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, using statecharts as an example. We extend context-free hyperedge-replacement—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.

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 hierarchical nesting) are limited; constraints (e.g., in the logic language OCL) have to be used for more complicated properties like connectedness.

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 must not be concealed that validating a given graph, by *parsing*, may be rather complex.

General graph grammars generate all recursively enumerable sets of graphs [16] so that there can be no parsing algorithm. Context-free graph grammars based on node replacement or hyperedge replacement [6] 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. We conclude that practically useful kinds of graph grammars should lie in between context-free and general ones. We take hyperedge replacement as a solid basis for devising such grammars, as it has

A. Schürr, D. Varró, and G. Varró (Eds.): AGTIVE 2011, LNCS 7233, pp. 182–197, 2012.

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* [2], devised with D. Janssens and N. Van Eetvelde, allows variables to be attached to arbitrary, unordered sets of nodes. Its generative power suffices to define sophisticated software models like program graphs [3]. Nevertheless, it inherits 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, which have been generated in earlier derivation steps [11]. 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 [11], 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 [15]), 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 decidability results, characterizations 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.

The remainder of this paper is structured as follows. In Section 2 we introduce contextual hyperedge replacement grammars and give some examples. In particular, we discuss a grammar for statecharts in Section 3. Normal forms for these grammars are presented in Section 4. In Section 5 we show some of their limitations w.r.t. language generation, and sketch parsing in Section 6. We conclude with some remarks on related and future work in Section 7.

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 $arity\colon \bar{\mathcal{C}} \uplus X \to \dot{\mathcal{C}}^*$.[1]

A *labeled hypergraph over* $\mathcal{C}$ (*graph*, for short) $G = \langle \dot{G}, \bar{G}, att_G, \dot{\ell}_G, \bar{\ell}_G \rangle$ consists of disjoint finite sets $\dot{G}$ of *nodes* and $\bar{G}$ of *hyperedges* (*edges*, for short) respectively, a function $att_G\colon \bar{G} \to \dot{G}^*$ that *attaches* sequences of pairwise distinct nodes to edges so that $\dot{\ell}^*_G(att_G(e)) = arity(\bar{\ell}_G(e))$ for every edge $e \in \bar{G}$,[2] and *labeling* 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:

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

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 [5].

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 not attached to x 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 [7].)

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 [11], 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.

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

[2] 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 $a_i \in A$, $1 \leqslant i \leqslant n$, $n \geqslant 0$.

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 $Z \in \mathcal{G}_{\mathcal{C}}$.

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 counterclockwise 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 "$\mathsf{n}|\mathsf{m}\cdots$" below the symbol $::=$ define names that are used to refer to rules in derivations, as in Figure 1 and Figure 3.

Example 1 (The Language of All Graphs). The contextual grammar in Figure 1 generates the set $\mathcal{A}$ of loop-free labeled graphs with binary edges, and Figure 2 shows a derivation with this grammar. Rules $\mathsf{0}$ and d generate $n \geqslant 0$ variables labeled with $\mathbf{G}$; the rules n_x generate a node labeled with x, and the rules e_a insert an edge labeled with a between two nodes that are required to exist in the context.

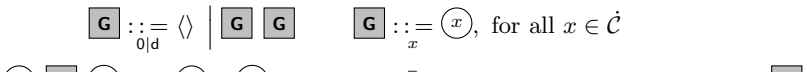

Fig. 1. A contextual grammar (generating the language of all graphs)

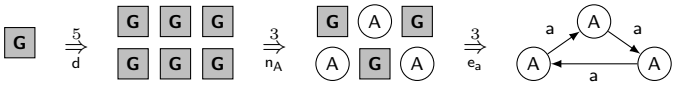

Fig. 2. A derivation with the rules in Figure 1

It is well known that the language $\mathcal{A}$ cannot be defined by hyperedge-replacement grammars [7, Chapter IV, Theorem 3.12(1)].[3] Thus, as CHR contains HR by definition, we have:

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

Flow diagrams are another example for this observation: In contrast to structured and semi-structured control flow diagrams, unrestricted control flow diagrams are not in HR, because they have unbounded tree-width [7, Chapter IV, Theorem 3.12(7)]. However, they can be generated by contextual grammars.

