On the expressive power of algebraic graph grammars with application conditions

Annika Wagner

Technical University of Berlin
Computer Science Department
Sekr. FR 6 - 1, Franklinstr. 28/29, D-10587 Berlin
e-mail: aw@cs.tu-berlin de

Abstract. In this paper we introduce positive, negative and conditional application conditions for the single and the double pushout approach to graph transformation. To give the reader some intuition how the formalism can be used for specification we consider consistency and an interesting representation for specific conditions, namely (conditional) equations. Using a graph grammar notion without nonterminal graphs, i.e. each derivation step leads to a graph of the generated language, we prove a hierarchy: graph grammars over rules with positive application conditions are as powerful as the ones over rules without any extra application condition. Introducing negative application conditions makes the formalism more powerful. Graph grammars over rules with conditional application conditions are on top of the hierarchy.

1 Introduction

Graph transformations are a good means for describing the development of structured states in elementary steps. Although the generative power of most of the known graph grammar approaches is sufficient to generate any recursively enumerable set of graphs, additional application conditions are a necessary part of every non-trivial specification. Often they are expressed informally by assuming some kind of control mechanism or they are coded into the graphs using flags and additional labels. Both possibilities make it hard to analyse the specification. In contrast to textual application conditions expressed in logical formulas (see e.g. [12], [13]) we propose a formalism where these additional and more complex application conditions are treated *formally and graphically*. In order to show that we can really express more with our new notion(s), i.e. that we extended the *expressive power* of the graph grammar approach, we consider specification aspects on one hand and the generative power one the other hand. The main result concerning specification is that specific positive (negative) application conditions can be characterized by (non-)equations while specific conditional application conditions correspond to conditional equations. As stated above the generative power of the pure algebraic approach is sufficient for every recursively enumerable set of graphs. Hence in order to achieve a new illustrative result we had to change the notion of a graph grammar not to contain non-terminal graphs, i.e. every derivation step leads to a graph of the generated language. This makes it nearly impossible to encode control structure into flags and labels of a graph. Using the formalism to formalize algorithms or to describe the operational semantics of complex systems hence leads to more abstract specifications.

The algebraic approach to graph transformation [1], i.e. the single and the double pushout approach ([11], [2]), provides a framework where because of its categorical nature a lot of different results (concerning parallelism and concurrency for example) have been achieved. The extensions presented in this paper are of the same nature. As presented in [6] additional algebraic application conditions are compatible with that theory. High-level replacement systems ([4], [5]) are a categorical generalization of the algebraic approach to graph grammars. In this context the same notions are formulated not only for graphs but for objects of arbitrary categories. Because of their nature all notions introduced in this paper can easily transfered to high-level replacement systems. Furthermore application conditions can close the gap between the pure single pushout approach and the more restrictive double pushout approach, allowing specific "gluing conditions" for arbitrary subgraphs of the left hand side of productions.

The paper is organized as follows: In section 2 we introduce all basic notions. Then we change over to the expressive power of the formalism. Section 3 is devoted to the specification aspects, while our notion of graph grammar and its generative power is presented in section 4. In the last section we conclude with a general discussion of the presented approach and possibilities for further development.

The reader is assumed to be familar with basic notions of category theory and universal algebra (see e.g. [9] or [1]).

2 Basic notions and definitions

In this section we first recover some basic notions of the single and the double pushout approach to graph transformation. Then we introduce positive, negative and conditional application conditions for rules and define their applicability.

Definition 1 (Graph, Morphism). A *graph* $G = (G_V, G_E, s^G, t^G)$ consists of a set of vertices G_V, a set of edges G_E and two mappings $s^G, t^G : G_E \rightarrow G_V$ which provide a source resp. target vertex for every edge. A *graph morphism* $f : G \rightarrow H$ is a pair of total mappings $(f_V : G_V \rightarrow H_V, f_E : G_E \rightarrow H_E)$ which are compatible with the source and target assignments, i.e. $f_V(s^G(e)) = s^H(f_E(e))$ for all $e \in G_E$ (and analogously for the target mapping). A *partial graph morphism* g from G to H is a total graph morphism from some subgraph $G(g)$ of G to H.

If we define composition of these morphisms by composition of the components and identities as pairs of component identities, the objects and (partial) morphisms w.r.t. definition 1 form a category denoted by $\underline{GRA}$ ($\underline{GRA^P}$) in the following. If we want to state a property of a morphism we often do not divide between vertices and edges. Hence we write $f(x)$ for all objects x instead of $f_V(v)$ for all vertices v and $f_E(e)$ for all edges e.

Generally, the rewriting of a graph G via some rule r is done by first deleting some part DEL (from G) and then adding a new part ADD finally resulting in a derived graph H. Often this shall only be done if G additionally contains some

[1] Note that although "algebraic" is frequently used as synonymous of "context-free" in language theory, in this paper we use it in a different sense.

context K which is essentially to be kept. We additionally allow that the context is glued together. Thus rewriting means to replace some part $L = DEL \cup K_L$ by another one $R = K_R \cup ADD$, where K_R is constructed from K_L by gluing of vertices and edges.

