Graph Grammars with Negative Application Conditions*

Annegret Habel[1], Reiko Heckel[2], Gabriele Taentzer[2]

[1] Department of Mathematics and Computer Science,
University of Bremen, D-28334 Bremen
[2] Computer Science Department,
Technical University of Berlin, D-10587 Berlin

Abstract

In each graph-grammar approach it is defined how and under which conditions graph productions can be applied to a given graph in order to obtain a derived graph. The conditions under which productions can be applied are called application conditions. Although the generative power of most of the known general graph-grammar approaches is sufficient to generate any recursively enumerable set of graphs, it is often convenient to have specific application conditions for each production. Such application conditions, on the one hand, include context conditions like the existence or non-existence of nodes, edges, or certain subgraphs in the given graph as well as embedding restrictions concerning the morphisms from the left-hand side of the production to the given graph.

In this paper, the concept of application conditions introduced by Ehrig and Habel is restricted to contextual conditions, especially negative ones. In addition to the general concept, we state local confluence and the Parallelism Theorem for derivations with application conditions. Finally we study context-free graph grammars with application conditions with respect to their generative power.

Keywords graph grammars, graph transformation systems, application conditions, contextual conditions.

1 Introduction

In each graph-grammar approach it is defined how and under which conditions graph productions can be applied to a given graph in order to obtain a derived graph (see Kreowski, Rozenberg [16] for a comparative study). The conditions under which productions can be applied are called application conditions. Usually, the application condition is inherent to each specific graph-grammar approach and applies in a uniform way to all productions of a graph-grammar within this approach. We propose allowing different application conditions for different productions in order to be more flexible using graph-grammars for the design of systems in all kinds of application areas.

The extension to graph grammars with application conditions is done in the framework of the single-pushout approach (see Löwe [18]) because of the generality of its notion of direct derivation that allows conflicts as well as deletion in unknown context. Hence, there is a lot of space for restrictions of applicability. Apart of this fact there is no need to restrict this paper to the single-pushout approach. Because of their pure

*This work was partly supported by the ESPRIT Basic Research Working Group No. 7183: Computing by Graph Transformation (COMPUGRAPH II).

categorical nature our application conditions could also be used in the double-pushout approach as well as in arbitrary high-level replacement systems (Ehrig, Löwe [11]).

In Ehrig, Habel [9] a very general but mathematically simple notion of application conditions is introduced. Focusing on the semantical aspects, they are defined as classes of total graph morphisms containing the admissible occurrences for the left- resp. right-hand sides of rules in the given resp. derived graph.

In this work we want to approach to application conditions in a more syntactical way and consider an application condition $A(p)$ for a graph production $L \xrightarrow{p} R$ as given by a set of required contexts and a set of forbidden contexts. The application of p with $A(p)$ to a graph G means:

1. CHOOSE an occurrence of L in G.
2. CHECK the application condition $A(p)$, i.e. check whether the required contexts are present and the forbidden contexts are absent.
3. REPLACE (the image of) L by R if the application condition $A(p)$ is satisfied.

There are several arguments for the extension of graph transformations by application conditions. Although the generative power of most of the known general graph-grammar approaches is sufficient to generate any recursively enumerable set of graphs, application conditions are a necessary part of every nontrivial specification. Often they are expressed informally by assuming some kind of control mechanism which is left unspecified, but obviously this allows neither formal analysis nor testing of the specification. Another possibility is to code application conditions into the graphs using flags and extra labels. The consequences are complicate graphs and productions and a lot of additional consistency conditions. Treating application conditions *explicitly and formally* we can take them into account in parallelism and independency analysis. Moreover we can decide whether a production is "useful" or not, meaning that its application condition contains no contradiction. To check this for "encoded" application conditions would require additional knowledge of the problem to be solved.

In contrast to textual application conditions expressed in logical formulas (see e.g. Montanari [19], Schürr [23]), our application conditions have straightforward visual representations within the left-hand sides of the productions. Hence, they do not affect the graphical character of our specifications.

The next argument is more technical. 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.

From the point of view of formal-language theory, context-free graph grammars with (somehow restricted) application conditions can provide intermediate levels of expressive power between pure context-free and context-sensitive graph grammars.

In this paper we restrict ourselves to left-sided application conditions that can express positive and negative contextual conditions, e.g. the existence or non-existence of certain nodes, edges or certain subgraphs in the given graph as well as embedding restrictions, e.g. injectivity constraints concerning the morphism from the left-hand side of the production to the given graph. Context conditions in the sense above were informally used in Pfaltz and Rosenfeld [20] and Montanari [19] and the use of morphisms between graphs is essential to the algebraic approach to graph grammars (Ehrig [8]).

The paper is organized as follows: Section 2 provides basic notions on graphs, graph productions and derivations. In section 3 we define positive and negative application conditions and compare them with [9]. We show how to express well-known application conditions like the dangling condition and the identification condition used in the double-pushout approach [8]. Moreover, we show that each positive application condition can be expressed directly by context enlargement of the production. Section 4 contains an example of a lift control system. This is to support our claim that every nontrivial graph specification needs application conditions and is used as a running example in section 5. There we propose a revised definition of independence of derivations with application conditions. Then, the local confluence and the Parallelism Theorem are shown wrt. this new definition. Section 6 is concerned with the generative power of context-free graph grammars with application conditions. In particular, it is shown that context-free graph grammars with application conditions are more powerful than ordinary context-free graph grammars without application conditions. The same holds true, if only negative (positive) conditions are allowed. In the last section, we conclude with a general discussion of the presented approach and possibilities for further development.

2 Preliminaries

This section provides the basic notions on graphs and graph morphisms, graph productions and derivations needed in the paper.

Definition 2.1 (*Graphs and graph morphisms*)

1. Let $LAB = (LAB_V, LAB_E)$ be a pair of sets, called pair of *label alphabets* for nodes and edges respectively, which will be fixed in the following.

2. A *graph* $G = (V_G, E_G, s_G, t_G, c_G, d_G)$ over LAB consists of a set V_G of nodes, a set E_G of edges, two mappings $s_G : E_G \to V_G$ and $t_G : E_G \to V_G$ assigning a source node and a target node to each edge, and two mappings $c_G : V_G \to LAB_V$ and $d_G : E_G \to LAB_E$, labeling nodes and edges, respectively. The set of all graphs over LAB is denoted by $\mathcal{G}(LAB)$.

3. G is called a *subgraph* of G', denoted by $G \subseteq G'$, if $V_G \subseteq V_{G'}$, $E_G \subseteq E_{G'}$, and the mappings s_G, t_G, c_G, and d_G are restrictions of the corresponding mappings in G'.

4. A *total graph morphism* $f : G \to G'$, $G \xrightarrow{f} G'$ or f for short, consists of two mappings $f_V : V_G \to V_{G'}$ and $f_E : E_G \to E_{G'}$ such that $f_V \circ s_G = s_{G'} \circ f_E$, $f_V \circ t_G = t_{G'} \circ f_E$, $f_V \circ c_G = c_{G'}$, and $f_E \circ d_G = d_{G'}$. The set of all total graph morphisms starting from a graph G is denoted by $\mathcal{MOR}(G)$.

   f is called *injective* (*surjective, bijective*) if both f_V and f_E are injective (surjective, bijective). A total bijective graph morphism $f : G \to G'$ is called an *isomorphism* and there is an inverse isomorphism $f' : G' \to G$. In this case, G and G' are said to be *isomorphic*, denoted by $G \cong G'$.

5. A *partial graph morphism* $f : G \to G'$ is a total graph morphism from some subgraph $G_f \subseteq G$ to G'. G is the *domain* and G' the codomain of f. G_f is called the *scope* of f. Alternatively, we can consider f as a pair of partial mappings $f_V : V_G \to V_{G'}$ and $f_E : E_G \to E_{G'}$ satisfying the same commutativity requirements

as for total morphisms but in the category of partial mappings, i.e. if one side of the equation is defined the other one has to be defined too. To avoid confusion, if we write "morphism" we usually mean a partial (i.e. not necessary total) graph morphism.

6. The composition $f' \circ f : G \to G''$ of two (partial) graph morphisms $f : G \to G'$ and $f' : G' \to G''$ is defined by $f' \circ f = (f'_V \circ f_V, f'_E \circ f_E)$, where "$\circ$" is the composition in the category of sets and partial mappings.

7. For a given pair of label alphabets LAB, graphs and total graph morphisms over LAB define the category of graphs with total morphisms, $\mathbf{Graphs^T_{LAB}}$ for short. Accordingly, graphs and partial graph morphisms define the category $\mathbf{Graphs_{LAB}}$. Note that $\mathbf{Graphs^T_{LAB}}$ is a complete subcategory of $\mathbf{Graphs_{LAB}}$.

$\triangle$

Definition 2.2 (*Graph productions and derivations*)

1. A *graph production* $L \xrightarrow{p} R$ over LAB is a (partial) graph morphism in the category $\mathbf{Graphs_{LAB}}$. L is called the *left-hand side* and R the *right-hand side* of p. It is *injective* if the partial morphism is injective.

2. The application of a graph production $L \xrightarrow{p} R$ to a graph G is given by a total graph morphism $L \xrightarrow{m} G$, also called *occurrence of* L *in* G or *match for production* p. The *direct derivation* $G \overset{p,m}{\Longrightarrow} H$ from G to the derived graph H is given by the pushout[3] of p and m in $\mathbf{Graphs_{LAB}}$ as shown below.

$$\begin{tikzcd} L \arrow[r, "p"] \arrow[d, "m"'] \arrow[dr, phantom, "\text{(PO)}"] & R \arrow[d, "m^*"] \\ G \arrow[r, "p^*"'] & H \end{tikzcd}$$

3. A sequence of direct derivations of the form $G_0 \overset{p_1,m_1}{\Longrightarrow} \ldots \overset{p_k,m_k}{\Longrightarrow} G_k$ constitutes a *derivation* from G_0 to G_k by $p_1, \ldots, p_k$. Such a derivation is denoted by $G_0 \Longrightarrow^*_P G_k$ if $p_1, \ldots, p_k \in P$ for some set P of productions.

$\triangle$

Remark 2.3 Intuitively, the application of a graph production $L \xrightarrow{p} R$ to a graph G works as follows: Replace the occurrence of L in G by R. Delete edges whose source or target nodes are deleted. If one node or edge shall be deleted as well as preserved, solve this conflict by deletion, too. An exact construction is presented in [18] in the framework of graph structures. $\triangle$

[3] For the fundamental definitions and constructions — such as pushouts — of category theory we refer to Arbib and Manes [2] and Adámek, Herrlich and Strecker [1].

3 Productions and Derivations with Application Conditions

In Ehrig, Habel [9] a very general notion of application conditions for productions in the algebraic approach is presented. In the following definition we restrict ourselves to positive and negative left-sided contextual conditions, which can be expressed by sets of total morphisms starting from the left-hand side of the productions.

