponding proofs for the category of E-graphs and they can be found in the Appendix.

The first property states that there is a finite number of ways of gluing together two finite symbolic graphs up to isomorphism. More precisely, we consider that a symbolic graph \( (G_0, \Phi_0) \) is the gluing of two graphs \( (G_1, \Phi_1) \) and \( (G_2, \Phi_2) \) if there is a pair of jointly surjective morphisms \( g \) and \( h \) going from \( (G_1, \Phi_1) \) and from \( (G_2, \Phi_2) \), respectively, into \( (G_0, \Phi_0) \), where \( g \) and \( h \) are jointly surjective if, on the one hand, every element (a node or an edge) in \( G_0 \) is the image of an element in \( G_1 \) through \( g \) or of an element in \( G_2 \) through \( h \) and, on the other hand, if \( \Phi_0 \) is equivalent to the union of \( \Phi_1 \) and \( \Phi_2 \):

**Definition 4 (Jointly Surjective Morphisms).** Two morphisms \( g : (G_1, \Phi_1) \rightarrow (G_0, \Phi_0) \) and \( h : (G_2, \Phi_2) \rightarrow (G_0, \Phi_0) \) are jointly surjective if:

---

3 Actually, in [Orejas and Lambers (2010b)](https://doi.org/10.1016/j.jsc.2009.04.008) we also prove that \( \text{SymbGraphs}_D \) is adhesive HLR.
1. For each element \( a_0 \) in \( G_0 \) there is an element \( a_1 \) in \( G_1 \) such that \( g(a_1) = a_0 \) or there is an element \( a_2 \) in \( G_2 \) such that \( h(a_2) = a_0 \), i.e. \( g \) and \( h \) are jointly surjective as E-graph morphisms.

2. \( SP_{\mathcal{D}} \models \Phi_0 \iff (g(\Phi_1) \cup h(\Phi_2)) \).

Given two symbolic graphs \( AG_1 = (G_1, \Phi_1) \) and \( AG_2 = (G_2, \Phi_2) \), we denote by \( AG_1 \otimes AG_2 \) the set of all pairs of jointly surjective monomorphisms from \( AG_1 \) and \( AG_2 \), that is:

\[
\{ g : AG_1 \rightarrow AG_0 \leftrightarrow AG_2 : h \mid g \text{ and } h \text{ are jointly surjective monomorphisms} \}.
\]

Given symbolic graphs \( AG_1 \) and \( AG_2 \), \( AG_1 \otimes AG_2 \) may be seen as consisting of all the possible ways of putting together \( G \) and \( G' \). This definition, in terms of sets of pairs of monomorphisms, may look a bit more complex than needed, but we often need to identify the specific instances of \( AG_1 \) and \( AG_2 \) inside \( AG_0 \). As we can see below, if \( AG_1 \) and \( AG_2 \) are finite graphs then \( AG_1 \otimes AG_2 \) is also a finite set (up to isomorphism). This is needed because in several inference rules (see Section 4) the result is a clause involving a disjunction related to a set of this kind.

**Proposition 1 (Finite Gluing).** Given \( AG_1 = (G_1, \Phi_1) \) and \( AG_2 = (G_2, \Phi_2) \), if \( G_1 \) and \( G_2 \) are finite graphs then \( AG_1 \otimes AG_2 \) is also finite up to isomorphism.

The second property is known as pair factorization (Ehrig et al., 2006b).

**Proposition 2 (Pair Factorization).** Given symbolic graphs \( AG_1 = (G_1, \Phi_1) \), \( AG_2 = (G_2, \Phi_2) \), and \( AG = (G, \Phi) \), and morphisms, \( h_1 : AG_1 \rightarrow AG \leftrightarrow AG_2 : h_2 \), there exists a symbolic graph \( AG_0 \), two morphisms \( g_1 : AG_1 \rightarrow AG_0 \leftrightarrow AG_2 : g_2 \), and a monomorphism \( h : AG_0 \rightarrow AG \) such that \( g_1 \) and \( g_2 \) are jointly surjective and the diagram below commutes:

\[
\begin{array}{ccc}
AG_1 & \xrightarrow{f_1} & AG_2 \\
\downarrow g_1 & & \downarrow h_1 \\
AG_0 & \xrightarrow{h} & AG \\
\downarrow g_2 & & \downarrow h_2 \\
AG_2 & & \\
\end{array}
\]

Moreover, if \( h_1 \) and \( h_2 \) are monomorphisms so are \( g_1 \) and \( g_2 \).

The last property that we need, to apply the results in Orejas et al. (2009) to our category of symbolic graphs, is the existence of colimits of infinite sequences of monomorphisms. Moreover, these colimits must satisfy an additional finiteness property:

**Proposition 3 (Infinite Colimits).** Given a sequence of monomorphisms:

\[
\begin{array}{ccc}
AG_1 & \xrightarrow{f_1} & AG_2 \\
& \vdots & \\
& \xrightarrow{f_{i-1}} & AG_i \\
\end{array}
\]

there exists a colimit:

\[
\begin{array}{ccc}
AG_1 & \xrightarrow{f_1} & AG_2 \\
\downarrow h_1 & & \downarrow f_2 \\
AG & \xrightarrow{f_{i-1}} & AG_i \\
\downarrow h_2 & & \downarrow f_i \\
AG & & \\
\end{array}
\]

that satisfies that for every monomorphism \( g : AG' \rightarrow AG \), where \( AG' = (G', \Phi') \) and \( G' \) is a finite graph, there is a \( j \) and a monomorphism \( g_j : AG' \rightarrow AG_j \) such that the diagram below commutes:

\[
\begin{array}{ccc}
AG' & \xrightarrow{g_j} & AG_j \\
\downarrow g & & \downarrow h_j \\
AG & & \\
\end{array}
\]
3. Attributed graph constraints

The underlying idea of a graph constraint is that it specifies that certain structures must be present (or must not be present) in a given graph. For instance, the simplest kind of graph constraint, $\exists C$, specifies that a given graph $G$ should include (a copy of) $C$. Obviously, $\neg \exists C$ specifies that a given graph $G$ should not include $C$. A slightly more complex kind of graph constraints are atomic constraints of the form $\forall(c : X \rightarrow C)$ where $c$ is a monomorphism (or just an inclusion). This constraint specifies that whenever a graph $G$ includes (a copy of) the graph $X$ it should also include (a copy of) its extension $C$. However, in order to enhance readability (the monomorphism arrow may be confused with the edges of the graphs), in our examples we will display this kind of constraint using an if–then notation, where the two graphs involved have been labeled to implicitly represent the given monomorphism. For instance, the constraint in Fig. 3 specifies that a graph must be transitive, i.e. the constraint says that for every three nodes, $a$, $b$, $c$ if there is an edge from $a$ to $b$ and an edge from $b$ to $c$ then there should be an edge from $a$ to $c$.

Graph constraints can be combined using the standard connectives $\lor$ and $\neg$ (as usual, $\land$ can be considered a derived operation). In Habel and Pennemann (2009) and Rensink (2004) a more complex kind of constraint, called nested constraints, is defined, but we do not consider them in this paper.

When dealing with attributed graphs, we may want not only to state properties about graph structures that may be included in the given graph, but also about the attributes involved. This can be easily done using symbolic graphs. More precisely, if the graphs included in a given graph constraint are symbolic graphs, we would be simultaneously stating properties about the graph substructures included in a given graph and about their associated attributes. For instance, if we consider a class of attributed graphs including an attribute on the edges expressing the distance between its source and target nodes, then the constraint in Fig. 4 would specify, not only that a graph must be transitive, but that the distances must satisfy the triangle inequality. Notice that we have omitted the data condition on the first graph, since it is just the true condition.