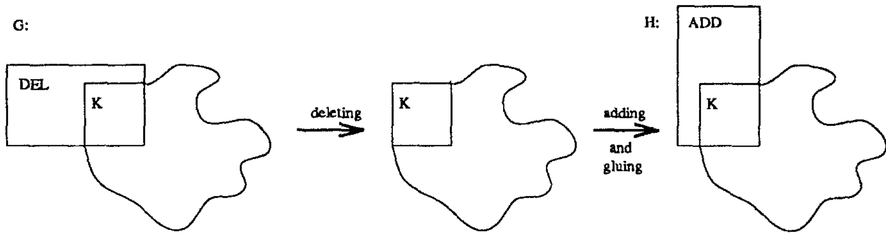

Fig. 1. How graph transformation works

Consequently a rule consists of two graphs L and R, called left and right hand side resp. (the square parts in figure 1), together with a partial graph morphism $r : L \rightarrow R$ which maps the context contained in the left hand side to the context of the right hand side of the rule. All vertices and edges for which r is not defined are intended to be deleted. Applying a rule means to match the left hand side with a subgraph of the mother graph (here we allow that different items from L are mapped onto the same item in G), then to remove it and finally to add the right hand side. The result is the daughter graph H.

Deleting is not always unproblematic. For example if a vertex shall be deleted by the rule but the mother graph G contains an edge pointing to that vertex. Another kind of conflict may arise if parts which shall be deleted and parts which shall be preserved are identified in G. In general we have two possibilities to handle these problems. First we can forbid the application of the rule in such cases and second we can solve all conflicts destructively by deleting the conflicting items. The first alternative is choosen in the 'classical' double-pushout approach, the second in the framework of single-pushout transformations.

Definition 2 (Simple rule, simple derivation). A *simple rule* $r : L \rightarrow R$ is a partial morphism from it's left hand side L to it's right hand side R. A *match* for r in some object G is a total morphism $m : L \rightarrow G$ from the left hand side of the rule to g. A match m is called *d - injective* if $m(x) = m(y)$ implies $x = y$ or $x, y \in L(r)$. It is called *d - complete* if for each edge $e \in G_E$ with $s^G(e), t^G(e) \in m_V(L_V - L(r)_V)$ we have $e \in m_E(L_E - L(r)_E)$. Given a simple rule r and a match m for r in a graph G the *simple direct derivation* from G with r at m, written $G \stackrel{r,m}{\Longrightarrow} H$, is the pushout of r and m in $\underline{GRA^P}$. If the match m is d-injective and d-complete we call a direct derivation *classical*. A sequence of direct derivations of the form $G_0 \stackrel{r_1,m_1}{\Longrightarrow} \dots \stackrel{r_k,m_k}{\Longrightarrow} G_k$ constitutes a *derivation* from G_0 to G_k by $r_1, \dots, r_k$. Such a derivation is denoted by $G_0 \stackrel{*}{\Longrightarrow} G_k$.

The following construction shows how the direct derivation of a graph can be achieved.

Construction 3 (Pushout in $\underline{GRA^P}$). If $f : A \to B$ and $g : A \to C$ is a pair of (partial) morphisms the pushout $(D, f^* : C \to D, g^* : B \to D)$ of f and g in $\underline{GRA^P}$ can be constructed in three steps:

1. Construction of the gluing graph: Let E be the largest subgraph of $A(f)$ such that for all vertices and edges $x \in E$ and $y \in A : f(x) = f(y)$ or $g(x) = g(y) \Longrightarrow y \in E$.
2. Construction of the definedness areas $B(g^*)$ and $C(f^*)$ of g^* and f^*: Delete all vertices and edges from C that have preimages in A but not in E and all edges whose source or target vertices are deleted. Symmetrically, treat B.
3. Gluing of graphs $B(g^*)$ and $C(f^*)$ along E: Now graph D is constructed by the disjoint union of $B(g^*)$ and $C(f^*)$ where corresponding images of vertices and edges in E are identified with each other.

The proof for the more general case of graph structures can be found in [11]. Some essential properties of the pushout diagram directly follow from the construction.

Lemma 4 (Properties of pushouts in $\underline{GRA^P}$). *Let $(D, f^* : C \to D, g^* : B \to D)$ be the pushout of $f : A \to B$ and $g : A \to C$ in $\underline{GS}$. Then the following properties are fulfilled:*

1. $v \notin C(f^*)_V \Longrightarrow v \in g(A(g)_V)$
2. $f^*(x) = g^*(y) \Longrightarrow x \in g(A)$ *and* $y \in f(A)$
3. f^* *and* g^* *are together surjective*

Proof. Property (1) is a direct consequence of step (2) of construction 3. The second and the third property follow from step (3) of the construction. □

The following technical lemma is used in later sections to show that the presented results can also be achieved if one is restricted to classical matches.

Lemma 5 (Embedding of derivations). *If $G \stackrel{r,m}{\Longrightarrow} H$ is a direct derivation with rule $r : L \to R$ it can be embedded into a bigger context G', i.e. every inclusion $i : G \to G'$ induces a match $i \circ m$ for the rule r. If m is classical and $\forall e \in (G'_E - i_E(G_E)) : s^{G'}(e), t^{G'}(e) \in i(G) \Longrightarrow s^{G'}(e), t^{G'}(e) \in i(G - m(L))$ or $s^{G'}(e), t^{G'}(e) \in i(m(L(r)))$ then $i \circ m$ is classical, too.*

Proof. $i \circ m$ is obviously a match for r. We only prove that it is classical under the stated condition. $i \circ m$ is d-injective because i is injective and m is d-injective. Now suppose that $s^{G'}(e') \in i_V(m_V(L_V - L(r)_V))$ for $e' \in G'_E$. With the above condition we get that $e' \in i_E(G_E)$. Because m is assumed to be d-complete $i \circ m$ is d-complete, too. □

Remark (Classical derivations). Rules in the double-pushout framework look quite different from those in the single pushout framework (definition 2) but they can be transformed into each other. Then for each transformation in the double-pushout approach there exists a transformation as defined in 2 in the single-pushout approach using the same match and achieving the same result. Conversely a single-pushout derivation has an equivalent one in the double-pushout framework if and only if it is classical. Hence the double-pushout framework is a very elegant formulation

of graph transformation with a *special application condition*: the gluing condition, which corresponds to d-injectivity and d-completeness of the match in the sigle-pushout framework. This is formally proven in [11].