Definition 3.1 (*Application conditions*)

1. An *application condition* $A(p) = (AP(p), AN(p))$ for a graph production $L \xrightarrow{p} R$ consists of two sets of total morphisms $AP(p), AN(p) \subseteq \mathcal{MOR}(L)$ starting from L, that contain positive and negative constraints, respectively. $A(p)$ is called positive (negative) if $AN(p)$ ($AP(p)$) is empty.

2. Let $L \xrightarrow{l} \hat{L}$ be a positive or negative constraint and $L \xrightarrow{m} G$ a total graph morphism. Then m *P-satisfies* l, written $m \models_P l$, if there exists a total graph morphism $\hat{L} \xrightarrow{n} G$ such that $n \circ l = m$. m *N-satisfies* l, written $m \models_N l$, if it does not P-satisfy l, i.e. $m \models_N l \iff m \not\models_P l$.

3. Let $A(p) = (AP(p), AN(p))$ be an application condition and $L \xrightarrow{m} G$ a total graph morphism. Then m *satisfies* $A(p)$, written $m \models A(p)$, if it P-satisfies all positive constraints and N-satisfies all negative constraints from $A(p)$, i.e. if

$$\forall\, l \in AP(p).\ m \models_P l \quad \wedge \quad \forall\, k \in AN(p).\ m \models_N k.$$

   $A(p)$ is said to be *consistent* if there is a total morphism $m \in \mathcal{MOR}(L)$ s.t. m satisfies $A(p)$.

4. A *graph production with application condition* is a pair $\hat{p} = (L \xrightarrow{p} R, A(p))$ consisting of a graph production p and an application condition $A(p)$ for p. It is *applicable* to a graph G via $L \xrightarrow{m} G$ if m satisfies $A(p)$.

5. Given $\hat{p}$ and m s.t. $\hat{p}$ is applicable to G via m the direct derivation $G \overset{\hat{p},m}{\Longrightarrow} H$ with application condition is the direct derivation $G \overset{p,m}{\Longrightarrow} H$.

△

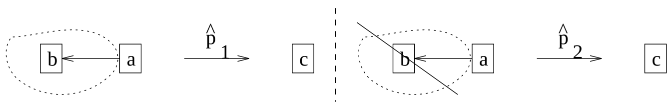
Remark 3.2 1. A single application condition is essentially the same as a constraint. It is positive or negative depending on whether the single constraint is in $AP(p)$ or in $AN(p)$. Therefore we use the term "application condition" for constraints as well as for the whole conditions as far as not both of them are involved.

2. Defining positive and negative application conditions in this way, they fit exactly into the framework of application conditions given in [9]. A left-sided application condition $\bar{A}(p)$ for a graph production $L \xrightarrow{p} R$ in the sense of [9] consists of a decidable class of graph morphisms starting from L, i.e. $\bar{A}(p) \subseteq \mathcal{MOR}(L)$, that describes explicitly the class of allowed occurrences. An application condition $A(p)$ as defined above can be seen as a specification for such a class of occurrences. For a given $A(p)$ we can define $\bar{A}(p) = \{m \mid m \text{ satisfies } A(p)\}$. Now m satisfies $A(p)$ if and only if $m \in \bar{A}(p)$.

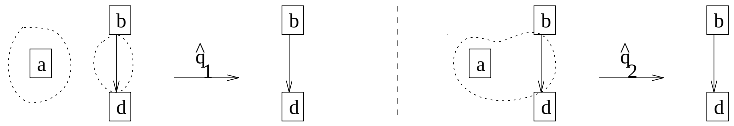
△

Example 3.3 Within the following examples we use a graphical layout for application conditions. Injective constraints $L \xrightarrow{l} \hat{L}$ are represented in the left-hand side of a production $L \xrightarrow{p} R$ where they are distinguished by dotted borders. All nodes and edges outside these borders form the left-hand side L while $\hat{L}$ is given by the left-hand side plus one of the dotted bordered parts and l by the corresponding embedding.

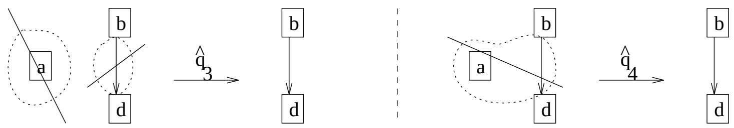
1. In the left part of the figure below the production $\hat{p}_1 = (p, (\{l\}, \emptyset))$ is shown, where the positive constraint l requires the existence of an edge from the node with label a to a node labeled by b. The left-hand side L of p consists of the a-labeled node while $\hat{L}$ is given by the two nodes and the edge in between. The right part shows the production $\hat{p}_2 = (p, A_2(p))$ with application condition $A_2(p) = (\emptyset, \{l\})$. The underlying production p and the constraint l are the same as in $\hat{p}_1$ but here l is a negative constraint which forbids the existence of an unlabeled edge from the a-node to a b-node.



2. Now we want to combine different positive or negative constraints in one application condition. They are distinguished graphically by framing them separately. Production $\hat{q}_1 = (q, (\{l_1, l_2\}, \emptyset))$ is shown in the left part of the following figure. Constraint l_1 requires an edge from the b- to the d-node and constraint l_2 the existence of a node labeled by a. Another way of combination of constraints is to glue them together wrt. the left-hand side, i.e. $\hat{q}_2 = (q, (\{l_1 \oplus_L l_2\}, \emptyset))$. This is shown in the right part and is equivalent to the former case.

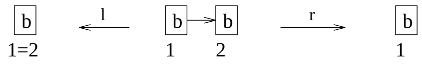


3. If we consider l_1 and l_2 as negative constraints we have to pay attention. Requiring two separate constraints l_1 and l_2 would mean that an a-node *and* an edge between the b- and the d-node are forbidden. This leads to the production $\hat{q}_3 = (q, (\emptyset, \{l_1, l_2\}))$ below. If an a-node *or* an edge between the b- and the d-node is forbidden, i.e. *both* objects must not exist this is expressed by the gluing of l_1 and l_2 wrt. L as in $\hat{q}_4 = (q, (\emptyset, \{l_1 \oplus_L l_2\}))$. Given a graph consisting of a b- and a d-node with an edge in between $\hat{q}_4$ is applicable since there is no a-node, while production $\hat{q}_3$ is not applicable because of the existing edge.



4. Non-injective constraints can be used to forbid (or require) identifications of certain nodes or edges. The application condition $A_1(r) = (\emptyset, \{l\})$ of $\hat{r}$ below is satisfied

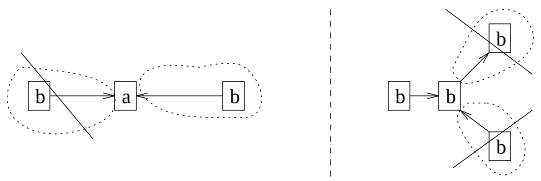
by a match m if $m(1) \neq m(2)$, i.e. it expresses exactly the identification condition[4] for r (which in this case requires injectivity). Morphisms r and l are denoted by the same numbers at corresponding nodes.



5. According to definition 3.1 an application condition is consistent if there is a match satisfying this condition. Inconsistency occurs if there are contradictions between positive and negative constraints or between negative constraints and the left-hand side. A trivial contradiction is to take an isomorphism as a negative constraint.

   The first contradictory situation is shown in the left part of the figure below. An edge from a b-node to the a-node is required by a positive constraint as well as forbidden by the negative one. The right part shows the left-hand side of the production r of the previous example together with negative constraints l_1, l_2 requiring that there is no outgoing edge and no ingoing edge for the node to be deleted by r. Note that this is a part of the dangling condition[5] for r. The second constraint, however, cannot be satisfied by a match for r because the ingoing edge is already present in the left-hand side. Thus, $A_2(r) = (\emptyset, \{l_1, l_2\})$ is not consistent.

$\triangle$



The inconsistency of $A_2(r)$ in example 3.3.5 reflects a gap of expressive power of our application conditions. Up to now, they cannot distinguish between the existence of one or two edges in the context of a node if they have the same label and direction, i.e. we cannot express cardinality restrictions. We solve this problem by introducing *injective satisfaction* of constraints and application conditions where the required and forbidden morphisms are assumed to be injective.

Definition 3.4 (*Injective satisfaction*)

1. A total graph morphism $L \xrightarrow{m} G$ *injectively P-satisfies (N-satisfies)* a constraint $L \xrightarrow{l} \hat{L}$, written $m \models_{P,inj} l$ ($m \models_{N,inj} l$), if there is (not) a total, *injective* morphism $\hat{L} \xrightarrow{n} G$ such that $n \circ l = m$.

---

[4]This example is motivated by the double-pushout approach to graph transformation which can be regarded as a special case of the single-pushout approach (see [18] for comparison) with additional application conditions for each production. These conditions are the *identification condition* not allowing conflicts between deletion and preservation of objects and the *dangling condition* prohibiting dangling edges after deletion of nodes. A match $L \xrightarrow{m} G$ satisfies the identification condition if each object x with $m(x) = m(y)$ for $x \neq y$ is in L_p.

[5]A match $L \xrightarrow{m} G$ satisfies the *dangling condition*, if each node v in L which is source or target of an edge in $G - m(L)$ is in L_p.

2. A match m *injectively satisfies* an application condition $A(p) = (AP(p), AN(p))$, written $m \models_{inj} A(p)$, if m injectively P-satisfies all constraints in $AP(p)$ and injectively N-satisfies all constraints in $AN(p)$.

$\triangle$

Now $A_2(r)$ with injective satisfaction is no longer inconsistent and the second constraint forbids the existence of a *second* ingoing edge. To express the dangling condition for r we additionally have to forbid all possible gluings of $\hat{L}_1$ resp. $\hat{L}_2$, i.e. the existence of a loop at the deleted node and the existence of a second edge between the two required nodes. The following proposition shows that application conditions with injective satisfaction are sufficiently powerful to express the dangling condition and the identification condition for arbitrary productions.

Proposition 3.5 (*Special application conditions*)

The injectivity condition: Let $INJ(p) = (\emptyset, INJ_N(p))$ where $INJ_N(p)$ consists of all total, surjective morphisms $L \xrightarrow{l} \hat{L}$ except isomorphisms $L \xrightarrow{l} L$. Then m is injective if and only if $m \models INJ(p)$.

The identification condition: Let $ID(p) = (\emptyset, ID_N(p))$ with $ID_N(p) \subseteq INJ_N(p)$ such that for all $l \in ID_N(p)$ $l(x) = l(y)$ for some $x, y \in L$ with $x \neq y$ and $x \notin L_p$. Then m satisfies the identification condition if and only if $m \models ID(p)$.

