Contextual Hyperedge Replacement

Frank Drewes[1], Berthold Hoffmann[2], and Mark Minas[3]

[1] Umeå universitet, Sweden
[2] DFKI Bremen and Universität Bremen, Germany
[3] Universität der Bundeswehr München, Germany

Abstract. In model-driven design, the structure of software is commonly specified by meta-models like UML class diagrams. In this paper we study how graph grammars can be used for this purpose. We extend context-free hyperedge-preplacement—which is not powerful enough for this application—so that rules may not only access the nodes attached to the variable on their left-hand side, but also nodes elsewhere in the graph. Although the resulting notion of contextual hyperedge replacement preserves many properties of the context-free case, it has considerably more generative power—enough to specify software models that cannot be specified by class diagrams alone.

1 Introduction

Graphs are ubiquitous in science and beyond. When graph-like diagrams are used to model system development, it is important to define precisely whether a diagram is a valid model or not. Often, models are defined as the valid instantiations of a *meta-model*, e.g, the valid object diagrams for a class diagram in UML. A meta model is convenient for capturing requirements as it can be refined gradually. It is easy to check whether a given model is valid for a meta-model. However, it is not easy to construct valid sample models for a meta-model, and they give no clue how to define transformations on all valid models. Also, their abilities to express structural properties (like connectedness) are limited; textual constraints (e.g., in OCL) have to be used in such cases.

In contrast to meta-models, *graph grammars* derive sets of graphs constructively, by applying rules to a start graph. This kind of definition is strict, can easily produce sample graphs by derivation, and its rules provide for a recursive structure to define transformations on the derivable graphs. However, it shall not been concealed that validating a given graph, by *parsing*, may be rather complex.

General graph grammars generate all recursively enumerable sets of graphs [19] so that there can be no parsing algorithm. Context-free graph grammars based on node replacement or hyperedge replacement [8] do not have the power to generate graphs of general connectivity, like the language of all graphs, of all acyclic, and all connected graphs etc. From this, we conclude that practically useful kinds of graph grammars should lie in between context-free and general ones. Hyperedge replacement is a solid basis for devising such grammars, as it

has a comprehensive theory, and is very simple: A step removes a variable (represented as a hyperedge) and glues the fixed ordered set of nodes attached to it to distinguished nodes of a graph. The authors have been working on several extensions of hyperedge replacement. *Adaptive star replacement* [3], devised with D. Janssens and N. Van Eetvelde, allows variables to be attached to arbitrary, unordered sets of nodes. Their generative power is sufficient for defining sophisticated software models like program graphs [6]. Nevertheless, they inherit some of the strong properties of hyperedge replacement. Unfortunately, adaptive star rules tend to have many edges, which makes them hard to understand—and to construct. Therefore the authors have devised *contextual graph grammars*, where variables still have a fixed, ordered set of attached nodes, but replacement graphs may be glued, not only with these attachments, but also with nodes occurring elsewhere in the graph, that have been derived in earlier derivation steps [13]. As we shall show, their generative power suffices to define non-context-free models. Typically, contextual rules are only modest extensions of hyperedge replacement rules, and are significantly easier to write and understand than adaptive star rules. This qualifies contextual hyperedge grammars as a practical notation for defining software models. When we add application conditions to contextual rules, as we have done in [13], even subtler software models can be defined. Since conditions are a standard concept of graph transformation, which have been used in many graph transformation systems (see, e.g., PROGRES [18]), such rules are still intuitive.

This paper aims to lay a fundament to the study of contextual hyperedge replacement. So we just consider grammars without application conditions for the moment, as our major subjects of comparison, context-free hyperedge replacement and adaptive star replacement, also do not have them. With context-free hyperedge replacement, contextual hyperedge replacement shares deciability results, characterisations of their generated language, and the existence of a parsing algorithm. Nevertheless, it is powerful enough to make it practically useful for average structural models. If it is extended by recursive application conditions [11], which allow to express requirements regarding the (non-) existence of paths in the graph, it reaches the expressiveness of adaptive star grammars.

The remainder of this paper is structured as follows. In Section 2 we introduce contextual hyperedge replacement grammars and give some examples. Normal forms for these grammars are presented in Section 3. In Section 4 we discuss some of their limitations wrt. language generation, and sketch parsing in Section 5. We conclude with some remarks on related and future work in Section 6.

2 Graphs, Rules, and Grammars

In this paper, we consider directed and labeled graphs. We only deal with abstract graphs in the sense that graphs that are equal up to renaming of nodes and edges are not distinguished. In fact, we use hypergraphs with a generalized notion of edges that may connect any number of nodes, not just two. Such edges will also be used to represent variables in graphs and graph grammars.