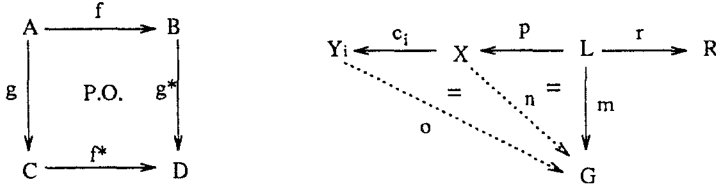

Fig. 2. Pushout diagram and satisfaction of conditional constraint

Now we want to consider more complex rules with application conditions. The general idea is to have a left hand side not only consisting of one graph but of several ones connected by morphisms. Positive and negative application conditions have already been investigated in [6] and [8]. Conditional application conditions consist of a premise and a conclusion which both are more or less simple positive application conditions.

Definition 6 (Application condition). Let $r : L \to R$ be a simple rule. A *simple constraint* s for r is total morphism $s : L \to X$. A *conditional constraint* for r $(p : L \to X, (c_i : X \to Y_i)_{i=1..n})$ is a pair consisting of a simple constraint p and a (possibly empty) family of total morphisms c_i. A *positive (negative)* resp. *conditional application condition* $A(r)$ for r consists of a finite set of simple resp. conditional constraints.

Definition 7 (Satisfaction of conditions). A total morphism $m : L \to G$ *p-satisfies* a simple constraint $s : L \to X$, written $m \models_p s$, if there exists a total morphism $n : X \to G$ such that $n \circ s = m$. m *n-satisfies* s if it does not p-satisfy s, i.e. $m \models_n s \iff m \not\models_p s$. Furthermore, we say that m *c-satisfies* a conditional constraint $cc = (p : L \to X, (c_i : X \to Y_i)_{i=1..n})$,written $m \models_c cc$, if for all total morphisms $n : X \to G$ with $n \circ p = m$ there exists a total morphism $o : Y_i \to G$ with $o \circ c_i = n$ for at least one $i \in \{1..n\}$. m *satisfies* a positive application condition if it p-satisfies at least one simple constraint of the condition. m *satisfies* a negative (conditional) application condition if it n-satisfies (c-satisfies) all negative (conditional) constraints the condition consists of.

Definition 8 (Conditional rule, derivation). A *p- (n-, c-) conditional rule* $\hat{r}$ is a pair $(r : L \to R, A(r))$ consisting of a simple rule r and a positive (negative, conditional) application condition. $\hat{r}$ is *applicable* to a graph G if there exists a match $m : L \to G$ for r that satisfies $A(r)$. If $\hat{r}$ is applicable to G via m the direct conditional derivation of G to H is the simple direct derivation.

Example 1 (Gluing condition). For each rule $r : L \to R$ the gluing condition of matches (d-injectivity and d-completeness) can be expressed using a conditional application condition. Because of a lack of space we do not show the general case,

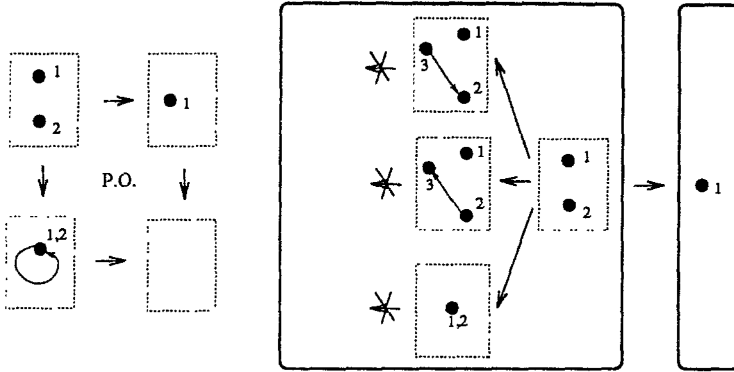

Fig. 3. Simulation of the gluing condition

but the very simple example in figure 3 demonstrates the idea. On the left hand side a single pushout derivation with a simple rule is shown where the match is not classical. On the right hand side the rule is equipped with three conditional application conditions. Hence it's left hand side is blown up. The arrows crossed out indicate an empty conclusion. The first two conditions make sure that the match is d-complete and the third one is dedicated to the d-injectivity of the match. If the rule contains edges the application conditions for the d-completeness have a non-empty conclusion.

3 Specification aspects

In this section we want to give the reader some intuition how application conditions can be used for specification. For this purpose we first show how different combinations of constraints can be put together in application conditions. Furthermore application conditions which contain surjective constraints only are characterized by (conditional) equations.

3.1 Combination of constraints

In definition 7 we define the satisfaction of a positive application condition to be a disjunction of constraints while we use a conjunction in the case of negative resp. conditional application conditions. This is motivated by the fact that the other combinations of constraints can be achieved by gluing the different constraints together as it is demonstrated in the following.