In Habel and Pennemann (2009), the authors studied graph constraints in the framework of any arbitrary HLR adhesive category, generalizing previous results obtained for constraints over standard graphs. Something similar happens to the inference system presented in Orejas et al. (2009) and the proof that this system is sound and complete. Although the presentation in that paper may give the impression of addressing the case of standard graphs, actually all the concepts and proofs are presented categorically, independently of the specific category considered. In particular, the main proofs and constructions of the paper are based on the properties proved in this paper in Propositions 1–3.

Following the ideas just discussed, in this section we will introduce in some detail attributed graph constraints instantiating the main concepts in our category of symbolic graphs. Then, in the following section we will present an inference system for reasoning with these constraints which is also an instantiation of the proof system presented in Orejas et al. (2009), whose soundness and completeness proofs are also inherited from that paper.

**Definition 5 (Syntax and Satisfaction of Attributed Graph Constraints).** An atomic attributed graph constraint, denoted $\forall(c : AX \rightarrow AC)$ over a data domain $\mathcal{D}$ is a symbolic graph morphism. A constraint
∀(c : AX → AC), where AX = (∅, true), is called a basic constraint and is denoted ∀AC. Given a constraint ∀(c : ⟨X, Φ⟩ → ⟨C, ΦC⟩), we call ∀(c : X → C) its associated graph constraint.

A symbolic graph AG = (G, Φ) satisfies a constraint ∀(c : AX → AC), if for every monomorphism h : AX → AG there is a monomorphism f : AC → AG such that h = f ◦ c.

It may be noted that the constraint ∃⟨∅, true⟩, where ∅ denotes the empty graph, is satisfied by any graph, i.e. ∃∅ may be considered the trivial true constraint.

As in the case of standard graph constraints, we can define general attributed constraints using the logical connectives ∨ and ¬. Then, satisfaction may be extended accordingly. Anyhow, in this paper, in order to make the presentation simpler, we will assume that our specifications are finite sets of clauses of the form:

\[
L_1 \lor \cdots \lor L_n
\]

where each literal \( L_i \) is either a positive constraint (either basic or atomic) or a negative basic constraint. That is, we assume that clauses do not include negative atomic constraints. As said above, there is no technical reason except simplicity for this limitation. The results in Orejas et al. (2009) cover also the general case, so their instantiation to symbolic graphs also cover the general case. However, the proof system when dealing with negative atomic constraints is a bit involved. For this reason we have thought that it was more adequate to consider this restriction.

**Example 1.** Let us suppose that we want to describe an information system modeling the lecturing organization of a department. Then the type graph of (part of) our system could be the following one:

```
Room
int Number
int Start
int End

Subject
string Name

Lecturer
string Name
```

This means that in our system we have three types of nodes. Rooms including three attributes, the room number and a time slot denoted by the attributes Start and End, and Subjects and Lecturers, both having one attribute Name. We also have two types of edges. In particular, an edge from a Subject \( S \) to a Lecturer \( L \) means that, obviously, \( L \) is the lecturer for \( S \). An edge from a Subject \( S \) to a Room means that the lecturing for \( S \) takes place in that room for the given time slot. Now for this system we could include the following constraint:

\[
\exists \left( \begin{array}{c}
\text{Subject} \\
\text{Name}=X
\end{array} \right) \quad \text{with} \quad X = CS1 \land Y = CS2
\]

meaning that the given system must include the compulsory subjects Computer Science 1 and Computer Science 2. Moreover we may have a constraint, like constraint (2), saying that every subject included in the system must have some lecturer assignment and some room assignment:

\[
(2) \quad \text{if} \quad \begin{array}{c}
\text{Subject} \\
\text{Name}=X
\end{array} \quad \text{then} \quad \begin{array}{c}
\text{Subject} \\
\text{Name}=Y
\end{array}
\]

Notice that when only the \text{true} condition is constraining the attributes of the given graphs we just do not display these attributes.

We may also have constraints expressing some negative conditions. For instance, that there cannot be a room node with a negative time slot (constraint (3)). Or that a room is not assigned at the same time to two subjects (constraint (4)) or that two different rooms are not assigned with overlapping time slots to the same subject (constraint (5)):
or, finally (constraint (6)), that a lecturer does not have to lecture on two different subjects in two different rooms at the same time:

\[
\neg \exists (\text{Lecturer} \leftarrow \text{Subject}) \quad \text{with} \quad (T_1 \leq T_1') \land (T_1' < T_2)
\]

4. Proof rules for attributed constraints

In this section we provide inference rules for attributed graph constraints as considered in this paper. For simplicity, we only consider the case where clauses include (positive or negative) basic attributed constraints and positive atomic attributed constraints. This means that the given clauses are assumed to consist of literals of the form \(\langle \exists C_1, \Phi_1 \rangle\), \(\neg \langle \exists C_1, \Phi_1 \rangle\), or \(\langle \forall (c : X \rightarrow C_2), \Phi_2 \rangle\).

The rest of the section is organized as follows. In Section 4.1 we describe how inference rules define refutation procedures. Then, in Section 4.2 we define the inference rules for the case considered. Finally, in Section 4.3 we present our soundness and completeness results.

4.1. Refutation procedures

In this section we describe the framework that we use to present our refutation procedure. We follow an approach which is quite standard in the area of automated deduction (e.g. this is the approach followed to describe resolution, or paramodulation theorem proving). The procedure is defined by means of some inference rules. Then, a refutation procedure can be seen as a (possibly nonterminating) nondeterministic computation where the current state is given by the set of formulas that have been inferred until the given moment and where a computation step means adding to the given state the result of applying an inference rule to that state. The procedure terminates when no new inference can be applied or when the false formula (which is represented by the empty clause, denoted \(\Box\)) is inferred. In the latter case, we conclude that the given set of formulas is unsatisfiable.

We assume that a first-order specification \(SP_D\) of the data domain is given which characterizes the algebras that can be used for defining the class of symbolic graphs of interest. Moreover, we also assume that we can check if a given formula or set of formulas is a logical consequence of \(SP_D\) or if it is satisfiable in \(\text{Mod}(SP_D)\). In this sense, our refutation procedure may be considered to be parameterized by the specification of the given data domain. For instance, some inference rules need to check, as a side condition, the existence of a symbolic graph morphism, which means if certain formulas are a
logical consequence of that specification. Thus a proof tool implementing this procedure needs to be built on top of a deductive tool for first-order specifications.

In our case, we assume that, in general, the inference rules have the form:

\[
\frac{\Gamma_1 \quad \Gamma_2}{\Gamma_3}
\]

where \( \Gamma_1, \Gamma_2, \) and \( \Gamma_3 \) are clauses. Then a refutation procedure for a set of constraints \( \mathcal{C} \) is a sequence of inferences:

\[
\mathcal{C}_0 \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_i \Rightarrow \cdots
\]

where the initial state coincides with the initial set of constraints (i.e. \( \mathcal{C}_0 = \mathcal{C} \)) and where we write \( \mathcal{C}_i \Rightarrow \mathcal{C}_{i+1} \) if there is an inference rule such as the one above such that \( \Gamma_1, \Gamma_2 \in \mathcal{C}_i \), and \( \mathcal{C}_{i+1} = \mathcal{C}_i \cup \{ \Gamma_3 \} \). Moreover, we assume that \( \mathcal{C}_i \subset \mathcal{C}_{i+1} \), i.e. \( \Gamma_3 \notin \mathcal{C}_i \), to avoid useless inferences.