The dangling condition: Let $DANG(p) = (\emptyset, DANG_N(p))$ and $DANG_N(p)$ be the set of all total morphisms $L \xrightarrow{l} \hat{L}$ such that l is surjective up to an edge e (and possibly a node) with $s(e) \in l(L - L_p)$ or $t(e) \in l(L - L_p)$. Then m satisfies the dangling condition if and only if $m \models_{inj} DANG(p)$.

$\triangle$

Proof: Direct consequence of definition 3.1 and 3.4. $\square$

Remark 3.6 If we consider finite label alphabets for nodes and edges the dangling and the identification condition can be expressed by a finite number of constraints provided that the left-hand side L of the production is finite. $\triangle$

Application conditions can be extended by context via total morphisms.

Lemma 3.7 (*Extension of constraints*) Let $L \xrightarrow{l} \hat{L}$ be a constraint and $L \xrightarrow{e} K$ a total morphism. Furthermore, let $K \xrightarrow{k} \hat{K}$ be the constraint obtained by the pushout of l and e as in the left diagram of figure 1. Then for all total morphisms $K \xrightarrow{m} G$ we have

$$m \models_P k \iff m \circ e \models_P l \quad \text{and} \quad m \models_N k \iff m \circ e \models_N l$$

$\triangle$

Proof: We show that there exists a total morphism $\hat{K} \xrightarrow{n} G$ with $n \circ k = m$ if and only if there exists $\hat{L} \xrightarrow{n'} G$ with $n' \circ l = m \circ e$.

Assume n as above. Then we have $n' = n \circ \hat{e}$ and the required commutativity by commutativity of pushout diagrams.

Now let n' be given as above. Then $\hat{K} \xrightarrow{n} G$ with $n \circ k = m$ exists by the universal property of pushouts. $\square$

Positive application conditions for a production $L \xrightarrow{p} R$ can be expressed directly by context enlargement of the production. Doing this, we first glue all positive constraints wrt. L using a colimit construction and apply p to the resulting graph $\bar{L}$. Finally, we extend the negative constraints to the new left-hand side $\bar{L}$ of the derived production $\bar{L} \xrightarrow{\bar{p}} \bar{R}$.

$$\begin{tikzcd}
\hat{L} \arrow[d, "\hat{e}"'] \arrow[dr, phantom, "\text{(PO)}"] & L \arrow[l, "l"'] \arrow[d, "e"] \\
\hat{K} \arrow[dr, "n"'] & K \arrow[l, "k"'] \arrow[d, "m"] \\
 & G
\end{tikzcd}
\qquad
\begin{tikzcd}
\hat{N}_j \arrow[dd, "\hat{l}_j"'] \arrow[ddr, phantom, "(1)"] & L \arrow[l, "n_j"'] \arrow[r, "p"] \arrow[d, "l_i"] \arrow[dd, bend right, "\bar{l}"'] \arrow[ddd, bend left, "m"] \arrow[ddr, phantom, "(2)"] & R \arrow[dd, "\bar{l}^*"] \\
 & \hat{L}_i \arrow[d, "\bar{l}_i"] & \\
\bar{N}_j & \bar{L} \arrow[l, "\bar{n}_j"'] \arrow[r, "\bar{p}"] \arrow[d, "\bar{m}"'] \arrow[dr, phantom, "(3)"] & \bar{R} \arrow[d, "\bar{m}^*"] \\
 & G \arrow[r, "\bar{p}^*"] & H
\end{tikzcd}$$

Figure 1: Extension and context enlargement.

Definition 3.8 (*Context enlargement*) Let $\hat{p} = (p, A(p))$ be a graph production with application condition $A(p) = (\{L \xrightarrow{l_i} \hat{L}_i | \ i \in I\}, \{L \xrightarrow{n_j} \hat{N}_j | \ j \in J\})$. The *context enlargement* $\hat{\bar{p}} = (\bar{L} \xrightarrow{\bar{p}} \bar{R}, A(\bar{p}))$ *of* $\hat{p}$ with application condition $A(\bar{p}) = (\emptyset, \{\bar{L} \xrightarrow{\bar{n}_j} \bar{N}_j | \ j \in J\})$ is defined by the following steps:

1. Construction of the colimit[6] $(\bar{L}, \hat{L}_i \xrightarrow{\bar{l}_i} \bar{L})$ over morphisms $L \xrightarrow{l_i} \hat{L}_i$ for $i \in I$.

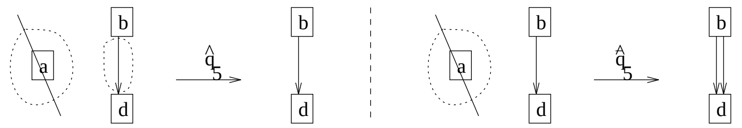
2. Construction of the pushout $\bar{L} \xrightarrow{\bar{p}} \bar{R} \xleftarrow{\bar{l}^*} R$ over morphisms $\bar{L} \xleftarrow{\bar{l}} L \xrightarrow{p} R$ where $\bar{l} = \bar{l}_i \circ l_i$ for some $i \in I$.

3. Construction of the pushouts $\hat{N}_j \xrightarrow{\hat{l}_j} \bar{N}_j \xleftarrow{\bar{n}_j} \bar{L}$ over morphisms $\hat{N}_j \xleftarrow{n_j} L \xrightarrow{\bar{l}} \bar{L}$ for all $j \in J$.

$\triangle$

Example 3.9 Consider production $\hat{q}_5 = (q, (\{l_1\}, \{l_2\}))$ on the left-hand side below, where the existence of an edge between a b-node and an a -node is required and an a-node is forbidden. Doing the context enlargement on $\hat{q}_5$ we obtain production $\hat{\bar{q}}_5 = (\bar{q}, (\emptyset, \{\bar{l}_2\}))$ depicted in the right part below. $\triangle$

[6]For the definition of arbitrary colimits we refer to [1].



Proposition 3.10 (*Context enlargement*) Let $\hat{p} = (p, A(p))$ be a graph production with application condition and $\hat{\bar{p}} = (\bar{L} \xrightarrow{\bar{p}} \bar{R}, A(\bar{p}))$ the context enlargement of $\hat{p}$. Then for each direct derivation $G \xRightarrow{\hat{p},m} H$ we have a direct derivation $G \xRightarrow{\hat{\bar{p}},\bar{m}} H$ and vice versa, i.e. there is a bijective correspondence between derivations by $\hat{p}$ and derivations by its context enlargement $\hat{\bar{p}}$. $\triangle$

Proof: Consider the right diagram of figure 1. The proof is given in two steps. First we show that there is a bijective correspondence between matches $L \xrightarrow{m} G$ that satisfy $A(p)$ and matches $\bar{L} \xrightarrow{\bar{m}} G$ that satisfy $A(\bar{p})$. Then, given corresponding m and $\bar{m}$, we show that the direct derivations $G \xRightarrow{p,m} H$ and $G \xRightarrow{\bar{p},\bar{m}} H$ are the same, meaning that their derivation morphisms p^* and $\bar{p}^*$ are isomorphic.

Given m as above. Since $m \models_P AP(p)$ there are total morphisms $\hat{L}_i \xrightarrow{m_i} G$ with $m_i \circ l_i = m$ for all $i \in I$, i.e. $m_i \circ l_i = m_j \circ l_j$ for all $i, j \in I$. By universal property of the colimit object $\bar{L}$ this implies the existence of a unique morphism $\bar{L} \xrightarrow{\bar{m}} G$ with $\bar{m} \circ \bar{l}_i = m_i$.

Given $\bar{m}$, we define $m = \bar{m} \circ \bar{l}$ which ensures, that m satisfies the positive constraints of $A(p)$. It is easy to see, that these two constructions are in fact inverse to each other, i.e. matches m with $m \models AP(p)$ and matches $\bar{m}$ are in bijective correspondence. Since negative constraints $n_j \in AN(p)$ are translated to $\bar{n}_j \in A(\bar{p})$ by the pushout of $\bar{l}$ and n_j we conclude with lemma 3.7 that $m \models_N n_j \iff \bar{m} \models \bar{n}_j$ for all $j \in J$ and therefore $m \models A(p) \iff \bar{m} \models A(\bar{p})$.

Now for the second step assume m and $\bar{m}$ as above, i.e. $m = \bar{m} \circ \bar{l}$. (2) is a pushout by definition 3.8 while (3) is the derivation diagram of $G \xRightarrow{\bar{p},\bar{m}} H$. By property of pushout composition (see for example [1]) (2+3) is a pushout and therefore isomorphic to the derivation diagram of $G \xRightarrow{p,m} H$. $\square$

Remark 3.11 By proposition 3.10, positive application conditions can be expressed by enlarging the production in consideration. Nevertheless, it seems to be convenient to allow positive application conditions for productions to obtain simple productions with application conditions instead of complicate productions without application conditions. $\triangle$

The following lemma characterizes productions with consistent application conditions by checking if the application condition is contradictory. Contradictions can occur between positive and negative constraints as well as between negative constraints and the left-hand side L (see example 3.3.5). By context enlargement we can reduce the problem to the second case.

Lemma 3.12 (*Consistency of application conditions*) Let $A(p) = (AP(p), AN(p))$ be an application condition for p. Then $A(p)$ is consistent if and only if $\bar{l}$ satisfies $(\emptyset, AN(p))$ where $\bar{l}$ is constructed as in definition 3.8. $\triangle$

Proof: "$\Longrightarrow$": Assume $L \xrightarrow{n_j} \hat{N}_j \in AN(p)$ s.t. $\bar{l}$ not N-satisfies n_j, i.e. there is $\hat{N}_j \xrightarrow{g} \bar{L}$ with $g \circ n_j = \bar{l}$. Now let $L \xrightarrow{m} G$ be a match for p that satisfies $AP(p)$, i.e. for all $l_i \in AP(p)$ there exists a $\hat{L}_i \xrightarrow{m_i} G$ with $m_i \circ l_i = m$. Then by universal property of $\bar{L}$ there is a $\bar{L} \xrightarrow{\bar{m}} G$ with $\bar{m} \circ \bar{l}_i = m_i$ and therefore there exists $\hat{N}_j \xrightarrow{\bar{m} \circ g} G$ with $\bar{m} \circ g \circ n_k = m$.

"$\Longleftarrow$": Assume that $\bar{l}$ satisfies all negative constraints in $AN(p)$ then $\bar{l}$ satisfies $A(p)$ since it satisfies all positive constraints by construction. $\square$

4 Lift Control System

In this section we show how to use graph grammars with negative application conditions for the specification of a lift control system. We start with an informal description of the equipment and intended behavior of our lift.

Besides a number of floors and the lift itself we assume call buttons "up" and "down" on every floor as well as target buttons for every floor inside the lift. In reply to user requests, which have to be entered using this buttons, the lift has to move its users from floor x to floor y. Doing this it is allowed to pass floor x once more[7].