Definition 9 (Combination of constraints). If $s_i : L \to X_i$ ($c_i = (p_i : L \to X_i, (c_{ij} : X_i \to Y_{ij})_{j=1..k_i})$) for $i = 1,2$ are simple (conditional) constraints for a rule $r : L \to R$, their combination is given by single simple (conditional) constraint $s = s_2^* \circ s_1$ ($c = (p = p_2^* \circ p_1, (c^*_{ij} : X \to Y'_{ij})_{i=1,2;j=1..k_i})$) s.t. (1) (1, (2) and (3)) in the left (right) diagram of figure 4 become(s) a pushout.

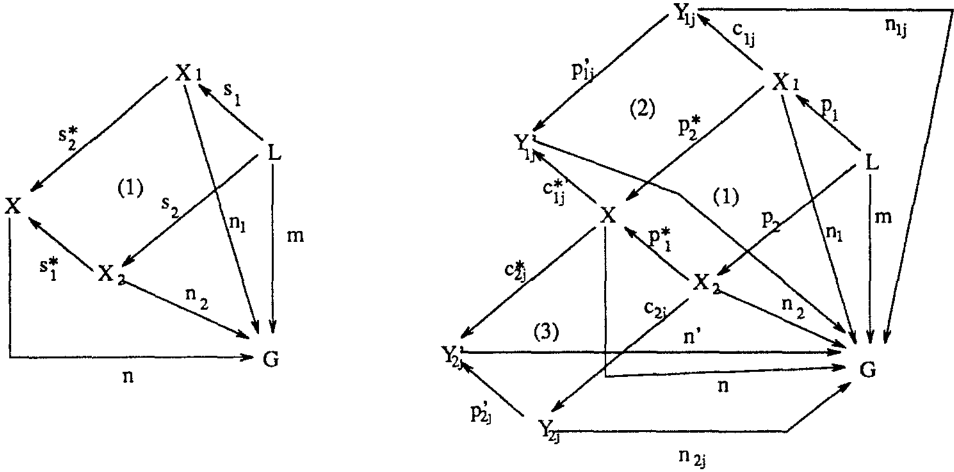

Fig. 4. Combination of constraints

Proposition 10 (Combination of constraints). *Let s (c) be the combination of two simple (conditional) constraints s_1 and s_2 resp. c_1 and c_2 as defined above. Then for all matches $m : L \to G$ we have*

1. $m \models_p s_1 \wedge m \models_p s_2 \iff m \models_p s$,
2. $m \models_n s_1 \vee m \models_n s_2 \iff m \models_n s$.
3. $m \models_c c_1 \vee m \models_c c_2 \iff m \models_c c$.

Proof. 1./2. : We show that $\exists n : X \to G$ with $n \circ s = m \iff \exists n_i : X_i \to G$ with $n_i \circ s_i = m$ for $i = 1, 2$. Then the assertion follows from definition 7. Given n as above we have $n_1 = n \circ s_2^*$ and $n_2 = n \circ s_1^*$, where $m = n \circ s = n \circ s_2^* \circ s_1 = n_1 \circ s_1$ (and similarily for s_2). Given n_1, n_2 as above, the left hand side of the equivalence follows from the universal property of X.

3. : Analogously to the above proof we get that $\exists n : X \to G$ with $n \circ p = m \iff \exists n_i : X_i \to G$ with $n_i \circ p_i = m$ for $i = 1, 2$. With the same arguments for the pushouts (2) and (3) we get $\exists n'_{ij} : Y'_{ij} \to G$ with $n'_{ij} \circ c^*_{ij} = n \iff \exists n_{ij} : Y_{ij} \to G$ with $n_{ij} \circ c_{ij} = n_i$ for $i = 1, 2$ and $j = 1..k_i$. Then the proposition 10 3. follows from definition 7. □

3.2 Surjective constraints

In this section we show that special application conditions, namely those which contain surjective constraints only, can be characterized using (conditional) equations. This seems to be useful, because the graphical representation of complex application conditions is rather hard to understand, but nice if one wants to achieve theoretical results.

Definition 11 ((Conditional) Equations). An *equation* E for a graph G is a pair $(a = b)$ such that a and b are elements of the same domain of G. A *conditional equation* CE for G is a pair $(\mathcal{P}, \mathcal{C})$ consisting of two finite sets of equations $\mathcal{P}$ and $\mathcal{C}$

called premise and conclusion respectively. A morphism $f : G \to H$ is a *solution for a set of equations* $\mathcal{E}$ for G if $f(a) = f(b)$ for all $(a = b) \in \mathcal{E}$. f is a *solution for the conditional equation* CE for G if it is a solution for all equations in the premise $\mathcal{P}$ and at least one equation in the conclusion $\mathcal{C}$ or if it is not a solution for any of the equations in $\mathcal{P}$. A *solution for a set of conditional equations* $\mathcal{CE}$ is a solution for all conditional equations $CE \in \mathcal{CE}$.

First we want to consider simple constraints and negative application conditions. Because of their definition the reader can easily transfer the result to positive application conditions.

Proposition 12 (Equational constraints). *Let $c : L \to X$ be a simple surjective constraint for a rule $r : L \to R$ such that the congruence $Eq(c)$ induced by c is equal to the congruence generated by a set of equations $\mathcal{E}$ for L. Then, given a match $m : L \to G$, m p-satisfies (n-satisfies) c if and only if m is (not) a solution for $\mathcal{E}$. We say that $\mathcal{E}$ represents c.*

For the proof of this proposition we refer to [8].

Corollary 13 (Negative application condition). *Given an n-conditional rule $\hat{r} = (r : L \to R, A(r))$ and sets of equations $\mathcal{E}_c$ such that each $\mathcal{E}_c$ represents a simple constraint $c \in A(r)$. Then a match $m : L \to G$ for r satisfies $A(r)$ if and only if m is not a solution for any $\mathcal{E}_c$.*

The proof directly follows from the definitions 7 resp. 11 and proposition 12.