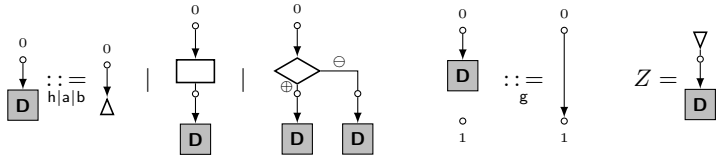

Fig. 3. Rules generating unrestricted control flow diagrams

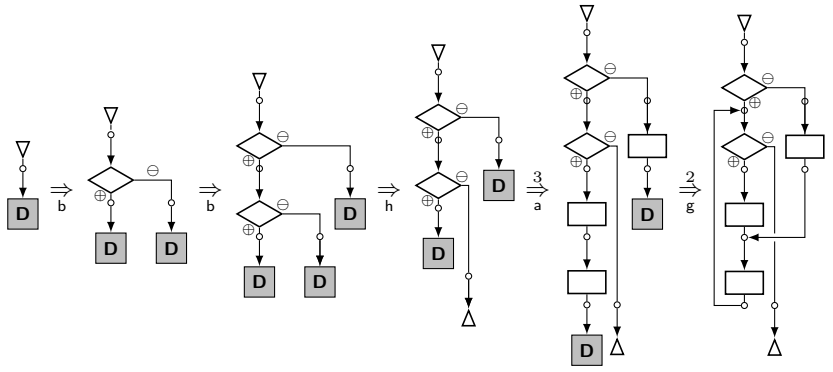

Fig. 4. A derivation of an unstructured control flow diagram

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

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

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.

Note that flow diagrams cannot be defined with class diagrams, because subtyping and multiplicities do not suffice to define rootedness and connectedness of graphs.

[3] It is well-known that node replacement (more precisely, confluent edNCE graph grammars) cannot generate $\mathcal{A}$ either [6, Thm. 4.17]. Hence, Observation 1 holds similarly for node replacement.

3 A Contextual Grammar for Statecharts

Statecharts [9] are an extension of finite automata modeling the state transitions of a system in reaction on events. The statechart in Figure 5 describes an auction. Blobs denote states, and arrows denote transitions between them. Black blobs are initial states, blobs with a black kernel are final states, and the others are inner states. Inner states may be compound, containing compartments, separated with dashed lines, which contain sub-statecharts that act independently from one another, and may themselves contain compound states, in a nested fashion. Text inside a blob names the purpose of a state or of its compartments, and labels at transitions name the event triggering them. (We consider neither more general event specifications, nor more special types of states.)

The structure of statecharts can be specified by the class diagram shown in Figure 6. The dotted circle with a gray kernel is abstract, for inner or stop states. An inner state may be composed of compartments (denoted as dashed blobs), which in turn are composed of other states (defining the sub-charts). In examples like Figure 5, a compound state is drawn as a big blob with solid lines wherein the compartments are separated by dashed lines.

The class diagram captures several structural properties of statecharts: It forbids isolated initial and final states and transitions to initial states; each compartment contains exactly one initial state, and compound states and their compartments form a tree hierarchy as the associations uniting and containing are compositions (indicated by the black diamonds at their sources).