Since the application of rules is nondeterministic, there is the possibility that we never apply some key inference. To avoid this problem we assume that procedures are \textit{fair}, which means that if at any moment \( i \), there is a possible inference \( \mathcal{C}_i \Rightarrow \mathcal{C}_i \cup \{ \Gamma \} \), then at some moment \( j \) we have that \( \Gamma \in \mathcal{C}_j \).

Then, a refutation procedure for \( \mathcal{C} \) is \textit{sound} if whenever the procedure infers the empty clause we have that \( \mathcal{C} \) is unsatisfiable. And a procedure is \textit{complete} if, whenever \( \mathcal{C} \) is unsatisfiable, we have that the procedure infers the empty clause.

4.2. The inference rules

As said above, the first three inference rules that we consider are just an instantiation, to the case of symbolic graphs, of the proof rules presented in Orejas et al. (2009). Then, since our category of symbolic graphs satisfies the required properties, we can inherit the soundness and completeness results which were proved in Orejas et al. (2009). Unfortunately, as we will see below, the completeness result inherited from Orejas et al. (2009) is too weak to be of interest. However, we will see that adding another rule makes our inference system complete in the right sense.

The instantiation of the inference rules from Orejas et al. (2009) yields the following three rules, where we have made explicit in their corresponding provisos what needs to be proved from the given data domain specification.

\[
\exists(C_1, \Phi_1) \lor \Gamma_1 \quad \neg\exists(C_2, \Phi_2) \lor \Gamma_2 \quad (R1)
\]

if there exists a monomorphism \( m : C_2 \rightarrow C_1 \) and \( \mathcal{S}_{P_D} \models \Phi_1 \Rightarrow m(\Phi_2) \)

\[
\exists AC_1 \lor \Gamma_1 \quad \exists AC_2 \lor \Gamma_2 \quad (R2)
\]

where \( \mathcal{G} = \{ \text{AG} | \langle f_1 : AC_1 \rightarrow AG \leftarrow AC_2 : f_2 \rangle \in (AC_1 \otimes AC_2) \} \) and \( \bigvee_{\text{AG} \in \mathcal{G}} \exists \text{AG} \) denotes the (finite) disjunction \( \exists \text{AG}_1 \lor \cdots \lor \exists \text{AG}_n \), if \( \mathcal{G} = \{ \text{AG}_1, \ldots, \text{AG}_n \} \).

\[
\exists(C_1, \Phi_1) \lor \Gamma_1 \quad \forall(c : \langle X, \Phi_X \rangle \rightarrow AC_2) \lor \Gamma_2 \quad (R3)
\]

if there is a monomorphism \( m : X \rightarrow C_1 \), such that \( \mathcal{S}_{P_D} \models \Phi_1 \Rightarrow m(\Phi_X) \), and \( \mathcal{G} = \{ \text{AG} | \langle f_1 : \langle C_1, \Phi_1 \rangle \rightarrow AG \leftarrow AC_2 : f_2 \rangle \in (\langle C_1, \Phi_1 \rangle \otimes AC_2) \} \) such that \( f_1 \circ m = f_2 \circ c \).
The first rule is similar to resolution and it is the rule that may allow us to infer the empty clause, since it is the only rule that eliminates literals from clauses. The second rule can be seen as a rule that, given two constraints, builds a new constraint that subsumes them. More precisely, the graphs involved in the new literals of the conclusion of the inference rule, i.e. the graphs \( AG \), satisfy both graph constraints \( \exists AC_1 \) and \( \exists AC_2 \) and, in addition, the associated attribute condition of each graph \( AG \) (according to Definition 4, this condition is \( f_1(\Phi_1) \cup f_2(\Phi_2) \)) represents the combination of the conditions of the atomic constraints involved in the rule. This means that if we apply this rule repeatedly, using all the positive constraints in the original set \( C \), we would build (minimal) symbolic graphs that satisfy all the positive basic constraints in \( C \). The third rule is similar to rule (R2) in the sense that, given a positive basic constraint and a positive atomic constraint, it builds a disjunction of literals representing symbolic graphs that try to satisfy both constraints. However, in this case the satisfaction of the graph constraint \( \forall (c : AX \rightarrow AC_2) \) is not ensured. In particular, the idea of the rule is that if we know that \( AX \) is included in \( AC_1 \) then we build all the possible extensions of \( AC_1 \) which also include \( AC_2 \) (each \( AG \) is one of such extensions). But in this case we cannot be sure that \( AG \) satisfies \( \forall (c : AX \rightarrow AC_2) \), because \( AG \) may include other instances of \( AX \), which perhaps were not included in \( AC_1 \).

Example 2. Let us consider the constraints that are included in Example 1 (i.e. the constraints (1), (2), (3), (4), (5), and (6)). If we apply the third rule on constraints (1) and (2), and again on the resulting clause and on constraint (2) then we infer the following clause:

\[
\exists (\begin{array}{l}
\text{Room} \\
\text{Subject Name=X} \\
\text{Lecturer}
\end{array}) \lor \exists (\begin{array}{l}
\text{Room} \\
\text{Subject Name=Y} \\
\text{Lecturer}
\end{array})
\]

with \( X = CS1 \land Y = CS2 \)

\[
\lor \exists (\begin{array}{l}
\text{Room} \\
\text{Subject Name=X} \\
\text{Lecturer}
\end{array}) \lor \exists (\begin{array}{l}
\text{Room} \\
\text{Subject Name=Y} \\
\text{Lecturer}
\end{array})
\]

with \( X = CS1 \land Y = CS2 \)

This clause states that the graph should include two subjects (CS1 and CS2) and these subjects may be assigned to two different rooms and to either two different lecturers, or to the same lecturer, or they may be assigned to the same room, and to either different lecturers, or the same lecturer. Obviously, the last two constraints in this clause violate constraint (4), which means that we can eliminate them using rule (R1) twice, yielding the following clause:

\[
\exists (\begin{array}{l}
\text{Room} \\
\text{Subject Name=X} \\
\text{Lecturer}
\end{array}) \lor \exists (\begin{array}{l}
\text{Room} \\
\text{Subject Name=Y} \\
\text{Lecturer}
\end{array})
\]

with \( X = CS1 \land Y = CS2 \)
At this point, we could stop the inference process since the two graphs in (8) are already (minimal) models of the given set of constraints, which means that it is satisfiable.

4.3. Soundness and completeness

As proven in Orejas et al. (2009), the rules (R1), (R2), and (R3) are sound and complete. The soundness of the first rule is quite obvious. If a symbolic graph $AG$ satisfies the constraint $\exists \Gamma_1$, then it cannot satisfy $\exists \Gamma_2$. Therefore if $AG$ satisfies both premises then it must satisfy either $\Gamma_1$ or $\Gamma_2$. And the soundness of the second and third rules is based of the pair factorization property. For instance, if $AG$ satisfies $\exists AC_1$ and $\exists AC_2$, i.e. $AC_1$ and $AC_2$ are embedded in $AG$, then the pair factorization property ensures that there is a symbolic graph $AC$ which is a gluing of $AC_1$ and $AC_2$, and which is also embedded in $AG$, i.e. $AG$ satisfies $\exists AC$.

**Lemma 1** (Soundness of the Inference Rules). Rules (R1), (R2), and (R3) are sound.

Proving completeness is more involved. The underlying idea of the completeness proof is to consider ascending sequences, with respect to a given precedence relation, of basic positive literals (or their associated graphs) occurring in clauses that may be inferred from the original set of constraints. Then, we show that the colimit of one of these sequences is a model of the given specification. More precisely, we see that the sequences considered represent a construction of possible models using the inference rules (R2) and (R3). In what follows, we present the main concepts and intermediate results, together with a sketch of their proofs, that are needed to prove completeness of our inference rules. See Orejas et al. (2009) for a detailed proof of these intermediate results.