If negative application conditions correspond to non-equations it seems to be natural that conditional ones correspond to conditional equations.

Proposition 14 (Conditional constraints). *Let $c = (p : L \to X, c_i : X \to Y_i)_{i=1..n}$ be a conditional constraint for a rule $r : L \to R$ with p being surjective and c_i being total and surjective for $i = 1..n$ such that the congruence $Eq(p)$ ($Eq(c_i \circ p)$) induced by p ($c_i \circ p$) is equal to the congruence generated by a set of equations $\mathcal{E}_p$ ($\mathcal{E}_{c_i op}$) for L. $\Pi\mathcal{E}$ denotes the cartesian product of $\mathcal{E}_{c_i op}$ for $i = 1..n$. $F : \Pi\mathcal{E} \to Set$ denotes the function assigning to each tuple $(E_1, .., E_n)$ the set $\{E_1, .., E_n\}$. Now, given a match $m : L \to G$, m c-satisfies c if and only if m is a solution for the set of conditional equations $\mathcal{CE} = \{(\mathcal{E}_p, F(E)) | E \in \Pi\mathcal{E}\}$. We say that $\mathcal{CE}$ represents c.*

Proof. If $m \models_c c$ and there is no morphism $n : X \to G$ with $n \circ p = m$ then by proposition 12 m is not a solution for $\mathcal{E}_p$ and hence m is a solution for $\mathcal{CE}$. Now assume that there is a morphism $n : X \to G$ with $n \circ p = m$ and m is a solution for $\mathcal{E}_p$. Then there exists $o : Y_i \to G$ with $o \circ c_i = n$ for at least one $i \in \{1..n\}$. By proposition 12 m is a solution for $\mathcal{E}_{c_i op}$. For each $CE \in \mathcal{CE}$ the conclusion contains one equation $E \in \mathcal{E}_{c_i op}$. Hence by definition 11 m is a solution for $\mathcal{CE}$.

Now assume that m is a solution for $\mathcal{CE}$. Again we only consider the case that m is a solution for $\mathcal{E}_p$. For the other case we refer to proposition 12. m must be a solution for all conclusions of conditional equations in $\mathcal{CE}$. This implies that m is a solution for at least one $\mathcal{E}_{c_i op}$. The existence of $o : Y_i \to G$ with $o \circ (c_i \circ p) = m$ follows from proposition 12. □

Corollary 15 (Conditional application condition). *Given a c-conditional rule $\hat{r} = (r : L \rightarrow R, A(r))$ and sets of conditional constraints $\mathcal{CE}_c$ such that each $\mathcal{CE}_c$ represents a conditional constraint $c \in A(r)$. Then a match $m : L \rightarrow G$ for r satisfies $A(r)$ if and only if m is a solution for $\bigcup_c \mathcal{CE}_c$.*

The proof directly follows from the definitions 11 resp. 7 and proposition 14.

4 Generative Power

The notion of a grammar as it is used in formal language theory deals with terminal and nonterminal objects. In the literature this is carried over to graph grammars leading to terminal and nonterminal graphs. Using graph transformations for system specification it makes sense to forbid nonterminal graphs in order to make it impossible to encode control structure into flags and additional labels. With this background we define graph grammars and their generated language.

Definition 16 (Graph grammar). A graph grammar $GG = (S, \mathcal{R})$ consists of a starting graph S and a finite set of rules $\mathcal{R}$. The language $L(GG)$ generated by the graph grammar GG is the set of all those graphs which can be derived from the starting graph S using rules of $\mathcal{R}$.

Note that the notion of graph grammars and their generated language is parametric over the notions of rules and derivations. We say $GG = (S, \mathcal{R})$ is a graph grammar over simple resp. p- (n-, c-) conditional rules if $\mathcal{R}$ contains simple resp. p- (n-, c-) conditional rules only. In this paper most of the results can be achieved for the single and the (classical) double pushout approach. Hence we denote the corresponding languages by $L^P(GG)$, $L^N(GG)$ and $L^C(GG)$ if no distinction is necessary whether the used matches are classical or not. Only if we want to indicate the match type we use additional subscripts c resp. $\not c$ (for classical resp. non-classical). $\mathcal{L}^S$, $\mathcal{L}^P$, $\mathcal{L}^N$ and $\mathcal{L}^C$ are the classes of all graph languages generated by graph grammars over simple, p-, n- resp. c-conditional rules.

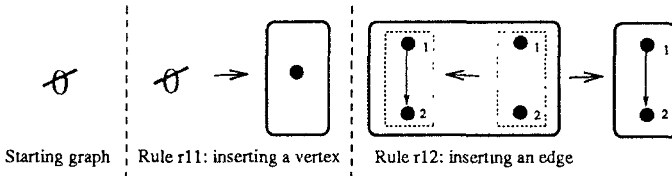

Fig. 5. Graph grammar $GG1$

Example 2 (Graph grammar over n-conditional rules). The graph grammar $GG1$ consisting of the starting graph $S1$ and the rules $r11$ and $r12$ as depicted in figure 5 generates the set of all finite graphs which have at most one edge between two vertices in each direction. We call this graph language $L1$ in the following. Rule $r11$ adds a new vertex in any situation. The negative application condition of rule $r12$ makes sure that there is not already an edge from $m(1)$ to $m(2)$ if a new edge

with the same direction between these two vertices is inserted. Note that all matches for the rules $r11$ and $r12$ are classical due to the fact that the rule morphisms are total.