Example 3 (A Grammar for Statecharts). The contextual rules in Figure 7 generate statecharts according to the class diagram in Figure 6. (Let us ignore the parts underlaid in gray for a moment.) The charts in these rules are drawn so that the compositions uniting and containing are just represented by drawing the blob of their target within the blob of their source. We assume (for regularity) that the topmost statechart is contained in a compartment, see the start graph Z. The rules for **S** add transitions to current states, which are either initial or inner states (drawn as gray nodes). The target of the transition is either a new final state (rule f), or a new inner state, to which further transitions may be

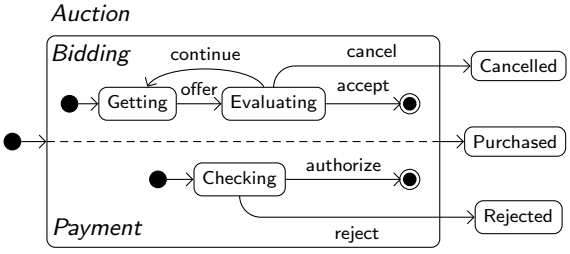

Fig. 5. A statechart modeling an auction

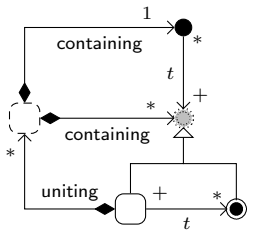

Fig. 6. A class diagram for statecharts

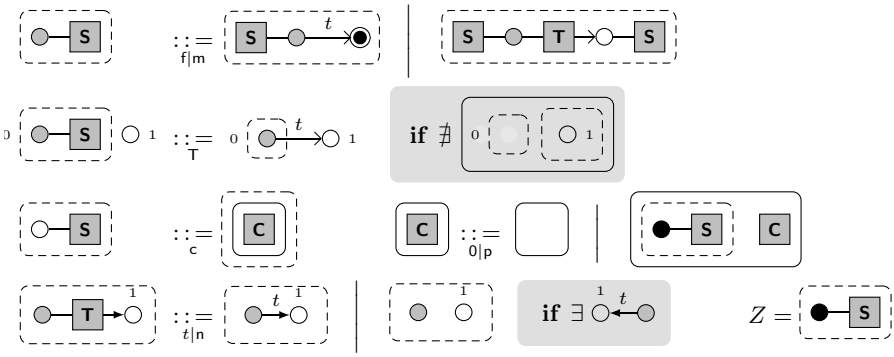

Fig. 7. Contextual rules for statecharts (with application conditions)

added (rule m), or an inner state that exists in the context, but not necessarily in the same compartment (rule T). Rule m inserts a variable named **T** that may generate a concrete transition (rule t), or leave the states separate (rule n). (This is necessary since the transitions to an inner state in a compartment need not come from a state in that compartment, like states Canceled and Rejected in Figure 5.) Finally, inner states may be filled (by rules 0 and p for the variable **C**) with compartments, each containing a statechart as in the start graph Z.

Every state in a chart should be reachable from an initial state. Reachability cannot be expressed by class diagrams alone. In order to specify this property in a meta-model, the inner state must be extended with an auxiliary attribute that determines this condition by inspecting its direct predecessors, and with a logical constraint, e.g., in OCL, which requires that the value of the attribute is true for every instance of a state.

Example 2 shows that contextual grammars can express reachability as such. In statecharts, reachability is combined with hierarchical nesting of sub-states, and cannot be specified with contextual rules. However, we may extend contextual rules with application conditions, as proposed in [11]. The parts underlaid in gray add application conditions to two rules of Figure 7. In Rule n, the condition requires that the target node of variable **T** is the target of another transition. It is easy to show that this guarantees reachability from initial states. The condition for rule T expresses yet another consistency requirement: The source and target of a transition must not lie in sister compartments of the same compound state.

4 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.

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

Proof. This is straightforward. Suppose we wish to implement a rule (L, R) whose left-hand side contains a variable with k attached nodes and $l \geqslant 1$ contextual nodes. We use l rules to collect the l contextual nodes one by one, finally ending up with a variable that is attached to $k + l$ nodes. The original rule is then turned into a context-free rule. □

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 those contextual grammars that do not generate the empty graph.*

Proof Sketch. While the overall structure of the proof is similar to the corresponding proof for the context-free case, its details are slightly more complicated. Therefore, we give only a very rough sketch of the proof. The full proof will be given in a forthcoming extended version of this article.