We consider labeling alphabets $\mathcal{C} = \dot{\mathcal{C}} \uplus \bar{\mathcal{C}} \uplus X$ that are sets whose elements are the *labels* (or "*colors*") for nodes, edges, and variables, with an *arity* function $\mathit{arity}\colon \bar{\mathcal{C}} \uplus X \to \dot{\mathcal{C}}^*$.[4]

A *labelled hypergraph over* $\mathcal{C}$ (*graph*, for short) $G = \langle \dot{G}, \bar{G}, \mathit{att}_G, \dot{\ell}_G, \bar{\ell}_G \rangle$ consists of disjoint finite sets $\dot{G}$ of *nodes* and $\bar{G}$ of *hyperedges* (*edges*, for short) respectively, a function $\mathit{att}_G\colon \bar{G} \to \dot{G}^*$ that *attaches* sequences of pairwise distinct nodes to edges so that $\dot{\ell}^*_G(\mathit{att}_G(e)) = \mathit{arity}(\bar{\ell}_G(e))$ for every edge $e \in \bar{G}$,[5] and *labelling* functions $\dot{\ell}_G\colon \dot{G} \to \dot{\mathcal{C}}$ and $\bar{\ell}_G\colon \bar{G} \to \bar{\mathcal{C}} \uplus X$. Edges are called *variables* if they carry a variable name as a label; the set of all graphs over $\mathcal{C}$ is denoted by $\mathcal{G}_{\mathcal{C}}$.

For a graph G and hyperedge $e \in \bar{G}$, we denote by $G - e$ the graph obtained by removing e from G. Similarly, for $v \in \dot{G}$, $G - v$ is obtained by removing v from G (together with all edges attached to v).

Given graphs G and H, a *morphism* $m\colon G \to H$ is a pair $m = \langle \dot{m}, \bar{m} \rangle$ of functions $\dot{m}\colon \dot{G} \to \dot{H}$ and $\bar{m}\colon \bar{G} \to \bar{H}$ that preserves labels and attachments:

$$\dot{\ell}_H \circ \dot{m} = \dot{\ell}_G, \bar{\ell}_H \circ \bar{m} = \bar{\ell}_G, \mathit{att}_H(\bar{m}(e)) = \dot{m}^*(\mathit{att}_G(e)) \text{ for every } e \in \bar{G}$$

As usual, a morphism $m\colon G \to H$ is *injective* if both $\dot{m}$ and $\bar{m}$ are injective.

The replacement of variables in graphs by graphs is performed by applying a special form of standard double-pushout rules [7].

Definition 1 (Contextual Rule). A *contextual rule* (*rule*, for short) $r = (L, R)$, consists of graphs L and R over $\mathcal{C}$ such that

- the *left-hand side* L contains exactly one edge x, which is required to be a variable (i.e., $\bar{L} = \{x\}$ with $\bar{\ell}_L(x) \in X$) and
- the *right-hand side* R is an arbitrary supergraph of $L - x$.

Nodes in L that are attached to x are its *neigbors*,[6] whereas the others are the *contextual nodes* of L (and of r); r is *context-free* if it has no contextual nodes. (Context-free rules are known as hyperedge replacement rules in the literature [10,2].)

Let r be a contextual rule as above, and consider some graph G. An injective morphism $m\colon L \to G$ is called a *matching* for r in G. The *replacement* of the variable $m(x) \in G$ by R (via m) is the graph H obtained from the disjoint union of $G - m(x)$ and R by identifying every node $v \in \dot{L}$ with $m(v)$. We write this as $H = G[R/m]$.

Note that contextual rules are equivalent to contextual star rules as introduced in [13], however without application conditions.

The notion of rules introduced above gives rise to a class of graph grammars. We call these grammars contextual hyperedge-replacement grammars, or briefly contextual grammars.

[4] A^* denotes the set of finite sequences over a set A; the empty sequence is denoted by ε.

[5] For a function $f\colon A \to B$, its extension $f^*\colon A^* \to B^*$ to sequences is defined by $f^*(a_1, \ldots, a_n) = f(a_1) \ldots f(a_n)$, for all $a_i \in A$, $1 \leqslant i \leqslant n$, $n \geqslant 0$.

[6] **Bert:** *Nachbarn brauche ich für replizierbare Teilgraphen.*

$$\mathsf{N} = \left\{ \boxed{\mathsf{N}} \underset{0|x}{::=} \langle\rangle \;\middle|\; (x)\ \boxed{\mathsf{N}} \;\middle|\; x \in \dot{\mathcal{C}} \right\} \qquad Z = \boxed{\mathsf{N}}\ \boxed{\mathsf{E}}$$

$$\mathsf{E} = \left\{ \boxed{\mathsf{E}} \underset{2}{::=} \boxed{\mathsf{E}}\ \boxed{\mathsf{E}},\ \overset{0}{(x)}\ \boxed{\mathsf{E}}\ \overset{1}{(y)} \underset{a}{::=} \overset{0}{(x)} \xrightarrow{a} \overset{1}{(y)} \;\middle|\; x, y \in \dot{\mathcal{C}}, a \in \bar{\mathcal{C}} \setminus X, arity(a) = xy \right\}$$

Fig. 1. Rules (generating the language of all graphs)

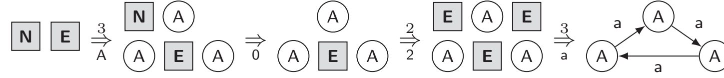

Fig. 2. A derivation of a graph with the rules in Figure 1

Definition 2 (Contextual Hyperedge-Replacement Grammar). A *contextual hyperedge-replacement grammar* (*contextual grammar*, for short) is a triple $\Gamma = \langle \mathcal{C}, \mathcal{R}, Z \rangle$ consisting of

- a finite labeling alphabet $\mathcal{C}$,
- a finite set $\mathcal{R}$ of rules, and
- a start graph $Z \in G_{\mathcal{C}}$.