Proposition 17 ($GG1$ **generates** $L1$). *The graph grammar $GG1$ from example 2 generates all and only those graphs in which for each pair of vertices $(v1, v2)$ there is at most one edge with source vertex $v1$ and target vertex $v2$, i.e. $L^N(GG1) = L1$.*

Proofidea. $L^N(GG1) \subseteq L1$ can easily be shown by induction over the length of the derivation sequence and $L1 \subseteq L^N(GG1)$ analogously by induction. over the number of objects (vertices and edges) in a graph $G \in L1$. □

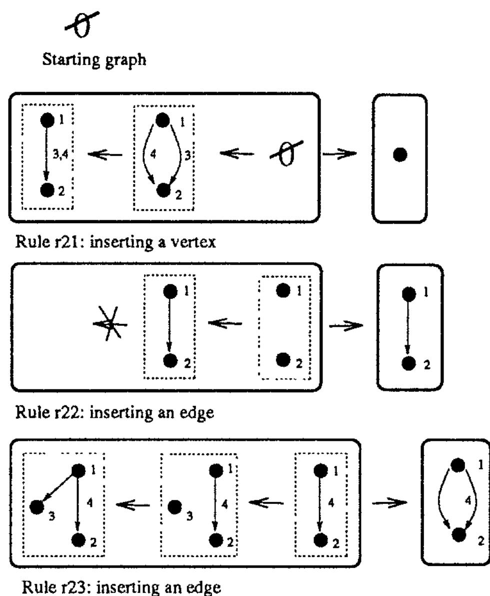

Fig. 6. Graph grammar $GG2$

Example 3 (Graph grammar over c-conditional rules). The graph grammar $GG2$ consisting of the starting graph $S2$ and the rules $r21$, $r22$ and $r23$ as depicted in figure 6 generates the set of all finite graphs which fulfill the following condition: If there are two edges with the same source vertex v and the same target vertex, there exists an edge from v to every vertex in the graph. We call this language $L2$ in the following.

Rule $r21$ adds a new vertex if there is no vertex with more than one outgoing edge to the same target vertex. Note that the edges in the application condition are identified. New edges are inserted by the rules $r22$ and $r23$ where $r23$ inserts an edge if the vertex becoming the source of this edge has an outgoing edge to every vertex in the graph. Rule $r22$ is applicable to vertices $m(1)$ and $m(2)$ if there is no edge from $m(1)$ to $m(2)$. Note that there is a certain application order for the rules of $GG2$: If rule $r23$ has been applied once rule $r21$ is not applicable any more. Note

that all matches for the rules $r21$, $r22$ and $r23$ are classical due to the fact that the rule morphisms are total.

Proposition 18 (**$GG2$ generates $L2$**). *The graph grammar $GG2$ from example 3 generates all and only those graphs which have the following property: If there are two edges with the same source vertex v and the same target vertex, there exists an edge from v to every vertex in the graph.*

Proof. We prove $L^C(GG2) = L2$ showing $L2 \subseteq L^C(GG2)$ by induction over the number of objects (vertices and edges) of a graph. It is easy to show $L^C(GG2) \subseteq L2$ by induction over the length of the derivation sequence.

The basis is given by the starting graph which is the only graph within $L2$ having no objects. For the induction step we divide three cases: if $G \in L2$ contains at least two edges with the same source and target vertex, an application of rule $r23$ derives G from a graph with n objects. In the second case we consider graphs that contain edges but none of them have the same source and target vertex. These graphs may have an application of rule $r22$ in their last derivation step. If we have a graph without any edge, rule $r21$ inserts a vertex to the graph with one vertex less. □

4.1 Generative power of positive and negative application conditions

In this section we investigate which classes of graph languages can be generated with graph grammars over rules with positive resp. negative application conditions. It turns out that positive application conditions are just avoiding rule schemes, if one does not restrict the matches to be classical. In contrast rules with negative application conditions really extend the generative power of the graph grammar notion.

Proposition 19 (Generative power of pos. appl. conditions). *For every graph grammar GG over rules with positive application conditions there exists a graph grammar GG' over simple rules such that $L^P_\gamma(GG) = L^S_\gamma(GG')$.*

Proofidea. Let $\hat{r} = (r : L \to R, A(r))$ be a p-conditional rule. Applying the simple rule r to every constraint $c : L \to X$ of $A(r)$ leads to a set $\mathcal{R}$ of simple rules r^* where $(R', r^* : X \to R', c^* : R \to R')$ is the pushout of r and c. For every direct derivation of a graph G to H with the p-conditional rule $\hat{r}$ there exists a direct derivation of G to H with one rule $r^* \in \mathcal{R}$ and vice versa. Note that not every classical match for r which satisfies $A(r)$ induces a classical match for r^*. Induction over the length of the derivation sequence makes sure that $L^P_\gamma(GG) = L^S_\gamma(GG')$. □

A simular proof can be found in [6]. It is a bit more complicated because the notion of satisfaction of a positive application condition is different but the general idea of context enlargement is the same.

In the following we want to show that graph grammars over rules with negative application conditions are more powerful than graph grammars over simple rules. For this purpose we use the graph language $L1$ and the graph grammar $GG1$ introduced in example 2 by showing that $L1$ cannot be generated by any graph grammar over simple rules. The principle of the proof is to state a property of every graph grammar

over simple rules generating $L1$ and show that a grammar with this property cannot generate $L1$.