The proof works as follows. First, it is shown that epsilon rules may be removed by composing non-epsilon rules with epsilon rules that remove some of the variables in the right-hand side of the original rule. Afterwards, chain rules are removed by replacing them with rules that correspond to a sequence of chain rules applied in succession, followed by the application of a non-chain rule.

The notion of composition used here has to take contextual nodes into account. Suppose we are given 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 need to be able to combine both rules even if R_1 does not supply r_2 with all the necessary contextual nodes. We do this by enriching L_1 with the contextual nodes needed by r_2. However, if r_1 contains nodes (with the right labels) that are isolated in both L_1 and R_1, these are used instead rather than adding even more contextual nodes to the left-hand sides. This is a necessary precaution, because the composition of chain rules may otherwise create an infinite set of rules.

The removal of epsilon rules is non-trivial, because we have to make sure to avoid introducing deadlocks. To see this, suppose a rule r_1 creates a variable e_1 that can be removed by an epsilon rule containing a contextual node labeled a_1. Similarly, assume that r_2 creates a variable e_2 that can be removed by an epsilon rule containing a contextual node labeled a_2. Assume furthermore that r_1 generates an a_2-labeled node and r_2 generates an a_1-labeled node. Then, given a graph that contains the left-hand sides of r_1 and r_2, we can apply r_1 and r_2, followed by the epsilon rules that delete e_1 and e_2. However, if we compose r_1 with the first epsilon rule and r_2 with the second one, neither of the composed rules may be applicable, because the first contains an a_1-labeled contextual node and the second contains an a_2-labeled contextual node. Fortunately, the problem can be solved by a guess-and-verify strategy, thanks to the fact that the number of contextual nodes in the left-hand sides of rule is bounded. Roughly speaking, the guess-and-verify strategy makes sure that the required contextual nodes will be generated somewhere in the graph.

Finally, let us sketch how to remove chain rules, assuming that the grammar 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 $\{x_{i_1}, \ldots, x_{i_n}\}$ 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, it is safe to modify Γ by adding all rules obtained by composing a sequence of chain rules with a single non-chain rule 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 Contextual Grammars). In $\Gamma = \langle \mathcal{C}, \mathcal{R}, Z \rangle$, 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 $u_1, \ldots, u_k$. Then the augmentation of ω consists in *contextual references* $(\omega_1, v_1), \ldots, (\omega_k, v_k)$, 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. The pair (ω_i, v_i) means 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.[4]

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 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). This may affect the graph generated, but does not invalidate 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 (a) all references to nodes with the same label are equal and (b) 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.*

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

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 $\mathcal{L}(\Gamma) = \emptyset$.*

5 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

$x_{i_1}, \ldots, x_{i_k-1}$ 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.*

6 Parsing

In [11], 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 the parser in more detail, including the grammar transformations that are necessary before it can be applied.

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,[5] 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.

[5] 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.

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.[6] 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 $S_1, S_2, \ldots, S_n$ where n is the number of edges in G. Each set S_i eventually contains all graphs (called "S_i-graphs" in the following) that are isomorphic to the left-hand side of any rule, except for their contextual nodes which are left out, and that can be derived to any subgraph of G that contains exactly i edges, if any required contextual nodes are provided.

Set S_1 is built by finding each occurrence s of the right-hand side R of any terminal rule (L, R) and adding the isomorphic image s' of L to S_1, but leaving out all of its contextual nodes. Graph s' additionally points to its "child" graph s.