In all the results below, we assume that we have a fair derivation $C_0 \Rightarrow C_1 \Rightarrow \cdots \Rightarrow C_i \Rightarrow \cdots$, and we denote by BasPosLit($\bigcup_{k \geq 1} C_k$) the set of all basic positive literals occurring in a clause in $\bigcup_{k \geq 1} C_k$. The fact that the derivation is fair ensures that if it is possible to infer a clause from the given set of constraints then this clause is in $\bigcup_{k \geq 1} C_k$, which implies that if a basic positive literal is in a clause that may be inferred from $C_0$ then this literal is in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$.

The precedence relation that we consider between basic positive constraints is just the existence of a monomorphism between the associated graphs: monomorphism from $AG_i$ to $AG_{i+1}$.

**Definition 6.** For every pair of literals $\exists AG_1$, $\exists AG_2$, $\exists AG_1 \prec \exists AG_2$ if there is a monomorphism $h_{AG_1 \rightarrow AG_2} : AG_1 \rightarrow AG_2$.

More precisely, the ascending sequences that we consider are the ones which are saturated, where intuitively a sequence is saturated if either it leads to a model of the given set of clauses, or if we know that the sequence cannot lead to a model because a literal in the sequence does not satisfy a negative constraint. In particular, in the latter case, we know that if $AG_i$ does not satisfy the constraint $\neg \exists AG$ then for any $AG_j$ such that $AG_i \prec AG_j$ then $AG_j$ does not satisfy $\neg \exists AG$ either. The same happens if $AG_i$ does not satisfy a clause consisting only of negative constraints. This means that if we are considering a sequence $\exists AG_0 \prec \exists AG_1 \prec \cdots \prec \exists AG_i$ for building a model of $C_0$ and $AG_i$ does not satisfy a negative constraint, then it is useless to try to extend this sequence with additional elements. In this case we say that its last element is closed.

**Definition 7.** A literal $\exists AG$ in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ is closed if there is a clause $\Gamma$ in $\bigcup_{k \geq 1} C_k$ consisting only of negative constraints, such that $AG \not\equiv \Gamma$. We also say that $\exists AG$ is open if it is not closed.

As said above, a saturated sequence is a sequence that provides successive approximations to a model unless it ends on a closed literal:

**Definition 8.** An ascending sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ $\exists AG_1 \prec \cdots \prec \exists AG_i \prec \cdots$ is saturated if one of the following cases applies:

- the sequence is finite and its last element $\exists AG_k$ satisfies that $AG_k$ is a model for $C$, or
- the sequence is finite and its last element is closed, or
- the sequence is infinite and for every clause $\Gamma$ in $\bigcup_{k \geq 1} C_k$ there is a literal $L$ in $\Gamma$ such that:
(a) if $L = \neg\exists AC$, then for every $j$ there is no monomorphism $m : AC \to AG_j$
(b) if $L = \exists AC$, there is a $j$, such that there is a monomorphism $m : AC \to AG_j$
(c) If $L = \forall(c : X \to AC)$ then for every $i$ and every monomorphism $m : X \to AG_i$ there is a $j$, with $i < j$, and a monomorphism $h : AC \to AG_j$ with $h_{AC_i \alpha AG_j} \circ m = h \circ c$.

The following lemma makes explicit in which sense an infinite saturated sequence provides successive approximations to a model of a given set of constraints:

**Lemma 2.** Let $\exists AG_1 < \cdots < \exists AG_i < \cdots$ be an infinite saturated sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ for a fair refutation procedure $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \cdots$ and let $AG$ be the colimit of the sequence:

\[
\begin{array}{c}
AG_1 \\
\downarrow h_{AG_1 \to AG_2} \\
AG_2 \\
\downarrow h_{AG_2 \to AG_3} \\
\vdots \\
\downarrow h_{AG_{i-1} \to AG_i} \\
\cdots \\
\downarrow h_{AG_{i} \to AG_1} \\
AG_i \\
\end{array}
\]

then $AG$ is a model for the given set of clauses, i.e. $AG \models \mathcal{C}$.

It is possible to build a saturated sequence out of any basic positive constraint:

**Lemma 3.** Given a fair refutation procedure $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \cdots$ if $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ is not empty then there is a saturated sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$.

The last result that we need, before proving completeness for our inference rules, shows that if all saturated sequences end in a closed literal and if the given set of constraints includes a clause consisting only of basic positive literals then we can infer a clause consisting only of closed literals.

**Lemma 4.** Let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \cdots$ be a fair refutation procedure defined over a finite set of constraints $\mathcal{C}$ based on the rules (R1), (R2), and (R3) such that $\mathcal{C}$ includes a clause $\Gamma$ consisting only of basic positive literals. If every saturated sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ is finite and its last element is a closed literal then there is a clause $\Gamma'$ in $\bigcup_{k \geq 1} C_k$ consisting only of closed literals.

Completeness is now not difficult to prove. The idea of the proof goes as follows. If a fair refutation procedure does not generate the empty clause, then we have two cases. If there is no clause in $\mathcal{C}$ consisting only of basic positive constraints then the trivial model (the empty graph) satisfies $\mathcal{C}$. Otherwise, by **Lemma 3** there is at least one saturated sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$. If this sequence is finite and its last element is $\exists AG$, then $AG \models \mathcal{C}$. If this sequence is infinite then, according to **Lemma 2**, its colimit is a model for $\mathcal{C}$. Finally, the remaining situation is when every saturated sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ is finite and its last element is a closed literal, then using **Lemma 4** we can show that it is possible to infer the empty clause. Hence, a fair procedure would infer it, against the original assumption.

**Lemma 5** (Completeness). Let $\mathcal{C}$ be a finite set of clauses consisting of basic constraints and positive atomic constraints, and let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \cdots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3). If $\mathcal{C}$ is unsatisfiable then there is a $j$ such that the empty clause is in $\mathcal{C}_j$.

Therefore, according to **Lemmas 1** and **5** we have that our inference rules (R1), (R2), and (R3) are sound and complete, i.e.

**Theorem 1** (Soundness and Completeness). Let $\mathcal{C}$ be a finite set of clauses consisting of basic constraints and positive atomic constraints, and let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \cdots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3). Then, $\mathcal{C}$ is unsatisfiable if and only if there is a $j$ such that the empty clause is in $\mathcal{C}_j$.

The above result seems to be completely satisfactory, but it is really not so adequate. We have proved that there is a symbolic graph that satisfies a set of constraints if and only if we cannot infer...
the empty clause. However, if we consider that the models of a class of constraints are attributed graphs in the standard sense, this result may be too weak. As pointed out in Section 2.2 a symbolic graph may be seen as the specification of a (possibly empty) class of attributed graphs. In particular, if $\Phi$ is unsatisfiable, then $\langle G, \Phi \rangle$ specifies the empty class of attributed graphs. As a consequence, if the only models of a given set of constraints are symbolic graphs $\langle G, \Phi \rangle$, where $\Phi$ is unsatisfiable then there would be no attributed graph that satisfies this set of constraints.

**Definition 9** (Strong Satisfiability, Strong Soundness, Strong Completeness). A set of clauses $C$ is strongly satisfiable if there is a symbolic graph $\langle G, \Phi \rangle$ such that $G \models C$ and $\Phi$ is satisfiable in $Mod(SP_D)$.