Lemma 20 (Property of $L1$). *For any possible graph grammar (over simple rules) generating $L1$ there is no derivation sequence containing a single step $G \overset{r,m}{\Longrightarrow} H$ in which a new edge from an existing vertex $v1$ to an existing vertex $v2$ is inserted.*

Proof. Assume such a derivation would exist. Because $H \in L1$ G cannot contain an edge from $v1$ to $v2$. But there exists an inclusion morphism $i : G \rightarrow G'$ where G' is the graph G extended by one edge from $v1$ to $v2$. G' belongs to the language $L1$ and hence has a derivation sequence from the starting graph. $i \circ m$ is a match for r in G' by lemma 5. If m is classical then $i \circ m$ is classical, too, because $v1, v2$ are preserved by the application of rule r. Applying r to G' leads to a graph H' which has two edges with the same source and target vertex. Because there cannot be such a derivation sequence for H' the outdegree of an existing vertex can only be increased by inserting a new vertex. □

Proposition 21 ($L1 \notin \mathcal{L}^S$). *There exists no graph grammar over simple rules that generates the language $L1$, i.e. $L1 \notin \mathcal{L}^S$.*

Proof. Assume that there is a graph grammar (over simple rules) generating $L1$. This graph grammar must certainly fulfill the condition of lemma 20. Let n be the maximum outdegree of a vertex in the right hand side of a rule which has no preimage in the left hand side, i.e. it is inserted if the rule is applied. m is the number of vertices of the starting graph. The graph N with $m + n + 2$ vertices which has exactly one edge from each vertex to any other vertex belongs to $L1$. Note that the outdegree of each vertex is $m+n+1$ and that in order to derive N from the starting graph at least $n + 2$ vertices must be inserted. Within the derivation sequence of N there must be one step where the last time a vertex is inserted. This vertex has an outdegree less or equal to n which is less than $m+n+1$. But because no more vertices are added the outdegree cannot be increased (see the property of the graph grammar above). Hence there is no derivation sequence for N leading to a contradiction. □

Proposition 17 and 21 immediately lead to the following corollary.

Corollary 22 (Generative power of negative appl. conditions). *Graph grammars over rules with negative application conditions are more powerful than graph grammars over simple rules, i.e. $\mathcal{L}^S \subset \mathcal{L}^N$.*

4.2 Generative power of conditional application conditions

In this section we show that graph grammars over rules with conditional application conditions are more powerful than graph grammars over rules with negative application conditions. First we show that for each graph grammar over rules with negative application conditions there exists a graph grammar over rules with conditional application conditions which generates the same language. Furthermore we show that there exists a graph language which can be generated using rules with conditional application conditions but not with negative ones.

Proposition 23 (Simulating negative application conditions). *For every rule $\hat{r} = (r : L \to R, A(r))$ with a negative application condition $A(r)$ there exists a rule $\hat{r}' = (r : L \to R, A(r'))$ with a conditional application condition $A(r')$ such that $\hat{r}$ is applicable to a graph G if and only if $\hat{r}'$ is applicable to G.*

Proof. Choose $A(r') = \{(L \to X, \emptyset) | (L \to X) \in A(r)\}$. [2] Let $m : L \to R$ be a total morphism that satisfies $A(r)$, i.e. for each simple constraint $s : L \to X$ there exists no total morphism $n : X \to G$ with $n \circ s = m$. By definition 7 m c-satisfies each single conditional constraint in $A(r')$ and hence m satisfies $A(r')$. Now assume that m does not satisfy $A(r)$, i.e. there exists a simple constraint $s : L \to X$ and a morphism $n : X \to G$ such that $n \circ s = m$. But because of the empty conclusion of the constraint in $A(r')$ m does not satisfy $A(r')$. □

Now we use the graph language $L2$ and the graph grammar $GG2$ from example 3 to show that introducing conditional application conditions increases the generative power of algebraic graph transformations, i.e. we show that $L2$ cannot be generated by any graph grammar over rules with negative application conditions. Analogously to the previous section we first prove a property such a grammar would have and then show that this is not possible.

Lemma 24 (Property of $L2$). *Every graph grammar $GG = (S, \mathcal{R})$ over rules with negative application conditions that generates the language $L2$ must fullfill the following property for all graphs $G, H \in L2$: If the number of vertices of G is greater than x and $G \Longrightarrow H$ then $V(H) \leq V(G)$.*

Where x is the maximum number of vertices of a left hand side of a rule in $\mathcal{R}$ and $V(G)$ denotes the number of vertices of graph G which have at least two outgoing edges with the same target vertex.

Proof. The stated property holds due to the fact that if $G \overset{\hat{r},m}{\Longrightarrow} H$ and $V(G) < V(H)$ then m must be surjective on the vertices which is not possible if G contains more than x vertices. This fact is shown in the following: Let $r : L \to R$ be the simple rule of $\hat{r}$ and $m : L \to G$ be a match for r which satisfies the negative application condition of $\hat{r}$. $(H, r^* : G \to H, m^* : R \to H)$ is the pushout for the direct derivation of G to H with rule r at match m. If m is not surjective for the vertices there exists a vertex v of graph G which has no preimage under m in L. $V(G) < V(H)$ implies that there is $v' \in H$ with $i \geq 2$ outgoing edges pointing to the same target vertex, what is not true for a possible preimage of v' under r^* in G. By lemma 4 (1) r^* is defined for v. Because $H \in L2$ there exists an edge e from v' to $r^*(v)$. By lemma 4 (2) $r^*(v) \notin m^*(R)$. m^* is a morphism. Hence e has no preimage under m^*. With lemma 4 (3) e and v' have a preimage under r^*. Remember that $r^{*-1}(v')$ has at most one edge pointing to each vertex of graph G. Hence G' which is G without the edge(s) $r^{*-1}(e)$ is also a graph of language $L2$ and can be derived from the starting graph using rules of GG. m induces a match $m' : L \to G'$ for r, because $r^{*-1}(e)$ has no preimage under m. m' satisfies the negative application condition of r because it is satisfied by m. Note that if m is classical m' is classical, too. Applying r to G' at m' leads to a graph H' which is just H without the edge e. H' is not in $L2$, because