If $\mathcal{R}$ contains only context-free rules, then Γ is a *hyperedge replacement grammar*. We let $G \Rightarrow_{\mathcal{R}} H$ if $H = G[R/m]$ for some rule (L, R) and for a matching $m\colon L \to G$. Now, the language generated by Γ is given by

$$\mathcal{L}(\Gamma) = \{G \in \mathcal{G}_{\mathcal{C}\setminus X} \mid Z \Rightarrow^*_{\mathcal{R}} G\}.$$

Contextual grammars Γ and Γ' are *equivalent* if $\mathcal{L}(\Gamma) = \mathcal{L}(\Gamma')$. The classes of graph languages generated by hyperedge-replacement grammars and contextual grammars are denoted by HR and CHR, respectively.

Notation (Drawing Conventions for Graphs and Rules). Graphs are drawn as in Figure 2 and Figure 4. Circles and boxes represent nodes and edges, respectively. The text inscribed to them is their label from $\mathcal{C}$. (If all nodes carry the same label, these are just omitted.) The box of an edge is connected to the circles of its attached nodes by lines; the attached nodes are ordered counter-clockwise around the edge, starting in its north. The boxes of variables are drawn in gray. Terminal edges with two attached nodes may also be drawn as arrows from the first to the second attached node. In this case, the edge label is ascribed to the arrow.

In figures, a contextual rule $r = (L, R)$ is drawn as $L ::= R$. Small numbers above nodes indicate identities of nodes in L and R. $L ::= R_1|R_2 \cdots$ is short for rules $L ::= R_1, L ::= R_2, \ldots$ with the same left-hand side. Subscripts "n" or "n|m$\cdots$" below the symbol $::=$ define names that are used to refer to rules in figures of derivations. See Figure 1 and Figure 3.

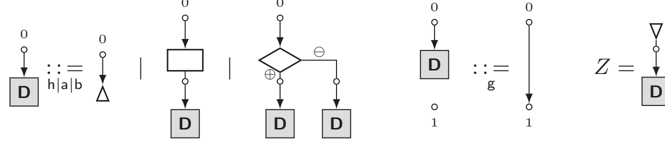

Fig. 3. Rules generating unrestricted control flow diagrams

Example 1 (The Language of All Graphs). The rules shown in Figure 1 generate the set $\mathcal{A}$ of loop-free labeled graphs with binary edges. The rule set N (which is context-free) generates the nodes, and the rule set E inserts the outgoing edges from a node to another node, which is required to exist in the context. The derivation in Figure 2 produces a triangular graph with three nodes and edges.

It is well known that the language $\mathcal{G}_\mathcal{C}$ of all graphs over $\mathcal{C}$ cannot be defined by hyperedge-replacement grammars [10, Chapter IV, Theorem 3.12(1)]. Thus, as CHR contains HR by definition, we have:

Observation 1. $\mathsf{HR} \subsetneq \mathsf{CHR}$.

Context-free hyperedge-replacement grammars can be used to generate the sets of all structured and semi-structured control flow diagrams [2], but contextual grammars are needed for unrestricted control flow diagrams (that have unbounded tree-width and can, therefore, not be generated by context-free hyperedge replacement).

Example 2 (Control Flow Diagrams). Unrestricted control flow diagrams represent sequences of low-level instructions according to a syntax like this:

$$I ::= [\ell :]\, \textbf{halt} \quad | \quad [\ell :]\, x := E \quad | \quad [\ell_1 :]\, \textbf{if } E \textbf{ then goto } \ell_2 \quad | \quad [\ell_1 :]\, \textbf{goto } \ell_2$$

The rules in Figure 3 generate unrestricted flow diagrams. The first three rules, h, a, and b, generate control flow trees, and the fourth rule g, which is not context-free, inserts gotos to a program state in the context. In Figure 4, these rules are used to derive an “ill-structured” flow diagram.

3 Normal Forms of Contextual Grammars

In this section, we study the basic properties of contextual grammars . As it turns out, these properties are not fundamentally different from the properties known for the context-free case. This indicates that contextual hyperedge replacement is a modest generalization of hyperedge replacement that, to the extent one might reasonably hope for, has appropriate computational properties.

Let us first look at some normal forms of contextual grammars. We say that a restricted class C of contextual grammars is a normal form of contextual grammars (of a specified type) if, for every contextual grammar (of that type), one can effectively construct an equivalent grammar in C.

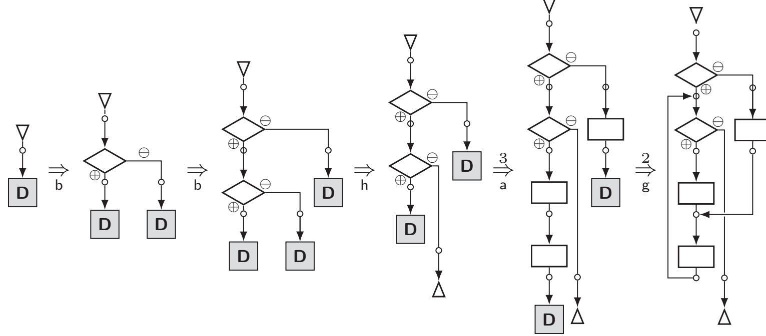

Fig. 4. A derivation of an unstructured control flow diagrams

Lemma 1. *Contextual grammars in which each rule contains at most one contextual node are a normal form of contextual grammars.*

Proof. Straightforward, by collecting contextual nodes one by one using chain rules (see below), thus turning all original rules into context-free rules. □

In the context-free case, so-called epsilon rules and chain rules can easily be removed from a grammar. A similar modification is possible for contextual grammars. In this context, a rule (L, R) with $\bar{L} = \{x\}$ is an *epsilon rule* if $R = L - x$, and a chain rule if $R - y = L - x$ for a variable $y \in \bar{R}$. Note that both epsilon and chain rules are more general than in the context-free case, because L may contain contextual nodes. In particular, chain rules can make use of these contextual nodes to "move" a variable through a graph. In the case of epsilon rules, the effect of contextual nodes is that the removal of a variable is subject to the condition that certain node labels are present in the graph.