In the following specification actions of the system like lift movement and lift control as well as external events like user requests are modelled by application of productions while their states are represented by graphs in the following way. Floors and the lift are represented by appropriately labeled nodes and its current position by an unlabeled edge from the *lift* node to a *floor* node. The ordering of floors is given by edges from every *floor* node to all upper *floor* nodes. A user request is modelled by a *req* node with an edge to the corresponding floor. Request nodes carry loops specifying the type of request which is one of *up*, *down* and *halt*. Up and down requests come from the call buttons of the floor they are pointing to, while halt requests are produced inside the lift and point to the user's target floor. A node labeled *up* or *down* shows the general direction and *move* edges are used to specify the atomic movements of the lift. The initial state of the system is given by the start graph Z_0 in figure 2. It has four floors with the lift in the ground floor, no user requests and the general direction is "up".

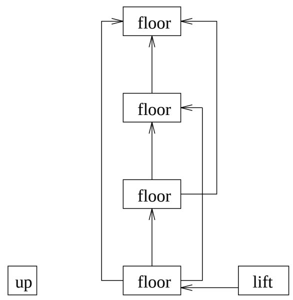


Figure 2: Start graph Z_0.

We distinguish between three kinds of productions modelling the creation of user requests, the lift control and the actual lift movement. The productions given below cover the cases where the general direction is "up". Similiar productions for "down"

[7]To give an example it could happen, that the lift stops at floor x in reply for a down request while moving upwards. Then it changes its direction, passes x once again and finally stops at floor y below x.

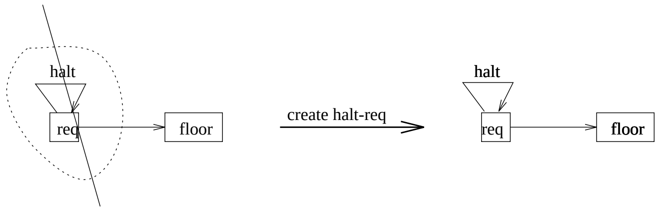


Figure 3: One of three request productions.

are obtained by replacing all occurrences of the string "up" by "down" and vice versa and by reversing all unlabeled edges between two *floor* nodes in the productions.

Request productions simulate the occurrence of user requests. There is a specific production for every type of request. Figure 3 shows one of them, similiar productions are needed for *up* and *down*. Application conditions avoid (to some extend) the creation of redundant requests.

Lift control productions (figure 4) translate user requests into move instructions that have to be executed by the lift. *initiate move* generates such an instruction in shape of a *move* edge pointing to the uppermost request, if there isn't already one. *split move* breaks a move instruction into smaller ones to serve requests that are closer to the current position and that would be missed otherwise. *delete request* allows to delete a request (and is forced to do so later on) if it meets the current position of the lift. If there is no move instruction and no request left above the current position but a request below the current position, the general direction changes to "down" using *change direction*.

The *move* production (figure 5) simulates the actual movement of the lift following the instructions generated by the control productions. The application condition ensures that *move* cannot be applied if there is anything to do for *delete request* or *split move*. This ensures that we don't leave behind requests that are already served and that we don't pass a request without serving it.

To illustrate the use of the productions we give a short derivation sequence (see figure 6 [8]). For simplicity we left out the transitive closure of the edges that specify the "above"-order of floors. The derivation goes as follows:

$Z_0 \rightarrow$ *create up-req (to 3rd floor); initiate move* $\rightarrow Z_1 \rightarrow$ *create halt-req (to 2nd floor); create halt-req (to 3rd floor)* $\rightarrow Z_2 \rightarrow$ *split move* $\rightarrow Z_3 \rightarrow$ *move; move* $\rightarrow Z_4 \rightarrow$ *create halt-req (to 4th floor); initiate move* $\rightarrow Z_5 \rightarrow$ *delete request; move* $\rightarrow Z_6$.

If there is a halt request for a floor below the current position the general direction may change to "down" using *change direction*.

Finally we want to discuss the various ways application conditions are used in our specification. In the request productions they simply avoid redundant application. So, the effort for deleting requests can be decreased.

Sometimes application conditions are used like priorities to restrict the order of application (with respect to certain objects in the graph). Considering for example the *move* and the *split move* production, we see that *split move* has priority before *move*. Similarily *move*, *initiate move* and *split move* have priority before *change direction* because of the forbidden request node and *move* edge in the left-hand side of the last production.

Two more ways negative conditions can be used are shown in *initiate move*. There

[8]This is dedicated to Reiko's little son Maximilian who was sitting on his lap, carefully watching what was going on, while Reiko was drawing this picture.

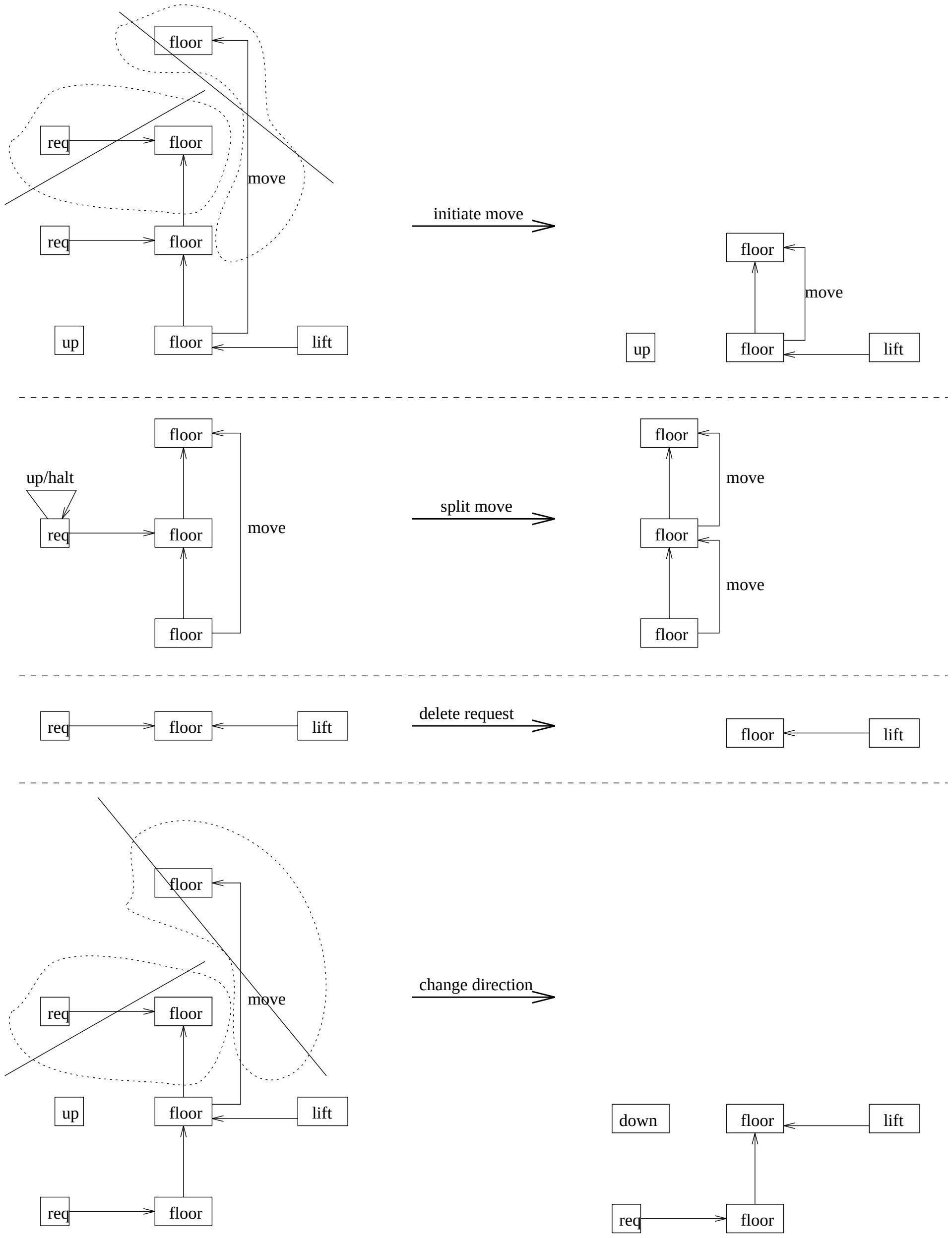


Figure 4: Productions modelling lift control.

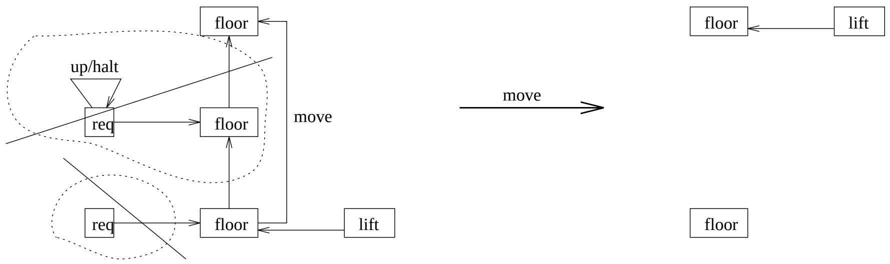


Figure 5: The *move* production.

the forbidden *req* node ensures that the *move* edge inserted by this production always points to the uppermost request, which restricts the nondeterministic choice of an arbitrary request. The other application condition is needed to ensure that *initiate move* can create only one move edge at a time between two given floor nodes. This shows that

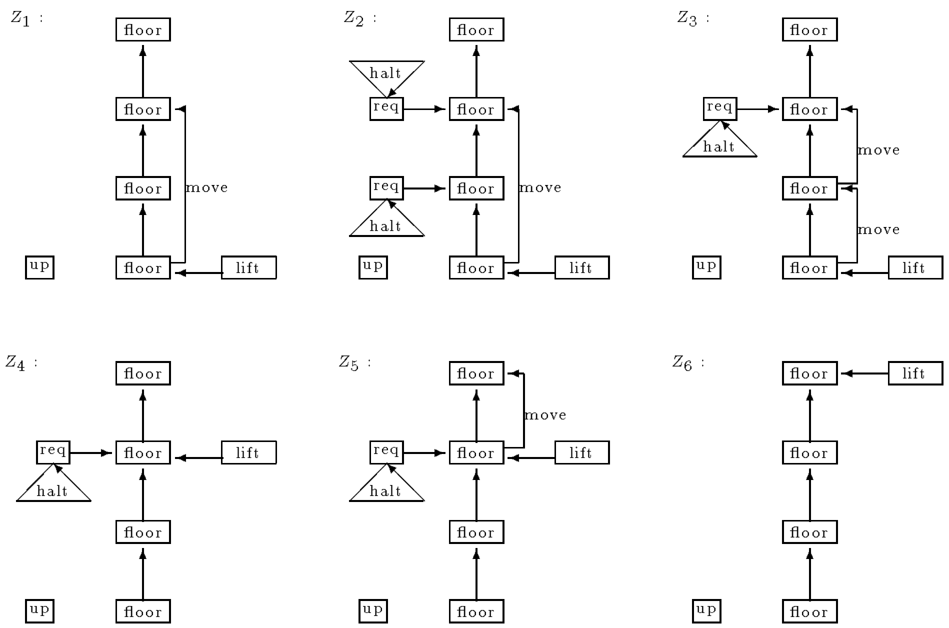


Figure 6: An examplary derivation sequence.

application conditions are also good for avoiding nonterminating derivations by excluding iterated application of the same production. It may be interesting to investigate if this property can be exploited formally to support proofs of termination.