A set of inference rules for graph constraints is strongly sound if for every finite set of clauses $C$ and for every fair refutation procedure $C \Rightarrow C_1 \Rightarrow \cdots \Rightarrow C_k \cdots$ if there is a $j$ such that the empty clause is in $C_j$ then $C$ is strongly unsatisfiable. A set of inference rules for graph constraints is strongly complete if for every finite set of clauses $C$ and for every fair refutation procedure $C \Rightarrow C_1 \Rightarrow \cdots \Rightarrow C_k \cdots$ if $C$ is strongly unsatisfiable then there is a $j$ such that the empty clause is in $C_j$.

If we want to have a sound and strongly complete set of inference rules we just have to add a new rule (R4) to our previous set of rules:

$$\frac{\langle \exists C, \Phi \rangle \lor \Gamma}{\Gamma} \quad \text{(R4)}$$

if $\Phi$ is not satisfiable in $Mod(SP_D)$.

stating that if a given clause includes a literal with an unsatisfiable attribute condition then we can delete that literal.

The new set of rules is obviously sound:

**Lemma 6** (Strong Soundness of the Inference Rules). Rules (R1), (R2), (R3), and (R4) are strongly sound.

To show the strong completeness of the rules we may use the proof of Lemma 5, just changing the notions of a closed and open literal:

**Definition 10.** A literal $\exists AG = \langle G, \Phi \rangle$ in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ is closed if there is a clause $\Gamma$ in $\bigcup_{k \geq 1} C_k$ consisting only of negative constraints, such that $AG \not\models \Gamma$ or if $\Phi$ is unsatisfiable. We also say that $\exists AG$ is open if it is not closed.

**Lemma 7** (Completeness). Let $C$ be a finite set of clauses consisting of basic constraints and positive atomic constraints, and let $C \Rightarrow C_1 \Rightarrow \cdots \Rightarrow C_k \cdots$ be a fair refutation procedure defined over $C$ based on the rules (R1), (R2), (R3), and (R4). If $C$ is strongly unsatisfiable then there is a $j$ such that the empty clause is in $C_j$.

Therefore:

**Theorem 2** (Soundness and Completeness). Rules (R1), (R2), (R3), and (R4) are strongly sound and strongly complete.

5. Related work

There are essentially two kinds of approaches to define attributed graphs and attributed graph transformation. On the one hand, we have the approaches (Heckel et al., 2002; Ehrig et al., 2006a) where an attributed graph is a pair $(G, D)$ consisting of a graph $G$ and a data algebra $D$ whose values are nodes in $G$. On the other hand, we have the approaches (Löwe et al., 1993; Berthold et al., 2000) 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 Ehrig (2004) the two approaches are compared showing that they are, up to a certain point, equivalent.
However, only a complete theory of graph transformation has been formulated for Ehrig et al. (2006a)
as a consequence of its characterization as an adhesive HLR category (for more detail see Ehrig et al.
(2006b)).

Our approach can be seen as a symbolic version of Ehrig et al. (2006a). However it has more
expressive power than Ehrig et al. (2006a) for the definition of constraints. Let us analyze intuitively
(i.e. not being completely rigorous) the expressive power of both approaches.

As discussed in Section 2 a symbolic graph in our sense can be seen as a (possibly empty) class
of attributed graphs. More precisely, an attributed graph can be considered equivalent to a symbolic
graph where the values associated to nodes or edges in the graph have been replaced by variables,
and where the associated formula consists of a conjunction of equalities binding each variable to the
 corresponding value. Let us now see the inverse relation, i.e. how we can associate an attributed graph
to a symbolic graph to be used as a basic constraint.

As described above, we know that an attributed graph is a pair \((G, D)\) consisting of a graph \(G\) (an E-
 graph, according to Ehrig et al. (2006a)) and a data algebra \(D\) whose values are nodes in \(G\). Then, given
a symbolic graph \(SG = (G, \Phi)\), a technique to define an equivalent attributed graph \(AG\), which may
work in many cases, is the following one. On the one hand, we define \(D\) as the initial algebra associated
to the specification \((\Sigma_D \cup L, \Phi)\), where \(\Sigma_D\) is the signature of the data domain specification \(SP_D\) and
\(L\) is the set of labels of \(G\). On the other hand, we define \(AG\) as \((G', D)\), where \(G'\) is the graph obtained
adding to \(G\) all the elements in \(D\). For example, let us suppose that we want to define the attributed
graph associated to the symbolic graph below:

\[
\begin{align*}
X &\xrightarrow{D_1} Y &\xrightarrow{D_2} Z \\
&D_3 \quad \text{with } (D_3 \leq D_1 + D_2) = t
\end{align*}
\]

where the signature of the data domain is:

\[
\begin{align*}
\text{Sorts} &\quad \text{nat, bool} \\
\text{Ops} &\quad 0 : \text{nat} \\
&\quad \text{suc} : \text{nat} \rightarrow \text{nat} \\
&\quad t, f : \text{bool} \\
&\quad + : \text{nat} \times \text{nat} \rightarrow \text{nat} \\
&\quad \leq : \text{nat} \times \text{nat} \rightarrow \text{bool}
\end{align*}
\]

Then, we can intuitively see that the symbolic graph \(SG\) and its associated attributed graph \(AG\) are
equivalent in the sense that given an attributed graph \(AG_0 = (G_0, \mathcal{N})\), where \(\mathcal{N}\) is the algebra of the
natural numbers (including the booleans), there is a morphism from \(AG\) to \(AG_0\) if and only if there
is a morphism from \(SG\) into \(SG_0\), where \(SG_0\) is the symbolic graph associated to \(AG_0\). Moreover both
morphisms essentially define the same matching.
Unfortunately, the technique that I have just described does not always work. In particular, the problem is that it may be impossible to define the algebra $D$ because the given specification has no initial algebra. In particular, let us consider the following symbolic graph:

![Symbolic Graph]

with $X \neq Y \Rightarrow X = 0$

where the signature of the data domain is:

Sorts $\text{nat}$  
Opns $0 : \text{nat}$  
$suc : \text{nat} \rightarrow \text{nat}$  

Then, the associated specification is:

Sorts $\text{nat}$  
Opns $0 : \text{nat}$  
$x, y : \text{nat}$  
$suc : \text{nat} \rightarrow \text{nat}$  

Axms $X \neq Y \Rightarrow X = 0$

This specification, as in general happens with specifications including conditional equations with negative conditions, has no initial algebra. Actually, there is no obvious attributed graph that can be considered equivalent to the above symbolic graph.

In addition to the expressive power, using symbolic graphs has some other advantages. For instance, in Naeem et al. (2010) working with symbolic 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.

In the case of the work that we present in this paper, working with our attributed constraints also simplifies some aspects. As pointed out in the introduction, one main problem that justified our approach is related to inference rules (R2) and (R3). In particular, in these rules the result involves a disjunction over the set $AC_1 \otimes AC_2$. Hence if this set is infinite the conclusion of these inferences would be an infinitary formula. In the case of symbolic graphs we have proved the finite gluing property (see Proposition 1) which ensures that $AC_1 \otimes AC_2$ is finite provided that $AC_1$ and $AC_2$ are finite. Unfortunately, this is not true in the case of attributed graphs for the most obvious definition of $AC_1 \otimes AC_2$. The problem is that even if the graph part of $AC_1$ and $AC_2$ (i.e. their regular nodes and edges) is finite, $AC_1$ and $AC_2$ are infinite graphs if their data algebra is infinite. If $AC_1$ and $AC_2$ share the same data algebra we can restrict the class of jointly surjective morphisms to those that coincide on the data part. Unfortunately, in general the data algebras of $AC_1$ and $AC_2$ will not coincide, because of the variables involved in the literals, and in this case it is not at all obvious what kind of restriction for the class of jointly surjective morphisms would be needed to ensure finite gluing.