Lemma 2. *Contextual grammars with neither epsilon nor chain rules are a normal form of contextual grammars that do not generate the empty graph.*

Proof Sketch. Assume that $\Gamma = \langle \mathcal{C}, \mathcal{R}, Z \rangle$ is a contextual grammar, and let $\mathcal{R}_{\text{chain}}$ and $\mathcal{R}_{\varepsilon}$ be the sets of epsilon and chain rules of Γ, respectively.

We first describe how rules can be composed by, intuitively, applying the second one to a variable in the first one. For this, consider rules $r_1 = (L_1, R_1)$ and $r_2 = (L_2, R_2)$, such that R_1 contains a variable with the same name as the variable in L_2. We want to be able to combine both rules even if R_1 does not supply r_2 with all the necessary contextual nodes. For this to be possible, we have to enrich L_1 with the contextual nodes needed by r_2. However, we want to do this in an economical way. Intuitively, if r_1 contains nodes that are isolated in both L_1 and R_1 (we call such nodes *free*), we can just as well use them rather than introducing even more contextual nodes. This is formalized next, in the notion of combinators.

A *combinator* of r_1 and r_2 is an injective morphism $c\colon L_2' \to R_1$ such that

– the graph L_2' is a subgraph of L_2 that includes the variable of L_2, and
– no free node of r_1 that is outside the image of c has the same label as a node in $\dot{L}_2 \setminus \dot{L}_2'$.

Given such a combinator, the combined rule $r_1 \bullet_c r_2 = (L, R)$ is constructed as follows: Let $r' = (L', R')$ be obtained from r_1 by disjointly adding the nodes in $\dot{L}_2 \setminus \dot{L}_2'$ to L_1 and R_1 (as isolated nodes). Let $c' \colon L_2 \to R'$ be the extension of c to L_2 and R' being the identity on $\dot{L}_2 \setminus \dot{L}_2'$. Then $R = R'[R_2/c']$.

For sets $\mathcal{R}_1, \mathcal{R}_2$ of rules, we let $\mathcal{R}_1 \bullet \mathcal{R}_2$ be the set of all $r_1 \bullet_c r_2$, for all $r_1 \in \mathcal{R}_1$, $r_2 \in \mathcal{R}_2$, and suitable combinators c. Moreover, let $\mathcal{R}_1^{\bullet}$ be the closure of $\mathcal{R}_1$ under $\bullet_c$, for arbitrary combinators c. Note that, for sets $\mathcal{R}_1$ of chain and epsilon rules, $\mathcal{R}_1^{\bullet}$ is finite if $\mathcal{R}_1$ is.

Now, let us sketch how to remove epsilon rules. For simplicity, we assume that, for all $\xi \in X$, there is a graph L_ξ with the following property: If (L, R) is a rule in $\mathcal{R}$ such that the unique variable in L is labeled with ξ, then $L = L_\xi$. This can, in fact, be shown to be another normal form of contextual grammars and is, thus, no restriction.

Now, let k be the maximal number of nodes with the same label in the graphs L_ξ. We observe first that, in every derivation, it can be assumed that derivation steps using a number of chain rules followed by an epsilon rule occur only after all steps using other rules. Thus, to know whether an epsilon rule will finally be applicable to a given variable, it suffices to know how many nodes of each label will finally be generated. Therefore, we modify Γ in such a way that it implements a guess-and-verify strategy, where every variable name ξ is equipped with mappings $g_\xi, l_\xi \colon \dot{\mathcal{C}} \to [k]$. Intuitively, $g_\xi(a) = m$ means that (it has been guessed that) the graph generated will eventually contain at least m nodes labeled with a. This information is simply propagated through the derivation. For the verification part, $l_\xi(a) = n \leqslant m$ means that the sub-derivation resulting from the current variable will generate (i.e., still has to generate) n of these m nodes. Now, consider the set X_0 of all variable names that appear as labels in the left-hand sides of rules in $\mathcal{R}_{\text{chain}}^{\bullet} \bullet \mathcal{R}_\varepsilon$. In other words, these are the names of variables for which an epsilon rule $(L_\xi, L_\xi - x)$ exists. We consider every rule $r = (L ::=, R)$ such that all names ξ of variables in R satisfy $l_\xi(a) = 0$ for all $a \in \dot{\mathcal{C}}$, and check whether their variables can be eliminated, in this way creating terminal rules that incorporate the epsilon rules and make them superfluous (similar to the construction known from the context-free case). To see how this can be done, let x be a variable in R, labeled with ξ. Then it is safe to delete x if $\xi \in X_0$ and the number of a-labeled nodes in L_ξ is at most $g_\xi(a)$, for all $a \in \dot{\mathcal{C}}$. If this is the case for a subset V of the variables in R, we add the rule r' to Γ, obtained by removing all variables in V from R. When all such rules have been added to the grammar, the (now superfluous) epsilon rules are removed.