5 Independence and Parallelism of Derivations

In this section we state a theorem on local confluence and the Parallelism Theorem for graph derivations with application conditions. As shown in section 3 productions with positive and negative application conditions can be translated into productions with negative ones only by context enlargement. Hence it is sufficient to show the results for negative application conditions.

General Assumption 5.1 For the rest of this section we assume to have productions $\hat{p}_1 = (L_1 \xrightarrow{p_1} R_1, A(p_1))$ and $\hat{p}_2 = (L_2 \xrightarrow{p_2} R_2, A(p_2))$ with negative application conditions only. $\triangle$

Usually, two derivations are considered to be parallel independent if they can be applied in any order with the same result. For single-pushout derivations this is true if the first derivation does not delete something that is needed for the application of the second. Now, since applicability can also be constrained by negative application conditions, we have to ensure that the first derivation does not *generate* something that contradicts an application condition.

Definition 5.2 (*Parallel independence of derivations*) Let $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ be two direct derivations.

1. $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ is *weakly parallel independent* of $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ if $x_2 := p_1^* \circ m_2 : L_2 \to H_1$ is total and x_2 satisfies $A(p_2)$.

2. $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ are *parallel independent* if $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ is weakly parallel independent of $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_1$ and vice versa (see left diagram in figure 7).

△

Example 5.3 Consider graph Z_1 from the derivation sequence in figure 6 where both, *create halt-req* (to the second floor) and *move*, are applicable alternatively. Now the first derivation is weakly parallel independent of the second but not vice versa because of the forbidden up/halt request in the left-hand side of *move*. △

Definition 5.4 (*Sequential independence of derivations*) Let $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ be two direct derivations.

1. $H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ is *weakly sequential independent* of $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ if there exists a total morphism $L_2 \xrightarrow{m_2} G$ such that $p_1^* \circ m_2 = x_2$ and m_2 satisfies $A(p_2)$.

2. $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ are *sequentially independent* if $H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ is weakly sequential independent of $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ is weakly parallel independent of $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ (see middle diagram in figure 7).

△

Example 5.5 Apply *delete request* to Z_5 of figure 6 and then *move* to the resulting graph. Now the second derivation is *not* weakly sequential independent of the first, i.e. the obsolete request has to be deleted *before* we leave this floor. Note that the second derivation is independent of the first if we consider *move* without its application conditions. △

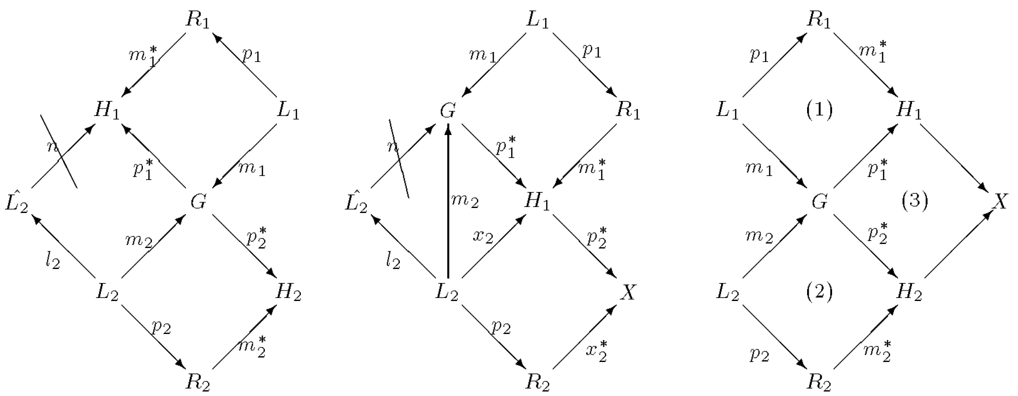


Figure 7: Independence of derivations and local confluence.

Remark 5.6 Two derivations with positive *and* negative application conditions are considered to be parallel (sequentially) independent if and only if their corresponding derivations with negative application conditions are. △

The following local confluence theorem formalizes our claim that the above definitions are "correct" wrt. the usual meaning of independence.

Theorem 5.7 (*Local confluence*)

1. Let $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ be parallel independent direct derivations. Then there are a unique graph X with direct derivations $H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ and $H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$ such that both derivation sequences become sequentially independent.

2. Let $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ be sequentially independent derivations. Then there is a graph H_2 with sequentially independent derivations $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$.

3. Let p_1 and p_2 be injective. Then parallel independent derivations $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ and sequentially independent derivations $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ resp. $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$ are in bijective correspondence.

△

Proof: 1. By parallel independence we have $x_1 = p_2^* \circ m_1$ satisfies $A(p_1)$ and $x_2 = p_1^* \circ m_2$ satisfies $A(p_2)$, i.e. there are derivations $H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ and $H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$ where the first one is given by pushout (2+3) (see right diagram in figure 7). By property of pushout decomposition (3) is a pushout too and by pushout composition (1+2) is a pushout (see [1]), hence it is isomorphic to the direct derivation $H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$. $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ is sequentially independent by existence of $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$ and vice versa while they are unique by uniqueness of the composites $p_2^* \circ m_1$ and $p_1^* \circ m_2$.

2. Since $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ is sequentially independent there is $L_2 \xrightarrow{m_2} G$ satisfying $A(p_2)$ s.t. $p_1^* \circ m_2 = x_2$ and $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$, $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ are parallel independent. Then existence and independence of $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$ is given by part 1.

3. For injective p_i also p_i^* are injective. Then in part 2 m_2 is uniquely given by $p_1^{*-1} \circ x_2$. Applying part 1 we get $p_1^* \circ (p_1^{*-1} \circ x_2)$ which is x_2 again since it is total by sequential independency. Similarly $p_1^{*-1} \circ (p_1^* \circ m_2)$ is equal to m_2. □

Given two productions $\hat{p}_1, \hat{p}_2$ with application conditions, their parallel production is constructed in two steps. First the parallel production of p_1 and p_2 is built and then their application conditions are extended to the larger left-hand side of the parallel production.

Definition 5.8 (*Parallel productions and derivations*) The *parallel production* $\hat{p}_1 + \hat{p}_2$ of $\hat{p}_1$ and $\hat{p}_2$ is the production $(p_1 + p_2, A(p_1 + p_2))$ where $L_1 + L_2 \overset{p_1+p_2}{\longrightarrow} R_1 + R_2$ is the parallel production of p_1 and p_2 as in [18] (i.e. the disjoint union of $L_1 \xrightarrow{p_1} R_1$ and $L_2 \xrightarrow{p_2} R_2$) and $A(p_1 + p_2)$ is the set defined by $\{ \; l_1^* \mid l_1 \in A(p_1) \; \} \cup \{ \; l_2^* \mid l_2 \in A(p_2) \; \}$ where l_i^* is obtained by the pushout of l_i and the embedding $L_i \xrightarrow{e_i} L_1 + L_2$ for $i = 1, 2$ (see left diagram in figure 8). A direct derivation $G \overset{\hat{p}_1+\hat{p}_2,m}{\Longrightarrow} X$ by $\hat{p}_1 + \hat{p}_2$ is called a *parallel derivation with negative application conditions.* △

$$\begin{tikzcd}
\hat{L}_1 \arrow[d, "e_1^*"'] & L_1 \arrow[l, "l_1"'] \arrow[r, "p_1"] \arrow[dd, "e_1"] \arrow[ddr, phantom, "(=)"] & R_1 \arrow[dd, "j_1"] \\
\hat{L}_1 + L_2 \arrow[ur, phantom, "(PO)"] & & \\
 & L_1 + L_2 \arrow[ul, "l_1^*"'] \arrow[dl, "l_2^*"] \arrow[r, "p_1 + p_2"] & R_1 + R_2 \\
\hat{L}_2 + L_1 \arrow[dr, phantom, "(PO)"] & & \\
\hat{L}_2 \arrow[u, "e_2^*"] & L_2 \arrow[l, "l_2"] \arrow[r, "p_2"'] \arrow[uu, "e_2"'] \arrow[uur, phantom, "(=)"] & R_2 \arrow[uu, "j_2"']
\end{tikzcd}$$

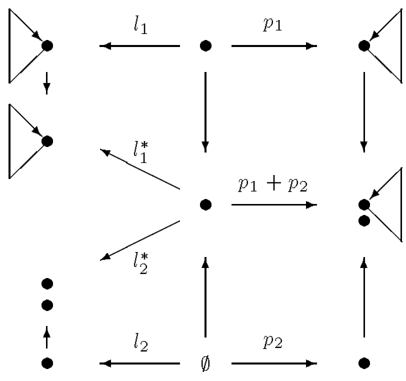


Figure 8: Construction of the parallel production with application conditions.