Concerning reasoning with graph constraints the work that we present is not the first logic to reason about graphs. On the one hand, Parisi and Koch (see e.g. Koch et al. (2005)) use graph constraints and graph transformation rules to specify access control policies. In particular, graph constraints are used to describe the valid states of a system, and graph transformation rules to specify operations. In addition, they use some form of deduction on constraints to check the consistency of a policy. Unfortunately, the kind of deduction used may be considered quite ad hoc and incomplete.

More recently Pennemann (2008) proposes a proof system for nested graph constraints, a generalization of the kind of constraints considered in our work. The proof system is proven sound but not complete. In addition, Pennemann describes an implementation of his approach providing interesting results.
On the other hand, with different aims, Courcelle in a series of papers (for a survey, see Courcelle (1997)) has studied in detail the definition and use of a graph logic (in the following called CL, from Courcelle Logic). His approach can be seen as a coding of graphs and graph properties into first-order or monadic second-order logic. In particular, the approach is based on the use of some predicates describing the existence of nodes and edges which, together with some given axioms, provide an axiomatization of the basic graph theory. Then, one can express graph properties using standard first-order or monadic second-order formulas over these predicates. Our constraints can be seen as a fragment of CL in the sense that a graph constraint can be coded into a sentence in that logic. Actually, nested constraints have been proved equivalent to the first-order fragment of CL (Habel and Pennemann, 2009). As a consequence, we can question whether it is really needed to develop proof techniques for our constraints, since we can do this indirectly: by coding the constraints into CL and using standard logic deduction. We think that there are two main reasons that justify our work in this direction. First, studying directly the constraints logic gives you insights about the logic that we do not obtain using the coding. For instance, our completeness proofs implicitly tell us how we can design procedures to build models for a given set of constraints. This is interesting for applications like the one presented in de Lara and Guerra (2008), where building a model is, in a sense, equivalent to synthesizing the specified model transformation. And, second, we believe that we can gain significant efficiency. Actually, this kind of discussion is not new. For instance, the development of proof techniques for first-order logic with equality has sometimes been questioned, considering that one could use the standard techniques for first-order logic without equality together with an axiomatization of the equality predicate. However, the study of first-order logic with equality has allowed the development of powerful techniques which are the basis of very efficient tools. In this sense, Pennemann (2008) compares his implementation for his proof system for nested constraints with an implementation based on coding the constraints into CL and then using some standard provers like VAMPIRE, DARWIN and PROVER9. The result is that his implementation outperforms the coding approach. Actually, in most examples considered, the above provers were unable to terminate in the given time (1 hour of cpu time). Unfortunately, these results cannot be considered technically valid, since the completeness of Pennemann’s proof system is not shown.

6. Conclusion

We have presented a new approach to deal with attributed graphs, which may be seen as a symbolic version of the approach in Ehrig et al. (2006b). In particular we have studied this new approach showing that the new categories of graphs enjoy some categorical properties that allow us to instantiate the inference rules from Orejas et al. (2009) obtaining a sound and (weakly) complete proof system for our attributed graph constraints. Moreover, we have seen that adding a new inference rule makes our system sound and strongly complete.

We believe that this new approach to attributed graphs may be of interest for other problems related to graph transformation. For instance, in Naeem et al. (2010) this approach was helpful in showing how we can compose incrementally services to satisfy complex requester needs.

We have not yet implemented these techniques, although it would not be too difficult to implement them on top of the AGG system, given that the basic construction that we use in our inference rules (e.g. building \( AG_1 \otimes AG_2 \)) is already implemented there.

Appendix

Proof of Proposition 1. If \( G_1 \) has \( N_1 \) elements (including nodes and edges) and \( G_2 \) has \( N_2 \) elements and \( g : AG_1 \rightarrow AG_0 \leftarrow AG_2 : h \) are jointly surjective, where \( AG_0 = (G_0, \Phi_0) \), then \( G_0 \) has at most \( N_1 + N_2 \) elements. Hence, there is a finite number of graphs \( G_0 \) that can be the E-graph component of a symbolic graph \( AG_0 \), which is the image of two jointly surjective morphisms. This also implies that there is a finite number of pairs of E-graph morphisms \( g : G_1 \rightarrow G_0 \leftarrow G_2 : h \) which are jointly surjective. Finally, suppose that we have two pairs of jointly surjective symbolic morphisms \( g : AG_1 \rightarrow AG_0 \leftarrow AG_2 : h \) and \( g' : AG_1 \rightarrow AG'_0 \leftarrow AG_2 : h' \), where \( AG_0 = (G_0, \Phi_0) \) and
AG₀ = ⟨G₀, Φ₀⟩ and where g = g' and h = h' as E-graph morphisms. Then, this means that SP₅₀ ⊨ Φ₀ ⇔ (g(Φ₁)∪h(Φ₂)) and SP₅₀ ⊨ (g(Φ₁)∪h(Φ₂)). Therefore, SP₅₀ ⊨ Φ₀ ⇔ Φ₀', implying that AG₀ and AG₀' are isomorphic. As a consequence, for each pair of jointly surjective E-graph morphisms g : G₁ → G₀ ← G₂ : h there is a unique pair of symbolic graph morphisms g : AG₁ → AG₀ ← AG₂ : h up to isomorphism, which means that AG₁ ⊗ AG₂ is finite. □

**Proof of Proposition 2.** E-graphs satisfy the pair decomposition property (Ehrig et al., 2006b), so there is an E-graph G₀, jointly surjective morphisms g₁ : G₁ → G₀ ← G₂ : g₂, and a monomorphism h : G₀ → G such that the diagram below commutes:

\[
\begin{array}{ccc}
G₁ & \xrightarrow{f₁} & G₂ & \cdots & \xrightarrow{fᵢ₋₁} & Gᵢ & \xrightarrow{fᵢ} & \cdots \\
h₁ & \downarrow & h₂ & \cdots & hᵢ & \downarrow & hᵢ' & \downarrow & h’
\end{array}
\]

Now, g₁ : ⟨G₁, Φ₁⟩ → ⟨G₀, g₁(Φ₁) ∪ g₂(Φ₂)⟩ ← ⟨G₂, Φ₂⟩ : g₂ are morphisms in SymbGraphs, since SP₅₀ ⊨ (g₁(Φ₁)∪g₂(Φ₂)) ⇒ g₁(Φ₁) and SP₅₀ ⊨ (g₁(Φ₁)∪g₂(Φ₂)) ⇒ g₂(Φ₂). Also, h : ⟨G₀, g₁(Φ₁) ∪ g₂(Φ₂)⟩ → ⟨G, Φ⟩ is a morphism in SymbGraphs since if SP₅₀ ⊨ Φ ⇒ h₁(Φ₁), SP₅₀ ⊨ Φ ⇒ h₂(Φ₂), then SP₅₀ ⊨ (h(g₁(Φ₁)) ∪ h(g₂(Φ₂))), implying SP₅₀ ⊨ Φ ⇒ h₁(Φ₁) ∪ h₂(Φ₂)).