Finally, let us sketch how to remove the chain rules from Γ, assuming that it does not contain epsilon rules. For this, the following observation is crucial. Consider a derivation $G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} \cdots \Rightarrow_{\mathcal{R}} G_m$ that consists of $m-1$ applications of chain rules followed by a single application of another rule. Suppose the variables replaced are $x_1, \ldots, x_m$, and let $1 \leqslant i_1 < \cdots < i_n = m$ be those

indices such that $x_m = x_{i_n}$ is a direct descendant of $x_{i_{n-1}}$, which is a direct descendent of $x_{i_{n-2}}$, and so on. Then all derivation steps that replace variables in $\{x_{i_1}, \dots, x_{i_n}\}$ can be postponed until after the other $m-n$ steps. This is because the chain rules do not create nodes that the other rules may use as contextual nodes. In other words, we can assume that $i_j = m - n + j$ for all $j \in [n]$. As a consequence, we can modify Γ as follows: we add all rules in $\mathcal{R}^{\bullet}_{\text{chain}} \bullet (\mathcal{R} \setminus \mathcal{R}_{\text{chain}})$ to $\mathcal{R}$ and remove all chain rules. Thanks to the observation above, the language generated stays the same. □

Note that, unfortunately, it seems that the normal forms of the previous two lemmas cannot be achieved simultaneously.

Definition 3 (Reducedness of Contexual Grammars). In a contexual grammar $\Gamma = \langle \mathcal{C}, \mathcal{R}, Z \rangle$, a rule $r \in \mathcal{R}$ is *useful* if there is a derivation of the form $Z \Rightarrow^*_{\mathcal{R}} G \Rightarrow_r G' \Rightarrow^*_{\mathcal{R}} H$ such that $H \in \mathcal{G}_{\mathcal{C} \setminus X}$. Γ is *reduced* if every rule in $\mathcal{R}$ is useful.

Note that, in the case of contextual grammars, usefulness of rules is not equivalent to every rule being reachable (i.e., for some G', the part of the derivation above up to G' exists) and productive (i.e., for some G, the part starting from G exists), because it is important that the pairs (G, G') are the same.

Theorem 1. *Reducedness is decidable for contextual grammars.*

Proof Sketch. Let us call a variable name ξ useful if there is a useful rule whose left-hand side variable has the name ξ. Clearly, it suffices to show that it can be decided which variable names are useful. To see this, note that we can decide reducedness by turning each derivation step into two, first a context-free step that nondeterministically "guesses" the rule to be applied and remembers the guess by relabeling the variable, and then a step using the guessed rule. Then the original rule is useful if and only if the new variable name recording the guess is useful.

Assume that the start graph is a single variable without attached nodes. Then, derivations can be represented as augmented derivation trees, where the vertices represent the rules applied. Suppose that some vertex ω represents the rule (L, R), where L contains the contextual nodes $u_1, \dots, u_k$. Then ω contains *contextual references* $(\omega_1, v_1), \dots, (\omega_k, v_k)$, where each ω_i is another vertex of the tree, and the v_i are distinct nodes, each of which is generated by the rule at ω_i and carries the same label as u_i. In other words, (ω_i, v_i) indicates that the contextual node u_i was matched to the node v_i generated at ω_i. Finally, in order to correspond to a valid derivation, there must be a linear order $\prec$ on the vertices of the derivation tree such that $\omega \prec \omega'$ for all children ω' of a vertex ω, and $\omega_i \prec \omega$ for each ω_i as above.[7]

Now, to keep the argument simple, assume that every rule contains at most one contextual node (see Lemma 1), and also that the label of this node differs

[7] To be precise, validity also requires that the variable replaced by the rule at ω is not attached to v_i.

from the labels of all nodes the variable is attached to. (The reader should easily be able to check that the proof generalizes to arbitrary contextual grammars.) The crucial observation is the following. Suppose that, for a given label $a \in \dot{\mathcal{C}}$, ω_a is the first vertex (with respect to $\prec$) that generates an a-labeled node v_a. Then, in each other vertex ω as above, if the rule contains an a-labeled contextual node u, the corresponding contextual reference (ω', v) can be replaced with (ω_a, v_a) without invalidating the derivation tree. We can do this for all vertices ω and node labels a. As a consequence, at most $|\dot{\mathcal{C}}|$ vertices of the derivation tree are targets of contextual references. Moreover, it should be obvious that, if the derivation tree is decomposed into $s(t(u))$, where the left-hand sides of the rules at the roots of t and u are the same, then $s(u)$ is a valid derivation tree, provided that no contextual references in s and u point to vertices in t. It follows that, to check whether a variable name is useful, we only have to check whether it occurs in the (finite) set of valid derivation trees such that

- all references to nodes with the same label are equal and
- for every decomposition of the form above, there is a contextual reference in s or u that points to a vertex in t. □

Clearly, removing all useless rules from a contextual grammar yields an equivalent reduced grammar. Thus, we can compute a reduced contextual grammar from an arbitrary one by determining the largest subset of rules such that the restriction to these rules yields a reduced contextual grammar.