Example 5.9 The construction above does not guarantee, that the application condition of the parallel production is consistent even if both $A(p_1)$ and $A(p_2)$ are. In the right part of figure 8 $\hat{p}_1 = (p_1, (\emptyset, \{l_1\}))$ adds a loop to a node if there isn't already one. $\hat{p}_2 = (p_2, (\emptyset, \{l_2\}))$ inserts a node in an empty graph. The application condition of the parallel production $\hat{p}_1 + \hat{p}_2 = (p_1 + p_2, (\emptyset, \{l_1^*, l_2^*\}))$ is not consistent (compare lemma 3.12) because the colimit object over $(L_1 + L_2, \emptyset)$ is $L_1 + L_2$ and we can reverse l_2^* by identifying the two nodes. But this is exactly what we would expect, since there is no graph to which we can apply both $\hat{p}_1$ and $\hat{p}_2$, i.e. their application domains are disjoint. That this is no mere coincident is shown by the following proposition. $\triangle$

Proposition 5.10 (*Applicability of parallel production*) Let $\hat{p}_1, \hat{p}_2, \hat{p}_1 + \hat{p}_2$ be given as above together with matches $L_1 \xrightarrow{m_1} G$ and $L_2 \xrightarrow{m_2} G$ for p_1 and p_2 into G and let $L_1 + L_2 \xrightarrow{m_1 + m_2} G$ be the parallel match for $p_1 + p_2$ (i.e. the disjoint union of m_1 and m_2). Then we have

$$m_1 \models A(p_1) \ \wedge \ m_2 \models A(p_2) \quad \Longleftrightarrow \quad m_1 + m_2 \models A(p_1 + p_2).$$

$\triangle$

Proof: $m_i = m \circ e_i$ for $i = 1, 2$ by universal property of $L_1 + L_2$. Since l_i^* is obtained by the pushout of l_i and e_i we have $m \models_N l_i^* \iff m \circ e_i \models_N l_i$ for all $l_i \in A(p_i)$ by lemma 3.7. $\square$

Since we can check consistency of application conditions by lemma 3.12 this provides us with a means to decide whether a parallel production makes sense or not. Furthermore, we are now in the position to show the "synthesis" part of the Parallelism Theorem. For the "analysis" part there is still something missing. Consider for example the parallel production *create halt-request* + *create halt-request* inserting two requests at once and let them be inserted in the same floor. Then this parallel derivation cannot be sequentialized because *create halt-request* is not applicable to a floor if it already has a halt request. This is similar to the situation for unrestricted parallel single-pushout derivations. In [18] the problem is solved by restricting occurrences $L_1 + L_2 \xrightarrow{m} G$ to d-injective ones. Unfortunately, there is no such solution here. The only thing we can do is to require, that the alternative derivations that correspond to the parallel derivation under consideration are parallel independent, which ensures the sequentialization in a somewhat axiomatical way. The more interesting question of criteria for parallel independence of productions with negative application conditions is non-trivial and will be studied in a forthcoming paper.

Definition 5.11 (*Independence of parallel derivations*) A parallel derivation with negative application conditions $G \overset{\hat{p}_1+\hat{p}_2,m}{\Longrightarrow} X$ is said to be *(parallel) independent* if the derivations $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ are parallel independent, where $m_i = m \circ e_i$ for $i = 1, 2$. $\triangle$

The connection between sequentially independent derivation sequences and independent parallel derivations is established in the Parallelism Theorem.

Theorem 5.12 (*Parallelism Theorem*) Let $\hat{p}_1, \hat{p}_2$ be productions with application conditions and p_1, p_2 be injective.

Synthesis. Given a sequentially independent derivation sequence $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ there is a unique synthesis leading to an independent parallel derivation $G \overset{\hat{p}_1+\hat{p}_2,m_1+m_2}{\Longrightarrow} X$.

Analysis. Given an independent parallel derivation $G \overset{\hat{p}_1+\hat{p}_2,m}{\Longrightarrow} X$ there is a unique analysis into two sequentially independent sequences $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X$.

Bijective Correspondence. The operations "synthesis" and "analysis" are inverse to each other in the following sense: Given $\hat{p}_1, \hat{p}_2$, and $\hat{p}_1 + \hat{p}_2$ there is a bijective correspondence between sequentially independent derivation sequences $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X$ and parallel independent derivations $G \overset{\hat{p}_1+\hat{p}_2,m}{\Longrightarrow} X$.

$\triangle$

Proof: The proof makes use of the following statement on parallel derivations without application conditions which is proven in [18]: If $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent, $G \overset{p_1+p_2,m_1+m_2}{\Longrightarrow} X$, $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,x_2}{\Longrightarrow} X$ and $G \overset{p_1,m_2}{\Longrightarrow} H_2 \overset{p_1,x_1}{\Longrightarrow} X$ lead to the same result.

Synthesis: According to theorem 5.7.2 we have $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ s.t. $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ are parallel independent. Then, there is $G \overset{p_1+p_2,m_1+m_2}{\Longrightarrow} X$ by the statement on derivations without application conditions above and $m_1 + m_2$ satisfies $A(p_1 + p_2)$ due to proposition 5.10.

Analysis: According to proposition 5.10 $m_1 \models A(p_1)$ and $m_2 \models A(p_2)$ for $m_i = m \circ e_i$, $i = 1, 2$, i.e. there are direct derivations $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$. Since $G \overset{\hat{p}_1+\hat{p}_2,m}{\Longrightarrow} X$ is independent, they are parallel independent. Existence and independence of sequences $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,x_2}{\Longrightarrow} X'$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2 \overset{\hat{p}_1,x_1}{\Longrightarrow} X'$ follows from theorem 5.7.1 and due to the above statement on parallel derivations without application conditions X and X' are isomorphic.

Bijective Correspondence: By the universal property of the coproduct there is a bijection between $\mathcal{MOR}(L_1 + L_2)$ and $\mathcal{MOR}(L_1) \times \mathcal{MOR}(L_2)$. The equivalence in proposition 5.10 ensures that $G \overset{\hat{p}_1+\hat{p}_2,m}{\Longrightarrow} X$ exists if and only if both $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2$ exist. The bijection between these and independent sequences follows from theorem 5.7.3. $\square$

6 Generative Power of Context-Free Graph Grammars

In this section, we investigate the generative power of graph grammars with application conditions. In particular, we show that context-free graph grammars (also called edge-replacement grammars) with positive and negative application conditions are more powerful than ordinary context-free graph-grammars.

Definition 6.1 (*Graph grammar with application conditions*)

1. A graph grammar *with (positive, negative) application conditions* $GGA = (N, T, \hat{P}, S)$ consists of a pair $N = (N_V, N_E)$ of nonterminal label alphabets, a pair $T = (T_V, T_E)$ of terminal label alphabets, a finite set $\hat{P}$ of graph productions with (positive, negative) application conditions over $N \cup T$, and a start graph S over $N \cup T$. Moreover, we assume that the start graph S and all left-hand sides of productions are not completely terminally labeled.

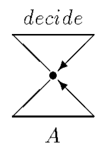
2. A graph grammar with application conditions $GGA = (N, T, \hat{P}, S)$ is called *context-free*, if all the productions of the grammar are context-free, i.e. if for all productions $\hat{p} = (L \xrightarrow{p} R, A(p))$ in $\hat{P}$, the left-hand side L is a graph of the form $(\{v_1, v_2\}, \{e\}, s, t, c, d)$ with $s(e) = v_1$ and $t(e) = v_2$, the preserved part $L_p = \{v_1, v_2\}$ of L, and all nodes in the graphs are labeled with a special symbol $\bullet$, which is left out in the following figures.

3. The graph language $L(GGA)$ generated by GGA consists of all terminally labeled graphs which can be derived from S applying rules of $\hat{P}$: $L(GGA) = \{G \in \mathcal{G}(T) | S \Longrightarrow^*_{\hat{P}} G\}$.

4. The class of graph languages generated by context-free graph grammars without, with positive, negative, positive and negative application conditions are denoted by $L(CF)$, $L(CF, pc)$, $L(CF, nc)$, and $\mathcal{L}(CF, pc, nc)$, respectively.

$\triangle$

Example 6.2 1. The set $FULLTREE = \{FULLTREE(n) | n \in \mathbb{N}_0\}$ of all full binary trees[9] of height n (for some $n \in \mathbb{N}_0$) can be generated by a context-free graph grammar with positive and negative application conditions. The grammar GGA for generating $FULLTREE$ is given as follows: The start graph S below



is a full binary tree of height 0 whose single node is decorated by an edge with label A. Moreover, for controlling purposes, it is decorated by an edge with label *decide*.

[9] A *full* binary tree of height n is a binary tree such that each of its nodes has two children, except that those at level n are all leaves.

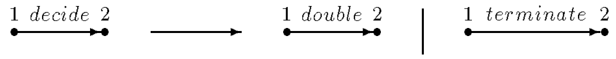


Figure 9: The decision productions $c_{decide,1}$ and $c_{decide,2}$.

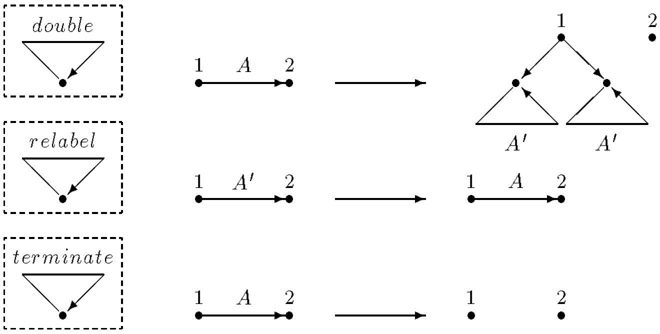


Figure 10: The main productions p_{double}, $p_{relabel}$ and $p_{terminate}$.

The production set consists of the decision productions $c_{decide,1}$ and $c_{decide,2}$ (figure 9), the main productions p_{double}, $p_{relabel}$, and $p_{terminate}$ (figure 10), and the control productions c_{double}, $c_{relabel}$, and $c_{terminate}$ (figure 11). The decision productions may be applied whenever there is an edge with label *decide*. The main productions only may be applied if a suitable positive application condition is satisfied. E.g. the existence of an edge with label *double*, *relabel*, and *terminate*, respectively, is required. The control productions only may be applied if a suitable negative application condition is satisfied. E.g. the existence of an edge with label A or with label A' is forbidden.

Obviously, GGA is a context-free graph grammar with positive and negative application conditions. Thus, it remains to prove that $L(GGA) = FULLTREE$. First, we will show that $FULLTREE \subseteq L(GGA)$. For symbols X, Y and $n \in I\!N_0$, let $FULLTREE_{X,Y}(n)$ denote the full binary tree of height n whose root is decorated by an X-edge and whose leaves are decorated by Y-edges. Moreover, let $FULLTREE_X(n)$ denote the full binary tree of height n whose root is decorated by an X-edge. Then we have the following (compare table 1).

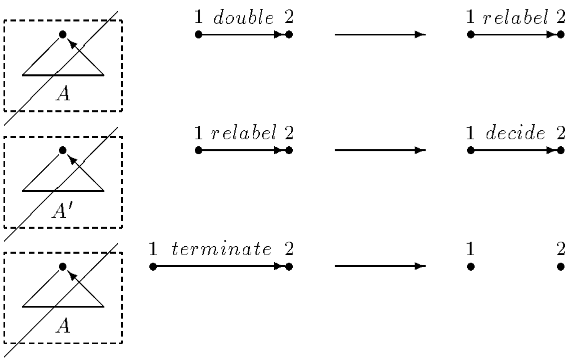


Figure 11: The control productions c_{double}, $c_{relabel}$ and $c_{terminate}$.

Table 1: Derivations in GGA.