**Proof of Proposition 3.** These colimits exist in the category of E-graphs (see Orejas et al. (2009) for a detailed proof in the case of graphs). Therefore, we can build the colimit diagram below:

\[
\begin{array}{ccc}
AG₁ & \xrightarrow{f₁} & AG₂ & \cdots & \xrightarrow{fᵢ₋₁} & AGᵢ & \xrightarrow{fᵢ} & \cdots \\
h’₁ & \downarrow & h’₂ & \cdots & h’ᵢ & \downarrow & h’ᵢ & \downarrow & h’
\end{array}
\]

This means that the diagram below is a cocone:

\[
\begin{array}{ccc}
G₁ & \xrightarrow{f₁} & G₂ & \cdots & \xrightarrow{fᵢ₋₁} & Gᵢ & \xrightarrow{fᵢ} & \cdots \\
h’₁ & \downarrow & h’₂ & \cdots & h’ᵢ & \downarrow & h’ᵢ & \downarrow & h’
\end{array}
\]

Hence, there is a unique morphism h : G → G' such that, for each i, h ∘ hᵢ = hᵢ'. So, we have to prove that h : AG → AG' is a morphism in SymbGraphs. But we know that, for every i, SP₅₀ ⊨ Φ' ⇒ hᵢ(Φᵢ), which means that SP₅₀ ⊨ Φ' ⇒ h(hᵢ(Φᵢ)), and this implies that SP₅₀ ⊨ Φ' ⇒ h(∪₁≤₁ hᵢ(Φᵢ)).
For the second part of the proposition, let \( g : \langle G', \Phi' \rangle \to AG \) be a monomorphism, where \( G' \) is a finite graph. In the category of E-graphs there is a \( k \) and a monomorphism \( g_k : G' \to G_k \) such that the diagram below commutes:

\[
\begin{array}{ccc}
G' & \xrightarrow{g_k} & G_k \\
g \downarrow & & \downarrow h_k \\
G & \xrightarrow{h} & \end{array}
\]

We know that \( SP_D \models h(\bigcup_{i \leq 1} h_i(\Phi_i)) \Rightarrow \Phi' \). This means that the set of formulas \( h(\bigcup_{i \leq 1} h_i(\Phi_i)) \cup \neg \Phi' \cup AX_D \) is unsatisfiable. Then, by the compactness of first-order logic, there should exist a finite set of indices \( I = \{i_1, \ldots, i_n\} \) such that \( h(\bigcup_{i \in I} h_i(\Phi_i)) \cup \neg \Phi' \cup AX_D \) is unsatisfiable. Actually, if \( i_n \) is the largest index in \( I \) then \( h(h_n(\Phi_{i_n})) \cup \neg \Phi' \cup AX_D \) is unsatisfiable, since \( \Phi_{i_n} \) implies all the other sets of conditions, with respect to the variable renamings associated to the monomorphisms in the sequence. But this means that \( SP_D \models h(h_n(\Phi_{i_n})) \Rightarrow \Phi' \). Let \( j = \max(k, i_n) \) and let \( g_j = f_{j-1} \circ \ldots \circ f_k \circ g_k \). It is routine to check that \( g_j \) is a morphism in \( \text{SymbGraphs}_D \) and that the diagram below commutes.

\[
\begin{array}{ccc}
AG' & \xrightarrow{g_j} & AG_j \\
g \downarrow & & g_j \downarrow \\
AG & \xrightarrow{h_j} & \end{array}
\]

\( \square \)

**Proof of Lemma 1.** This proof is just an instantiation of the corresponding proof in Orejas et al. (2009). Anyhow, to give an idea of the proof in Orejas et al. (2009), below we show the soundness of (R3), which may be considered the most complex of the three rules:

Suppose that \( AG \models \exists AC \lor I_1, AG \models \forall (c : AX \to AC_2) \lor I_2 \), and there is a monomorphism \( m : AX \to AC_1 \). The case where \( AG \models I_1 \) or \( AG \models I_2 \) is trivial. Suppose that \( AG \models \exists AC_1 \) and \( AG \models \forall (c : AX \to AC_2) \), this means that there is a monomorphism \( h_1 : AC_1 \to AG \). On the other hand, this also means that there is a monomorphism \( h_2 : AC_2 \to AG \) such that \( h_1 \circ m = h_2 \circ c \), since \( AG \models \forall (c : AX \to AC_2) \). Hence, by Proposition 2 there is a factorization:

\[
\begin{array}{ccc}
AC_1 & \xrightarrow{f_1} & h_1 \\
\downarrow & & \downarrow \\
AX & \xrightarrow{c} & AG \\
\downarrow & & \downarrow \\
AG' & \xrightarrow{m'} & AG \\
\downarrow & & \downarrow \\
AC_2 & \xrightarrow{f_2} & h_2 \\
\end{array}
\]

where \( f_1 : AC_1 \to AG' \) and \( f_2 : AC_2 \to AG' \) are jointly surjective, which means that \( AG \) is in \( \bar{g} \), and \( m' \) is injective. Hence, \( AG \models \exists AG' \). Moreover, \( m' \circ f_1 \circ m = h_1 \circ m = h_2 \circ c = m' \circ f_2 \circ c \) and, since \( m' \) is injective, \( f_1 \circ m = f_2 \circ c \), which means that \( AG' \in \bar{g} \). This implies that \( AG \models (\bigvee_{AG' \in \bar{g}} \exists AG') \). \( \square \)

**Proof of Lemma 2.** Let \( \Gamma \) be a clause in \( C \). We have to prove that \( AG \models \Gamma \). Since the sequence is saturated there should be a literal \( L \) in \( \Gamma \) such that the conditions (a), (b), or (c) in Definition 8 are satisfied. We consider each case:

(a) if \( L = \neg \exists AC \), then for every \( j \) there is no monomorphism \( m : AC \to AG_j \). But, by Proposition 3, there is no monomorphism \( h : AC \to AG \). Therefore \( AG \models \neg \exists AC \) and as a consequence \( AG \models \Gamma \).

(b) if \( L = \exists AC \), there is a \( j \), such that there is a monomorphism \( m : AC \to AG_j \), i.e. there is a monomorphism \( f_j \circ m : AC \to AG \). Therefore \( AG \models \exists AC \) and as a consequence \( AG \models \Gamma \).
(c) If \( L = \forall(c : X \rightarrow AC) \) then for every \( i \) and every monomorphism \( m_0 : X \rightarrow AG_i \) there is a \( j \), with \( i < j \), such that there is a monomorphism \( h : AC \rightarrow AG_j \) with \( h_{AG_i \rightarrow AG_j} \circ m_0 = h \circ c \). Suppose that there is a monomorphism \( m : X \rightarrow AG \). According to Proposition 3, there exists an \( i \) such that there is a monomorphism \( m' : X \rightarrow AG_i \), with \( f_i \circ m' = m \). Then, there is a \( j \), with \( i < j \), such that there is a monomorphism \( h : AC \rightarrow AG_j \) with \( h_{AG_i \rightarrow AG_j} \circ m' = h \circ c \). Hence, \( f_j \circ h : AC \rightarrow AG \) and \( f_j \circ h \circ c = f_j \circ h_{AG_i \rightarrow AG_j} \circ m' = f_i \circ m' = m \). Therefore, \( AG \) satisfies \( \forall c : X \rightarrow AC \) and as a consequence \( AG \models \Gamma' \). □

Proof of Lemma 3. Given a symbolic graph \( AG \) and a set of clauses \( C \) let \( I(AG, C) \) be a finite set of monomorphisms \( h : AC \rightarrow AG' \) satisfying that for every clause \( \Gamma \in C \) there is a literal \( L \in \Gamma \) such that:

(a) if \( L = \neg \exists AC \), then there is no monomorphism \( m : AC \rightarrow AG' \)
(b) if \( L = \exists AC \) there is a monomorphism \( m : AC \rightarrow AG' \)
(c) If \( L = \forall(c : X \rightarrow AC) \) then for every monomorphism \( m : X \rightarrow AG \) there is a monomorphism \( g : AC \rightarrow AG' \) with \( h \circ m = g \circ c \).

In Orejas et al. (2009) we may see how the sets \( I(AG, C) \) are built explicitly, essentially by applying (in all possible ways) inference rules (R2) and (R3) to the literal \( \exists AC \) and each positive literal in each clause in \( C \).