Corollary 1. *Reduced contextual grammars are a normal from of contextual grammars.*

By turning a grammar into a reduced one, it can furthermore be decided whether the generated language is empty (as it is empty if and only if the set of rules is empty and the start graph contains at least one variable).

Corollary 2. *For a contextual grammar Γ, it is decidable whether $\mathcal{L}(\Gamma) = \emptyset$.*

4 Limitations of Contextual Grammars

Let us now come to two results that show limitations of contextual grammars similar to the known limitations of hyperedge-replacement grammars. The first of these results is a rather straightforward consequence of Lemma 2: as in the context-free case, the languages generated by contextual grammars are in NP, and there are NP-complete ones among them.

Theorem 2. *For every contextual grammar Γ, it holds that $\mathcal{L}(\Gamma) \in NP$. Moreover, there is a contextual grammar Γ such that $\mathcal{L}(\Gamma)$ is NP-complete.*

Proof. The second part follows from the fact that this holds even for hyperedge-replacement grammars, which are a special case of contextual grammars. For the first part, by Lemma 2, it may be assumed that Γ contains neither epsilon nor chain rules. It follows that the length of each derivation is linear in the size of the graph generated. Hence, derivations can be nondeterministically "guessed". □

It should be pointed out that the corresponding statement for hyperedge-replacement languages is actually slightly stronger than the one above, because, in this case, even the uniform membership problem is in NP (i.e., the input is (Γ, G) rather than just G). It is unclear whether a similar result can be achieved for contextual grammars, because the construction given in the proof of Lemma 2 may, in the worst case, lead to an exponential size increase of Γ.

Theorem 3. *For a graph G, let $|G|$ be either the number of nodes of G, the number of edges of G, or the sum of both. For every contextual grammar Γ, if $\mathcal{L}(\Gamma) = \{H_1, H_2, \dots\}$ with $|H_1| \leqslant |H_2| \leqslant \dots$, there is a constant k such that $|H_{i+1}| - |H_i| \leqslant k$ for all $i \in \mathbb{N}$.*

Proof Sketch. The argument is a rather standard pumping argument. Consider a contextual grammar Γ without epsilon and chain rules, such that $\mathcal{L}(\Gamma)$ is infinite. (The statement is trivial, otherwise.) Now, choose a derivation $Z = G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_n$ of a graph $G_n \in \mathcal{L}(\Gamma)$, and let x_i be the variable in G_i that is replaced in $G_i \Rightarrow G_{i+1}$, for $0 \leqslant i < n$. If the derivation is sufficiently long, there are $i < j$ such that x_i and x_j have the same label and x_j is a descendant of x_i (in the usual sense). Let $i = i_1 < \cdots < i_k = j$ be the indices l, $i \leqslant l \leqslant j$, such that x_l is a descendant of x_i. The steps in between those given by $i_1, \dots, i_k$ (which replace variables other than the descendants of x_i) may be necessary to create the contextual nodes that "enable" the rules applied to $x_{i_1}, \dots, x_{i_k-1}$. However, in G_j, these contextual nodes do all exist, because derivation steps do not delete nodes. This means that the sub-derivation given by the steps in which $x_{i_1}, \dots, x_{i_k-1}$ are replaced can be repeated, using x_j as the starting point (and using, in each of these steps the same contextual nodes as the original step). This pumping action can, of course, be repeated, and it increases the size of the generated graph by at most a constant each time. As there are neither epsilon nor chain rules, this constant is non-zero, which completes the proof. □

Corollary 3. *The language of all complete graphs is not in CHR.*

5 Parsing

In [13], a parser has been briefly sketched that can be used for contextual hyperedge replacement grammars with application conditions and, therefore, for contextual grammars. The following describes what grammar transformations are necessary before the parser can be applied, and the parser in more detail.

The parser adopts the idea of the *Cocke-Younger-Kasami* (CYK) parser for strings, and it requires the contextual grammar to be in *Chomsky normal form* (CNF), too. A contextual grammar is said to be in CNF if each rule is either terminal or nonterminal. The right-hand side of a terminal rule contains exactly one edge which is terminal, whereas the right-hand side of a nonterminal rule contains exactly two edges which are variables. Rules must not contain isolated nodes in their right-hand sides. In the following, we first outline that every contextual grammar Γ can be transformed into a grammar Γ' in CNF so that

a parser for Γ' can be used as a parser for Γ. We then consider a contextual grammar in CNF and sketch a CYK parser for such a grammar.