Derivation			Production applied
$FULLTREE_{decide,A}(n)$	$\Longrightarrow$	$FULLTREE_{double,A}(n)$	$c_{decide,1}$
	$\Longrightarrow^*$	$FULLTREE_{double,A'}(n+1)$	p_{double} (2^n times)
	$\Longrightarrow$	$FULLTREE_{relabel,A'}(n+1)$	c_{double}
	$\Longrightarrow^*$	$FULLTREE_{relabel,A}(n+1)$	$p_{relabel}$ (2^{n+1} times)
	$\Longrightarrow$	$FULLTREE_{decide,A}(n+1)$	$c_{relabel}$
$FULLTREE_{decide,A}(n)$	$\Longrightarrow$	$FULLTREE_{terminate,A}(n)$	$c_{decide,2}$
	$\Longrightarrow^*$	$FULLTREE_{teminate}(n)$	$p_{terminate}$ (2^n times)
	$\Longrightarrow$	$FULLTREE(n)$	$c_{terminate}$

(a) $S = FULLTREE_{decide,A}(0)$.

(b) For $n \in I\!N_0$, $FULLTREE_{decide,A}(n) \Longrightarrow^* FULLTREE_{decide,A}(n+1)$.

(c) For $n \in I\!N_0$, $FULLTREE_{decide,A}(n) \Longrightarrow^* FULLTREE(n)$.

Consequently, $FULLTREE = \{FULLTREE(n) | n \in I\!N_0\} \subseteq L(GGA)$.

Now we will show that $L(GGA) \subseteq FULLTREE$. For a symbol X and $n, k, l \in I\!N_0$ with $k + l \leq 2^n$, let $FULLTREE_X(n, k, l)$ ($TREE_X(n, k, l)$) denote the set of all full binary trees (complete binary trees[10]) of height n where the root is decorated by an X-edge, k of its leaves are decorated by an A-edge, and l of its leaves are decorated by an A'-edge. Then we have the following. Whenever $S \Longrightarrow^* G$ is a derivation in GGA, then G is of one of the following forms.

(a) $G \in FULLTREE_{decide}(n, 2^n, 0)$

(b) $G \in TREE_{double}(n+1, k, l)$ where $2 \cdot k + l = 2^{n+1}$

(c) $G \in FULLTREE_{relabel}(n, k, l)$ where $k + l = 2^n$

(d) $G \in FULLTREE_{terminate}(n, k, 0)$ where $k \leq 2^n$

(e) $G = FULLTREE(n)$

This can be proved by induction on the length m of derivations $S \Longrightarrow^* G$. If $S \Longrightarrow^* G$ is a derivation of length 0, then $G \in FULLTREE_{decide}(0, 1, 0)$. If $S \Longrightarrow^* G$ is a derivation of length $m+1$ and $S \Longrightarrow^* G' \Longrightarrow G$ the decomposition of $S \Longrightarrow^* G$ into a derivation of length m and a direct derivation $G' \Longrightarrow G$, then, by the induction hypothesis, G' is of one of the mentioned forms. Looking at table 2 we see that G is of the required form, too.

In particular, we get that whenever $S \Longrightarrow^* G$ is a derivation in GGA and G is terminally labeled, then $G = FULLTREE(n)$ for some $n \in I\!N_0$. Consequently, $L(GGA) \subseteq FULLTREE$.

2. The set $FULLTREE$ can be generated by a context-free graph grammar GGA' with negative application conditions, only. GGA' may be obtained from the grammar GGA in example 6.2.1 by replacing the main productions p_{double}, $p_{relabel}$, and

[10]A binary tree of height n is *complete* if at every level i, $0 \leq i \leq (n-1)$, it has exactly 2^i nodes, and if all nodes at level $(n-1)$ have two or zero children.

Table 2: Properties of derived graphs.

G' belongs to	G belongs to	Production applied
$FULLTREE_{decide}(n, 2^n, 0)$	$TREE_{double}(n+1, 2^n, 0)$	$c_{decide,1}$
$FULLTREE_{decide}(n, 2^n, 0)$	$FULLTREE_{terminate}(n, 2^n, 0)$	$c_{decide,2}$
$TREE_{double}(n, k, l)$	$TREE_{double}(n, k-1, l+2)$	p_{double} if $k \geq 1$
$TREE_{double}(n, 0, 2^n)$	$FULLTREE_{relabel}(n, 0, 2^n)$	c_{double}
$FULLTREE_{relabel}(n, k, l)$	$FULLTREE_{relabel}(n, k+1, l-1)$	$p_{relabel}$ if $l \geq 1$
$FULLTREE_{relabel}(n, 2^n, 0)$	$FULLTREE_{decide}(n, 2^n, 0)$	$c_{relabel}$
$FULLTREE_{terminate}(n, k, 0)$	$FULLTREE_{terminate}(n, k-1, 0)$	$p_{terminate}$ if $k \geq 1$
$FULLTREE_{terminate}(n, 0, 0)$	$\{FULLTREE(n)\}$	$c_{terminate}$

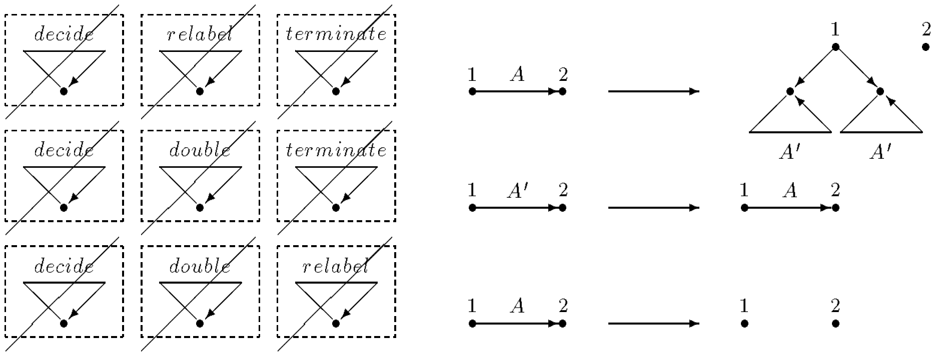


Figure 12: The main productions p'_{double}, $p'_{relabel}$ and $p'_{terminate}$.

$p_{terminate}$ (see figure 10) by the productions p'_{double}, $p'_{relabel}$, and $p'_{terminate}$ given in figure 12. Obviously, GGA' is a context-free graph grammar with negative application conditions, only. Moreover, $L(GGA') = L(GGA) = FULLTREE$ because the start graphs S and S' of the grammars are equal and whenever, a graph G is derived by GGA or GGA', then a production p_X with positive application condition is applicable to G if and only if the corresponding production p'_X with negative application condition is applicable to G. Thus, $FULLTREE$ is in $\mathcal{L}(CF, nc)$.

3. Up to now, it is not known whether or not the set $FULLTREE$ can be generated by a context-free graph grammar GGA'' with positive application conditions, only. We conjecture that $FULLTREE$ is not in $\mathcal{L}(CF, pc)$.

$\triangle$

Remark 6.3 In formal language theory, there are several approaches for regulated rewriting which are closely related to graph grammars with application conditions (see e.g. Salomaa [22], Dassow, Păun [7] and Csuhaj-Varjú [6]).

1. A *context-sensitive grammar* $G = (N, T, P, S)$ is a string grammar in which all productions in P are of the form $u_1 A u_2 \to u_1 x u_2$ where $A \in N$, $x \in (N \cup T)^+$, $u_1, u_2 \in (N \cup T)^*$ with the possible exception of $S \to \lambda$ which occurrence requires that S does not appear in the right-hand side of any production (see [7], page 13). For such a grammar G, there is a context-free graph grammar $graph(G)$ with

positive application conditions such that $graph(L(G)) = L(graph(G))$[11]. The grammar is constructed as follows: $graph(S)$ is the start graph and whenever we have a production of the form $u_1Au_2 \to u_1xu_2$ in P, then the graph production $graph(A) \to graph(x)$ with the positive constraint $graph(A) \to graph(u_1Au_2)$ is in the production set of $graph(G)$. If $S \to \lambda$ is in P, then the graph production $graph(S) \to graph(\lambda)$ is added to the production set of $graph(G)$.

2. A *random context grammar* is a system $G = (N, T, P, S)$ where N, T, S are as in a usual string grammar, and P is a finite set of rules of the form $(\alpha \to \beta, Q, R)$, where $\alpha \to \beta$ is a rewriting rule over $N \cup T$ and Q and R are subsets of N (see [7], page 30). For $w, w' \in (N \cup T)^*$, we write $w \Longrightarrow w'$ if and only if $w = w_1\alpha w_2$, $w' = w_1\beta w_2$, for some $w_1, w_2 \in (N \cup T)^*$, $(\alpha \to \beta, Q, R) \in P$, all symbols of Q appear in w_1w_2, and no symbol of R appears in w_1w_2. Q is called the permitting context of $\alpha \to \beta$ and R is the forbidding context of this rule. The language generated by G is defined as $L(G) = \{w \in T^* | S \Longrightarrow^* w\}$ where $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$. For such a grammar G, there is a graph grammar with positive and negative application conditions such that $graph(L(G)) = L(graph(G))$. The grammar is constructed as follows: $graph(S)$ is the start graph and whenever we have a production of the form $(\alpha \to \beta, Q, R)$ in P, then the graph production $graph(\alpha) \to graph(\beta)$ with the positive application condition $\{graph(\alpha) \to graph(\alpha) + graph(q) | q \in Q\}$ and the negative application condition $\{graph(\alpha) \to graph(\alpha) + graph(r) | r \in R\}$ is in the production set of $graph(G)$. (Note that *all* positive and *all* negative application constraints have to be satisfied. Moreover the required (and the forbidden) morphisms have to be injective on the edge set.)

3. A *string random context grammar of degree* (i, j), $i, j \geq 0$, is a system $G = (N, T, P, S)$ where N, T, S are as in a usual string grammar, and P is a finite set of rules of the form $(A \to x, \alpha, \beta)$, where $A \to x$ is a context-free rule over $N \cup T$, $\alpha \in (N \cup T)^*$ is a string of length $\leq i$ and $\beta \in (N \cup T)^*$ is a string of length $\leq j$ (see [7], page 86). Such a rule can be applied to a string w if and only if the string α (if $\alpha \neq \lambda$) is a substring of w and the string β (if $\beta \neq \lambda$) is not a substring of w. For such a grammar G, there is a graph grammar with positive and negative application conditions such that $graph(L(G)) = L(graph(G))$. The grammar is constructed as follows: $graph(S)$ is the start graph and whenever we have a rule of the form $(A \to x, \alpha, \beta)$ in P, then the graph production $graph(A) \to graph(x)$ with the positive constraint $graph(A) \to graph(A) + graph(\alpha)$ and the negative constraint $graph(A) \to graph(A) + graph(\beta)$ is in the production set of $graph(G)$. (Note that the required (forbidden) morphism may be non-injective.)

While in formal language theory, the types of regulation, matrix, programmed, and random context grammars, are intensively studied, up to now, there are only some investigations concerning regulated graph grammars. The only known papers in this direction are the ones on

1. graph grammars with application conditions (see Pfaltz, Rosenfeld [20], Montanari [19], von Solms [24] Ehrig, Habel [9], Kreowski [15]).