The remaining sets $S_i, i > 1$, are then constructed using nonterminal rules. A nonterminal rule (L, R) is reversely applied by selecting appropriate graphs s and s' in sets S_i and S_j, respectively, such that $R \cong s \cup s'$. A new graph s'' is then[7] added to the set S_k where s'' is isomorphic to L without its contextual nodes. Note that $k = i + j$ since each S_i-graph 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 S_n represents the root of a tree of rule applications and, therefore, a derivation candidate for G. Note that contextual nodes are not explicitly indicated in these trees because they have been removed from the S_i-graphs. Contextual nodes are rather treated as if they were generated by the rules. However, they can be easily distinguished from really generated ones by inspecting the rules used for creating the S_i-graphs.

The second parser phase tries to establish the linear order $\prec$ on the nodes of the derivation tree. The order must reflect the fact that each contextual node must have been generated earlier in the derivation. 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.

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

[7] 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 S_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.

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 [12] that uses the same kind of parser.

A simpler, more efficient way of parsing can be chosen for grammars with the following property: A contextual grammar $\Gamma = (\mathcal{C}, \mathcal{R}, Z)$ is *uniquely reductive* if its derivation relation $\Rightarrow_{\mathcal{R}}$ has an inverse relation $\Rightarrow_{\mathcal{R}^{-1}}$ (called *reduction relation*) that is is terminating and confluent. Then every graph has a reduction sequence $G \Rightarrow^*_{\mathcal{R}^{-1}} Y$ so that no rule of $\mathcal{R}^{-1}$ applies to Y. Confluence of reduction implies that the graph Y is unique up to isomorphism so that G is in the language of Γ if and only if Y equals Z up to isomorphism.

Let Γ be a contextual grammar with neither epsilon, nor chain rules (By Lemma 2, each contextual grammar without epsilon rules can be transformed into such a normal form). Then every right-hand side of a rule contains at least one terminal edge or one new node, and reductions $G \Rightarrow^*_{\mathcal{R}^{-1}} Y$ terminate, after a linear number of steps. Confluence of terminating reductions $\Rightarrow_{\mathcal{R}^{-1}}$ can be shown by checking that their *critical pairs* are strongly convergent [13]. So it can be decided whether Γ is uniquely reductive.

Since the construction of a single reduction step is polynomial for a fixed set of rules, the complexity of parsing is polynomial as well. Note, however, that parsing does not yield unique derivation structures if the reduction relation has critical pairs.

Example 4 (Parsing of Control Flow Diagrams). The grammar in Example 2 does not contain epsilon or chain rules. 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 variables. 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 uniquely reductive.

7 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 8. 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 [6, Section 4.3], as is NR in ASR [2, Corollary 4.9]. The proper inclusion of HR in CHR is stated in Observation 1. Corollary 3 implies that CHR neither

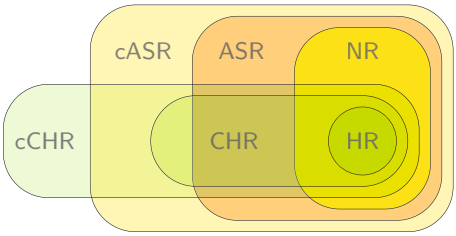

Fig. 8. Inclusion of languages studied in this paper and in [2,11]

includes NR, nor ASR, nor cCHR, because these grammars generate the language of complete graphs. We do not yet know whether ASR includes CHR; the relation of cCHR to ASR and cASR is open as well. Example 2 indicates that contextual grammars allow for a finer definition of structural properties of models than class diagrams. Application conditions do further increase this power, as discussed in Section 3.

Some work related to the concepts shown in this paper shall be mentioned here. Context-exploiting rules [4] correspond to contextual rules with a positive application condition, and are equivalent to the context-embedding rules used to define diagram languages in DiaGen [12]. The context-sensitive hypergraph grammars discussed in [7, Chapter VIII] correspond to context-free rules with a positive application condition. We are not aware of any attempts to extend node replacement in order to define graph languages as they are discussed in this paper. The graph reduction specifications [1] mentioned in Section 6 need not use nonterminals, and their rules may delete previously generated subgraphs. They are therefore difficult to compare with contextual grammars. Example 4 shows that some contextual rules specify graph reductions, and may thus use their simple parsing algorithm. *Shape analysis* aims at specifying pointer structures in imperative programming languages (e.g., leaf-connected trees), and at verifying whether this shape is preserved by operations. Several logical formalisms have been proposed for this purpose [14]. For graph transformation rules, shape analysis has been studied for shapes defined by context-free grammars [10] and by adaptive star grammars [3]. We are currently working on shape analysis of graph transformation rules w.r.t. contextual grammars.