Now, let us suppose that there is a literal \( \exists AC \) in \( BasPosLit(\bigcup_{k \geq 1} C_k) \). We define a sequence \( \exists AC_1 < \cdots < \exists AC_i < \cdots \) in \( BasPosLit(\bigcup_{k \geq 1} C_k) \) as follows:

- \( AG_1 = AC \).
- If \( h : AG_j \rightarrow AG' \) is a monomorphism in \( I(AG_j, C_j) \), then we define \( AG_{j+1} = AG' \) and, hence, for \( C \) as a consequence, the sequence \( \exists AC_1 < \cdots < \exists AC_j \) is saturated.
- Otherwise, the sequence is infinite and, for every \( j \), \( \exists AC_j \) is open. Moreover, it can be shown that the definition of the sets \( I(AG_j, C_j) \) ensures that the sequence \( \exists AC_1 < \cdots < \exists AC_j < \cdots \) is saturated. □

Proof of Lemma 4. We define inductively the sequence of clauses \( \Gamma_1, \ldots, \Gamma_n, \ldots \) where:

- \( \Gamma_1 = \Gamma' \).
- \( \Gamma_{n+1} = (\bigvee_{AG \in g_{n+1}} \exists AC) \), where \( g_{n+1} = \{AG \mid \text{there is a literal } \exists AC' \in \Gamma_n \text{ with } (h : AG' \rightarrow AG) \in I(AG', C_n) \} \).

By construction, for every literal \( \exists AC \) in \( \Gamma_n \), \( I(AC, C_n) \) is finite. As a consequence, if for every \( i \) there is an open literal included in \( \Gamma_i \) this means that there should be an infinite sequence of open literals \( \exists AC_1 < \cdots < \exists AC_n < \cdots \) where each \( AG_n \in \Gamma_n \) and \( h_{AG_n \rightarrow AG_{n+1}} \in I(AG_{n+1}, C_n) \). But this sequence would be saturated against our original assumption. Therefore, there should exist an \( i \) where all the literals in \( \Gamma_i \) are closed. So it is enough to define \( \Gamma' = \Gamma_i \). □

Proof of Lemma 5. We have to show the existence of a graph \( AG \) such that \( AG \models C \) if the empty clause is not in \( C \) for any \( j \). We consider four cases:

1. There is no clause \( \Gamma \in C \) consisting only of basic positive literals. This means that every clause \( \Gamma \) includes a negative literal \( \neg \exists AC \) or a non-basic literal \( \forall(c : X \rightarrow AC) \), where \( X \) is not empty. In this case, the empty graph, i.e. \( (\emptyset, true) \), satisfies all these atomic and negative literals and, as a consequence, is a model for \( C \).
2. Otherwise, we have a clause \( \Gamma \in C \) consisting only of basic positive literals. Then, by Lemma 3, we know that there exists at least one saturated sequence in \( BasPosLit(\bigcup_{k \geq 1} C_k) \). By Definition 8, we have the following cases:

- If \( L = \neg \exists AC \), then there is no monomorphism \( m : AC \rightarrow AG' \).
- If \( L = \exists AC \) there is a monomorphism \( m : AC \rightarrow AG' \) with \( h \circ m = g \circ c \).
- If \( L = \forall(c : X \rightarrow AC) \) then for every monomorphism \( m : X \rightarrow AG \) there is a monomorphism \( g : AC \rightarrow AG' \) with \( h \circ m = g \circ c \).
(a) Every saturated sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ is finite and its last element is a closed literal. This case is not possible. Let $\exists \text{AG}_1 \lor \Gamma$ be a minimal clause in $\bigcup_{k \geq 1} C_k$ consisting only of closed literals (according to Lemma 4 we know that such a clause must exist and, according to our assumption, it must not be empty). Since we are assuming that $\exists \text{AG}_1$ is closed, there should exist a clause $\neg \exists \text{AC}_1 \lor \cdots \lor \neg \exists \text{AC}_n$ in $\bigcup_{k \geq 1} C_k$ such that for every $i$ there is a monomorphism $m_i : \text{AC}_i \rightarrow \text{AG}_1$. Using rule (R1) we can infer $\Gamma \lor \neg \exists \text{AC}_2 \lor \cdots \lor \neg \exists \text{AC}_n$. Then, using again rule (R1) with this clause and the clause $\exists \text{AG}_1 \lor \Gamma$, we can infer $\Gamma \lor \Gamma \lor \neg \exists \text{AC}_3 \lor \cdots \lor \neg \exists \text{AC}_n = \Gamma \lor \neg \exists \text{AC}_3 \lor \cdots \lor \neg \exists \text{AC}_n$. Then, applying repeatedly rule (R1) in a similar way, we finally infer $\Gamma$, against the assumption that $\exists \text{AG}_1 \lor \Gamma$ was minimal.

(b) There is a finite saturated sequence in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$ whose last element is $\text{AG}$. Then $\text{AG} \models C$.

(c) There is an infinite saturated sequence $\exists \text{AG}_1 \prec \cdots \prec \exists \text{AG}_i \prec \cdots$ in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$. Then, according to Lemma 2, its colimit is a model for the given set of clauses. 

Proof of Lemma 6. The soundness of (R1), (R2), and (R3) implies their strong soundness. Let us now consider (R4). If $AG = \langle G, \Phi' \rangle \models \exists \langle C, \Phi \rangle \lor \Gamma$ and $\Phi'$ is satisfiable. Then there cannot exist a monomorphism $h : \langle C, \Phi \rangle \rightarrow \langle G, \Phi' \rangle$ since $SP_D \models \Phi' \Rightarrow h(\Phi)$ and the unsatisfiability of $\Phi$ implies the unsatisfiability of $\Phi'$ against the hypothesis. Therefore, $AG \models \Gamma$. 

Proof of Lemma 7. Suppose that the empty clause is not in $C_j$ for any $j$. We have to show the existence of a graph $AG$ such that $AG = \langle G, \Phi \rangle \models C$ and $\Phi$ is satisfiable. As in the proof of Lemma 5, we can consider four cases. The first three cases are essentially similar. With respect to the fourth case, if there is an infinite saturated sequence $\exists \langle G_1, \Phi_1 \rangle \prec \cdots \prec \exists \langle G_i, \Phi_i \rangle \prec \cdots \prec \exists \langle G_j, \Phi_j \rangle \prec \cdots$ in $\text{BasPosLit}(\bigcup_{k \geq 1} C_k)$, where all its elements are open. Then, according to Lemma 2, its colimit $AG = \langle G, \bigcup_{i \geq 1} h_i(\Phi_i) \rangle$ is a model for the given set of clauses. Moreover, $\bigcup_{i \geq 1} h_i(\Phi_i)$ is satisfiable. Otherwise, by compactness of first-order logic, there is a finite subset $\{h_1(\Phi_1), \ldots, h_i(\Phi_i)\}$ which is unsatisfiable, implying that $h_1(\Phi_1)$ is unsatisfiable, against the hypothesis that $\langle G_j, \Phi_j \rangle$ is open.

References

APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems, pp. 171–178.

pp. 313–400.


Sci. EATCS 82, 175–190.

Ehrig, H., Ehrig, K., Prange, U., Taentzer, G., 2006a. Fundamental theory of typed attributed graph transformation based on

of Theoretical Computer Science. Springer.

pp. 87–100.


Workshop on Applied Graph Transformation. pp. 11–22.


1–33.


Naeem, M., Heckel, R., Orejas, F., Hermann, F., 2010. Incremental service composition based on partial matching of visual