If the right-hand side of a rule contains an isolated node, it is either (i) a contextual node, or (ii) a node generated by the rule, or (iii) attached to the variable of the left-hand side. In case (i), we simply remove the node from the rule. However, the parser must make sure in its second phase (see below) that the obtained rule is only applied after a node with corresponding label has been created previously. Case (ii) can be avoided if we transform the original rule set $\mathcal{R}$ to $\mathcal{R}'$ where each node generated by a rule is attached to a unary hyperedge with a new label, say $\nu \in \bar{\mathcal{C}}$. Instead of parsing a graph G we have to parse a graph G' instead where each node is attached to such a ν-edge. Finally, case (iii) can be avoided by transforming $\mathcal{R}'$ again, obtaining $\mathcal{R}''$. The transformation process works iteratively: Assume a rule $L ::= R$ with R containing isolated nodes of kind (iii). Let $x \in \bar{L}$ with label ξ be the variable in L. This rule is replaced by a rule $L' ::= R'$ where L' and R' are obtained from L and R by removing the isolated nodes of kind (iii) and by attaching a new variable to the remaining nodes of $att(x)$, introducing a new variable name $\xi' \in X$. We now search for all rules that have ξ-variables in their right-hand sides. We copy these rules, replace all variables labeled ξ by ξ'-variables in their right-hand sides,[8] and add the obtained rules to the set of all rules. This process is repeated until no rule with isolated nodes is left. Obviously, this procedure terminates eventually. We assume that the start graph is a single variable labeled ζ, for some $\zeta \in X$ with $arity(\zeta) = \varepsilon$. Thus, no ζ-edge will ever be replaced by a ζ'-edge. It is clear that $Z \Rightarrow^*_{\mathcal{R}'} G$ iff $Z \Rightarrow^*_{\mathcal{R}''} G$ for each graph $G \in \mathcal{G}_{\mathcal{C}\setminus X}$.

Afterwards, chain rules are removed (see Lemma 2), and the obtained contextual grammar is transformed into an equivalent grammar in CNF using the same algorithm as for string grammars.[9] Based on this grammar, the parser analyzes a graph G in two phases. The first phase creates trees of rule applications bottom-up. The second phase searches for a derivation by trying to find a suitable linear order $\prec$ on the nodes of one of the derivation trees, as in the proof of Theorem 1.

In the first phase, the parser computes n sets $L_1, L_2, \ldots, L_n$ where n is the number of edges in G. Each set L_i eventually contains all graphs that contain exactly one variable and that can be derived to any subgraph of G that contains exactly i edges. Note that these subgraphs may contain contextual nodes, i.e., isolated nodes. Set L_1 is built by first finding each embedding of the right-hand side of each terminal rule and adding the isomorphic image of the corresponding *lhs*' to L_1. The remaining sets are then constructed using nonterminal rules. A nonterminal rule is reversely applied by searching for appropriate graphs s and s' in sets L_i and L_j, respectively. If a nonterminal rule is reversely applicable, i.e., if its right-hand side is isomorphic to the union of s and s' without any of

[8] This procedure assumes that no rule contains more than one ξ-edge in its right-hand side. It is easily generalized to rules with multiple occurrences of ξ-edges.

[9] This is possible iff the $\mathcal{L}(\Gamma)$ does not contain the empty graph which is easily accomplished since chain rules have been removed.

their isolated nodes,[10] a new graph s'' corresponding to the left-hand side of the rule is added to the set L_k. Note that $k = i+j$ since each graph in a set L_i can be derived to a subgraph of G with exactly i edges. Graph s'' additionally points to its child graphs s and s'. Therefore, each instance of the start graph Z in L_n represents the root of a tree of rule applications and, therefore, a derivation candidate for G.

The second parser phase tries to establish the linear order $\prec$. Contextual nodes are exactly the isolated nodes in graphs within the tree, and any graph generated previously in the tree must contain a newly generated node with an appropriate label. This process is similar to topological sorting, and it succeeds iff a derivation of G exists.

The run-time complexity of this parser highly depends on the grammar since the first phase computes all possible derivation trees. In bad situations, it is comparable to the exponential algorithm that simply tries all possible derivations. In "practical" cases without ambiguity (e.g., for control flow diagrams, cf. Example 2), however, the parser runs in polynomial time. Reasonably fast parsing has been demonstrated by DiaGen [14] that uses the same kind of parser.

Plump et al. have proposed *graph reduction grammars* [1]. The form of their rules $\mathcal{R}$ is not restricted; they may delete nodes, and need not have variables. Instead, they have to satisfy the following semantical condition: $\mathcal{R}$ is *reductive* if its *inverses* $\mathcal{R}^{-1}$ have terminating and confluent reductions $\Rightarrow^*_{\mathcal{R}^{-1}}$.

Then parsing of a graph G can be done by constructing an arbitrary reduction sequence $G \Rightarrow^*_{\mathcal{R}^{-1}} Y$ so that no rule of $\mathcal{R}^{-1}$ applies to Y. G is in the language of the grammar if and only if $Y = Z$ (up to isomorphism). No backtracking is needed in this case, and the complexity of (non-uniform) parsing is polynomial if the reduction sequence has polynomial length. (For a fixed set of rules, the complexity of a single step is always polynomial.)

This idea can be applied to contextual grammars as well. Monotonicity gives a simple criterion for termination of contextual rules (saying that every right-hand side of a rule contains at least one terminal edge or one new node). Then confluence of the rules' inverses can be done by checking that their *critical pairs* are strongly convergent [15]. So it can be verified mechanically whether a monotonic contextual grammar has reductive rules.

Example 3 (Parsing of Control Flow Diagrams). The rules of Example 2 are monotonic. The right-hand sides of the rules may overlap in their interface node. Overlap in interface nodes alone does not lead to a critical pair, because the rules are still parallelly independent. The right-hand sides of the recursive rules for assignment and branching may also overlap in the variables on their right-hand sides. This gives no critical pair either, because the inverse rules cannot be applied to the overlap: they violate the dangling condition. The rules are thus

[10] Furthermore, the parser must check whether the subgraphs of G being derivable from s and s' do not have edges in common. This is easily accomplished by associating each graph in any set L_i with the set of all edges in the derivable subgraph of G. A rule may be reversely applied to s and s' if the sets associated with s and s' are disjoint.

reductive, and their form reveals that reductions will have at most $\mathcal{O}(n)$ steps, where n is the number of terminal edges in the input graph G.