[2] Here $\emptyset$ denotes the empty family.

there is no edge from the vertex corresponding to v' to the vertex corresponding to $r^*(v)$. Hence we have a contradiction to the assumption that GG generates $L2$. □

Proposition 25 ($L2 \notin \mathcal{L}^N$). *There exists no graph grammar over rules with negative application conditions that generates the language $L2$, i.e. $L2 \notin \mathcal{L}^N$.*

Proof. Assume that there exists a graph grammar $GG = (S, \mathcal{R})$ over rules with negative application conditions generating $L2$. Furthermore we use the notation of lemma 24. Let o be the number of vertices of the starting graph S and n the maximum number of vertices which can be inserted in one derivation step, i.e. the maximum number of vertices in the right hand side of a rule which have no preimages under the rule morphism. Choose the graph M to have $o+n+1+x$ vertices where for each pair of vertices $v1, v2$ there are exactly two edges with source vertex $v1$ and target vertex $v2$, i.e. $V(M) = o+n+1+x$. $M \in L2$ because every vertex has an outgoing edge to every other vertex. Now consider a derivation sequence $S \stackrel{*}{\Longrightarrow} N \Longrightarrow M$. Within the last derivation step not more than n vertices can be added. Hence N has more than x vertices and we get $V(N) \leq V(M) = o+n+1+x$ by lemma 24. But $V(N)$ cannot be greater than the number of vertices of N and hence N has at least $o+n+1+x$ vertices. With the same argument we can follow backwards the derivation sequence. Because it is finite we sometimes arrive at the first derivation step. But here we know that S only has o vertices such that at least one must be added leading to a contradiction to the assumption. □

Proposition 18 and 25 immediately lead to the following corollary.

Corollary 26 (Generative power of condit. appl. conditions). *Graph grammars over rules with conditional application conditions are more powerful than graph grammars over rules with negative application conditions, i.e. $\mathcal{L}^N \subset \mathcal{L}^C$.*

5 Conclusions

In this paper the concept of positive and negative application conditions as introduced in [6] is extended to conditional application conditions. It is shown that from the specification point of view these additional features increase the expressive power. We presented a hierarchy of classes of graph languages $\mathcal{L}^S_g = \mathcal{L}^P_g \subset \mathcal{L}^N \subset \mathcal{L}^C$ for the single pushout approach which is not much different for the double pushout approach ($\mathcal{L}^S_c \subseteq \mathcal{L}^P_c$).

Furthermore we have shown that special application conditions can be represented using non-equations resp. conditional equations. In the general case additionally the existence of specific structures can be recommanded. It is left to future research to give a logical representation for all possible application conditions. Such a representation could be very useful if one wants to prove the consistency of a specification. In [3] also application conditions for the right hand side of the rule were presented. Consistency conditions based one equations were already introduced in [10]. It is an open question wether both can be brought together, i.e. it is possible to prove that the satisfaction of application conditions implies that consistency is preserved.

Another interesting task is to carry over theoretical results known for the pure single pushout approach to this extended one, for example concepts of independence and parallelism of graph transformations, embedding, concurrency etc. First results are present for graph grammars over rules with positive and negative application conditions in [6] and [7].

Acknowledgements

The author is grateful to H.Ehrig, R.Heckel and G.Taentzer for comments and fruitful discussions on the topics presented in this paper.

References

1. M. Arbib and E.G. Manes. *Arrows, Structures, and Functors: The Categorical Imperative.* Academic Press, New York, 1975.
2. H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *1st Graph Grammar Workshop, Lecture Notes in Computer Science 73*, pages 1–69, 1979.
3. H. Ehrig and A. Habel. Graph grammars with application conditions. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 87–100. 1985.
4. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. From graph grammars to High Level Replacement Systems. pages 269–291, 1991. Lecture Notes in Computer Science 532.
5. H. Ehrig and M. Löwe. Categorical principles, techniques and results for high-level replacement systems in computer science. *Applied Categorical Structures*, 1(1):21–50, 1993.
6. A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. accepted for special issue of Fundamenta Informaticae, 1994.
7. R. Heckel. Embedding of conditional graph transformations. unpublished, 1994.
8. R. Heckel, J. Müller, G. Taentzer, and A. Wagner. Attributed graph transformations with controlled application of rules. submitted to proceedings of Graphgrammar Mallorca Workshop 94, 1994.
9. H. Herrlich and G. Strecker. *Category Theory.* Allyn and Bacon, Rockleigh, New Jersey, 1973.
10. M. Korff. Single pushout transformations of equationally defined graph structures with applications to actor systems. In *Proc. Graph Grammar Workshop Dagstuhl 93*, pages 234–247, 1994. Lecture Notes in Computer Science 776.
11. M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109:181–224, 1993.
12. U. Montanari. Separable graphs, planar graphs and web grammars. *Information and Control 16*, pages 243–267, 1970.
13. A. Schürr. Progress: A vhl-language based on graph grammars. In *LNCS532.* Springer, 1991.
14. A. Wagner. On the expressive power of graph grammars with application conditions. Technical Report 27, TU Berlin, 1994.