[11] For a non-empty string $w = a_1 \dots a_n$, $graph(w)$ denotes a graph of the form $(\{v_0, v_1, \dots, v_n\}, \{e_1, \dots, e_n\}, s, t, c, d)$ with $s(e_i) = v_{i-1}$, $t(e_i) = v_i$, $c(v_i) = \bullet$, and $d(e_i) = a_i$ for $i = 1, \dots, n$. For the empty string λ, $graph(\lambda)$ denotes the graph with one node and no edges. For a string language L, $graph(L)$ denotes the set $\{graph(w) | w \in L\}$.

2. programmed graph grammars (see Bunke [4],[5]),

3. graph grammars with priorities (see Litovsky and Métivier [17]),

Nevertheless, a number of results can be adapted from formal language theory. In particular, it is not surprising that context-free graph grammars with application conditions are more powerful than ordinary context-free graph grammars without application conditions and that the same holds true, if only negative (positive) conditions are allowed. $\triangle$

In the following, we investigate the generative power of context-free graph grammars, also called edge-replacement grammars, equipped with application conditions.

Theorem 6.4 $\mathcal{L}(CF) \subsetneq \mathcal{L}(CF, pc, nc)$. $\triangle$

Proof: As shown in example 6.2.1, the set $FULLTREE$ of all full binary trees of height n (for some $n \in \mathbb{N}_0$) can be generated by a context-free graph grammar with positive and negative application conditions. On the other hand, $FULLTREE$ is a non-semilinear language. Thus, by Parikh's theorem for context-free graph languages (see [14]), $FULLTREE$ cannot be generated by a context-free graph grammar without application conditions. $\square$

Theorem 6.5 $\mathcal{L}(CF) \subsetneq \mathcal{L}(CF, nc)$. $\triangle$

Proof: As shown in example 6.2.2, the set $FULLTREE$ of all full binary trees of height n (for some $n \in \mathbb{N}_0$) can be generated by a context-free graph grammar with negative application conditions. On the other hand, $FULLTREE$ is a non-semilinear language. Thus, by Parikh's theorem for context-free graph languages (see [14]), $FULLTREE$ cannot be generated by a context-free graph grammar without application conditions. $\square$

Theorem 6.6 $\mathcal{L}(CF) \subsetneq \mathcal{L}(CF, pc)$. $\triangle$

Proof: The set $L = \{a^{2^n} | n \in \mathbb{N}_0\}$ of all strings over $\{a\}$ of length 2^n (for some $n \in \mathbb{N}_0$) is a context-sensitive string language. The corresponding string-graph language $graph(L) = \{graph(w) | w \in L\}$ can be generated by a context-free graph grammar with positive application conditions. On the other hand, $graph(L)$ is a non-semilinear language. Thus, by Parikh's theorem for context-free graph languages (see [14]), $graph(L)$ cannot be generated by a context-free graph grammar without application conditions. $\square$

Remark 6.7 The following questions remain:

1. $\mathcal{L}(CF, pc) \subsetneq \mathcal{L}(CF, pc, nc)$?

2. $\mathcal{L}(CF, nc) \subsetneq \mathcal{L}(CF, pc, nc)$?

3. How $\mathcal{L}(CF, pc)$ and $\mathcal{L}(CF, nc)$ are related?

Influenced by some problems and results from formal language theory on string random context grammars with context-free productions (see [7]), we conjecture that $\mathcal{L}(CF, pc) \subsetneq \mathcal{L}(CF, pc, nc)$ and that $\mathcal{L}(CF, pc)$ and $\mathcal{L}(CF, nc)$ are incomparable. $\triangle$

7 Conclusions

In this paper, the concept of application conditions, introduced in Ehrig, Habel [9], is restricted to a specific type of left-sided application conditions which can be specified by sets of graphs (and graph morphisms). The following other types of application conditions seem to be important.

1. Structural Contextual Conditions. Beside *concrete contextual conditions* — considered in this paper — one may be interested in expressing *structural conditions*, concerning the existence or non-existence of a homoeomorphic image of $\hat{L}$. In particular, the existence or non-existence of a path (the homoeomorphic image of an edge), a circuit (the homoeomorphic image of a circuit of length 3), a non-planar subgraph (the homoeomorphic image of a K_5 or a $K_{3,3}$), etc. are of interest. One way to specify this is to define the set of all graphs of the structure $\hat{L}$, which is infinite in general. Another way is to consider a containment relation like the one used by Wankmüller [25]. Then we have the handle containment relations instead of morphisms.

2. Cardinality Conditions. Cardinality conditions concerning for example the existence or non-existence of a number of ingoing or outgoing edges for a given node may be important. They can be expressed using application conditions with injective satisfaction in the sense of definition 3.4. The results concerning consistency and context enlargement of application conditions as well as independence and parallelism of derivations with application conditions can be obtained for application conditions with injective satisfaction, too. This requires more technical effort and will be studied in the future.

3. Right-Sided Application Conditions. For practical reasons, it seems to be convenient to allow right-sided application conditions, too. For example, one may be interested to require that the resulting graph is of degree $\leq k$. The question remains, under which conditions right-sided application condition may be expressed by equivalent left-sided ones.

Graph grammars with application conditions provide one possibility to restrict the applicability of productions. Programmed graph grammars (see Bunke [4], [5]) and graph grammars with priorities (see Litovsky and Métivier [17]) provide other possibilities. In the first case, an application of a production may determine which productions are applicable at the next step; in the second case, a finite set of graph productions is equipped with a partial order, called priority, and a production may be applied on an occurrence if the occurrence of the left-hand side of the production is not overlapping with an occurrence of a left-hand side of a more priority production. It depends on the kind of application which method should be taken, and it may also be useful to combine both of them.

Another direction of future research is given by the results from algebraic graph-grammar theory that are still missing for graph grammars with application conditions.

1. Embedding of Derivations. As already mentioned we are going to provide a test of independence of productions with application conditions, i.e. given a set of productions we want to know all critical pairs. Furthermore, there is hope for a sufficient condition for embedding of derivations with application conditions.

Together this could give us the possibility to analyse local confluence of production sets (at least in some cases).

2. Amalgamation & Distribution of Derivations. Results on amalgamation (see [3]) and distributed graph grammars (see [12]) should be possible in a similar way as the Parallelism Theorem since first of all they also require the translation of application conditions via total morphisms.

The results on local confluence and parallelism can be obtained completely independent from the definition of constraints and satisfaction in the framework of institutions (see [13]). In particular the application conditions considered here build an institution if we take $\mathbf{Graphs}^{\mathbf{T}}_{\mathbf{LAB}}$ as the category of signatures and constraints as sentences. Then lemma 3.7 turns out to be the satisfaction condition and is sufficient to prove the local confluence and the Parallelism Theorem.

Because of their categorical nature, application conditions can easily be adopted to high-level replacement systems (see [10], [11]) as well as to graph structures (see [18]). Doing so they become available to a big variety of transformation systems.

References

[1] Jiří Adámek, Horst Herrlich, George Strecker. Abstract and Concrete Categories. John Wiley & Sons, New York, 1990.

[2] Michael A. Arbib, Ernest G. Manes. Arrows, Structures, and Functors. Academic Press, New York, 1975.

[3] Paul Böhm, Harald-Reto Fonio, Annegret Habel. Amalgamation of graph transformations: A synchronization mechanism. Journal of Computer and System Sciences 34, 377–408, 1987.

[4] Horst Bunke. Programmed graph grammars. In V. Claus, H. Ehrig, G. Rozenberg, eds., Graph-Grammars and Their Application to Computer Science and Biology, Lecture Notes in Computer Science 73, 155–166, 1979.

[5] Horst Bunke. On the generative power of sequential and parallel programmed graph grammars. Computing 29, 89–112, 1982.

[6] Erzsébet Csuhaj-Varjú. On grammars with local and global context conditions. Intern. J. Computer Math. 47, 17–27, 1993.

[7] Jürgen Dassow, Gheorghe Păun. Regulated Rewriting in Formal Language Theory, volume 18 of EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1989.

[8] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, G. Rozenberg, eds., Graph-Grammars and Their Application to Computer Science and Biology, Lecture Notes in Computer Science 73, 1–69, 1979.

[9] Hartmut Ehrig, Annegret Habel. Graph grammars with application conditions. In G. Rozenberg, A. Salomaa, eds., The Book of L, 87–100. Springer-Verlag, Berlin, 1986.

[10] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, Francesco Parisi-Presicce. Parallelism and concurrency in high level replacement systems. Mathematical Structures in Computer Science 1, 361–404, 1991.

[11] Hartmut Ehrig, Michael Löwe. Categorical Principles, Techniques and Results for High-Level-Replacement Systems in Computer Science. In Applied Categorical Structures, volume 1, 1993. Kluwer Academic Publishers.

[12] Hartmut Ehrig, Michael Löwe. Parallel and distributed derivations in the single-pushout approach. Theoretical Computer Science 109, 123–143, 1993.

[13] Joseph A. Goguen, Rod Burstall. Institutions, Abstract Model Theory for Specification and Programming. JACM, January 1990

[14] Annegret Habel. Hyperedge Replacement: Grammars and Languages, volume 643 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.

[15] Hans-Jörg Kreowski. Five facets of hyperedge replacement beyond context-freeness. In Z. Ésik, ed., Fundamentals of Computation Theory, Lecture Notes in Computer Science 710, 69–86, 1993.

[16] Hans-Jörg Kreowski, Grzegorz Rozenberg. On structured graph grammars, I and II. Information Sciences 52, 185–210 and 221–246, 1990.

[17] Igor Litovsky, Yves Métivier. Computing with graph rewriting systems with priorities. Theoretical Computer Science 115, 191–224, 1993.

[18] Michael Löwe. Algebraic approach to single-pushout graph transformation. Theoretical Computer Science 109, 181–224, 1993.

[19] Ugo Montanari. Separable graphs, planar graphs and web grammars. Information and Control 16, 243–267, 1970.

[20] John L. Pfaltz, Azriel Rosenfeld. Web grammars. In Int. Joint Conference on Artificial Intelligence, 609–619, 1969.

[21] Terrence W. Pratt. Pair grammars, graph languages and string-to-graph translations. Journal of Computer and System Sciences 5, 560–595, 1971.

[22] Arto K. Salomaa. Formal Languages. Academic Press, New York, 1973.

[23] Andy Schürr. PROGRESS: A VHL-Language Based on Graph Grammars. Lecture Notes in Computer Science, 532, 641–659, 1991.

[24] S.H. von Solms. Node-label controlled graph grammars with context conditions. Intern. J. Computer Math. 15, 39–49, 1984.

[25] Frank Wankmüller. Characterization of graph classes by forbidden structures and reductions. Lecture Notes in Computer Science 153, 405–414, 1983.