6 Conclusions

In this paper we have studied fundamental properties of contextual grammars. They have useful normal forms, namely rules with at most one contextual node, grammars without epsilon and chain rules, and reduced grammars. With context-free grammars, they share certain algorithmic properties (i.e., decidability of reducedness and emptiness, as well as an NP-complete membership problem) and the linear growth of their languages. Nevertheless, contextual grammars are more powerful than context-free ones, as illustrated in Figure 5. Let NR, ASR, cCHR, and cASR denote the classes of graph languages generated by node replacement, adaptive star replacement, conditional contextual hyperedge replacement, and conditional adaptive star grammars, respectively. HR is properly included in NR [8, Section 4.3], as is NR in ASR [3, Corollary 4.9]. The proper inclusion of HR in CHR is stated in Observation 1. Since the language $\mathcal{K}$ of complete graphs is in NR, Corollary 3 implies that CHR neither includes NR, nor ASR. It is still open whether ASR includes CHR. $\mathcal{K}$ is generated by a cCHR grammar so that this class includes CHR properly as well. The relations of cCHR to ASR and cASR are also still open. In Figure 5, $\mathcal{F}$ is the language of general flow diagrams of Example 2. Finally, $\mathcal{P}$ denotes the language of program graphs, which has a cCHR grammar [13] and an ASR grammar [6].

Some work related to the concepts shown in this paper shall be mentioned here. In [10, Chapter VIII], Annegret Habel discusses *context-sensitive hypergraph grammars*, which correspond to general graph grammars. The rules of these grammars do not only allow to connect to nodes in the context, but also to delete nodes and edges in it. This makes them more powerful: The languages of complete graphs can be defined with such grammars.

Shape analysis is about specification and verification of invariants for pointer structures in imperative programming languages, e.g., whether a data structure is a leaf-connected tree. Often, logical formalisms are used for this purpose [17]. The *graph reduction grammars* mentioned in Section 5 have been proposed for shape specification as well [1].

Future work on contextual grammars shall clarify the open questions concerning their generative power, and study rules with recursive application condi-

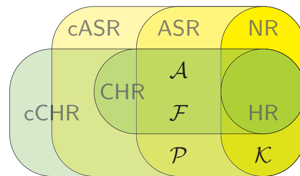

Fig. 5. Inclusion of languages studied in this paper and in [3,13]

tions [11]. Furthermore, we aim at an improved parsing algorithm for contextual grammars that are unambiguous modulo associativity and commutativity of certain replicative rules.

Acknowledgments. We wish to thank Annegret Habel for numerous useful comments on the contents of this paper.

References

1. A. Bakewell, D. Plump, and C. Runciman. Specifying pointer structures by graph reduction. *Mathematical Structures in Computer Science*, 2011. Accepted for publication. 12, 13
2. F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 2, pp. 95–162. World Scientific, Singapore, 1997. 3, 5
3. F. Drewes, B. Hoffmann, D. Janssens, and M. Minas. Adaptive star grammars and their languages. *Theoretical Computer Science*, 411:3090–3109, 2010. 2, 13
4. F. Drewes, B. Hoffmann, and M. Minas. Adaptive star grammars for graph models. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *4th Int. Conf. on Graph Transformation (ICGT'08)*, LNCS 5214, pp. 201–216. Springer, 2008. 2, 13
5. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs on Theoretical Computer Science. Springer, 2006. 3
6. J. Engelfriet. Context-free graph grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3: Beyond Words, chapter 3, pp. 125–213. Springer, 1999. 1, 13
7. A. Habel. *Hyperedge Replacement: Grammars and Languages*. LNCS 643. Springer, 1992. 3, 5, 13
8. A. Habel and H. Radke. Expressiveness of graph conditions with variables. *ECEASST*, 30, 2010. International Colloquium on Graph and Model Transformation (GraMoT'10). 2, 14
9. B. Hoffmann and M. Minas. Defining models – meta models versus graph grammars. *ECEASST*, 29, 2010. Proc. 6th Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'10), Paphos, Cyprus. 2, 3, 10, 13, 17, 18
10. M. Minas. Concepts and realization of a diagram editor generator based on hypergraph transformation. *Science of Comp. Progr.*, 44(2):157–180, 2002. 12
11. D. Plump. Hypergraph rewriting: Critical pairs and undecidability of confluence. In M. R. Sleep, R. Plasmeijer, and M. v. Eekelen, editors, *Term Graph Rewriting, Theory and Practice*, pp. 201–213. Wiley & Sons, Chichester, 1993. 12
12. M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM Transactions on Programming Languages and Systems*, 20(1):1–50, 1998. 13
13. A. Schürr, A. Winter, and A. Zündorf. The PROGRES approach: Language and environment. In G. Engels, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. II: Applications, Languages, and Tools*, chapter 13, pp. 487–550. World Scientific, Singapore, 1999. 2
14. T. Uesu. A system of graph grammars which generates all recursively enumerable sets of labelled graphs. *Tsukuba J. Math.*, 2:11–26, 1978. 1