Future work on contextual grammars shall clarify the open questions concerning their generative power, and continue the study of contextual rules with recursive application conditions [8] that has been started in [11]. Furthermore, we aim at an improved parsing algorithm for contextual grammars that are unambiguous modulo associativity and commutativity of certain replicative rules.

Acknowledgements. We wish to thank Annegret Habel for numerous useful comments on the contents of this paper, and the reviewers for their advice to enhance the "smack of industrial relevance" of this paper.

References

1. Bakewell, A., Plump, D., Runciman, C.: Specifying pointer structures by graph reduction. Mathematical Structures in Computer Science (2011) (accepted for publication)
2. Drewes, F., Hoffmann, B., Janssens, D., Minas, M.: Adaptive star grammars and their languages. Theoretical Computer Science 411(34-36), 3090–3109 (2010)
3. Drewes, F., Hoffmann, B., Janssens, D., Minas, M., Van Eetvelde, N.: Shaped Generic Graph Transformation. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) AGTIVE 2007. LNCS, vol. 5088, pp. 201–216. Springer, Heidelberg (2008)
4. Drewes, F., Hoffmann, B., Minas, M.: Context-exploiting shapes for diagram transformation. Machine Graphics and Vision 12(1), 117–132 (2003)
5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs on Theoretical Computer Science. Springer (2006)
6. Engelfriet, J.: Context-Free Graph Grammars. In: Handbook of Formal Languages. Beyond Words, vol. 3, ch. 3, pp. 125–213. Springer (1999)
7. Habel, A.: Hyperedge Replacement: Grammars and Languages. LNCS, vol. 643. Springer, Heidelberg (1992)
8. Habel, A., Radke, H.: Expressiveness of graph conditions with variables. In: Ermel, C., Ehrig, H., Orejas, F., Taentzer, G. (eds.) International Colloquium on Graph and Model Transformation 2010. ECEASST, vol. 30 (2010)
9. Harel, D.: On visual formalisms. Communication of the ACM 31(5), 514–530 (1988)
10. Hoffmann, B.: Shapely hierarchical graph transformation. In: Proc. of the IEEE Symposia. on Human-Centric Computing Languages and Environments, pp. 30–37. IEEE Computer Press (2001)
11. Hoffmann, B., Minas, M.: Defining models – Meta models versus graph grammars. In: Küster, J.M., Tuosto, E. (eds.) Graph Transformation and Visual Modeling Techniques 2010. ECEASST, vol. 29 (2010)
12. Minas, M.: Concepts and realization of a diagram editor generator based on hypergraph transformation. Science of Computer Programming 44(2), 157–180 (2002)
13. Plump, D.: Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In: Sleep, M.R., Plasmeijer, M.J., van Eekelen, M.C. (eds.) Term Graph Rewriting, Theory and Practice, pp. 201–213. Wiley & Sons (1993)
14. Sagiv, M., Reps, T., Wilhelm, R.: Solving shape-analysis problems in languages with destructive updating. ACM Transactions on Programming Languages and Systems 20(1), 1–50 (1998)
15. Schürr, A., Winter, A., Zündorf, A.: The Progres Approach: Language and Environment. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rosenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation. Applications, Languages, and Tools, vol. 2, ch. 13, pp. 487–550. World Scientific (1999)
16. Uesu, T.: A system of graph grammars which generates all recursively enumerable sets of labelled graphs. Tsukuba Journal of Mathematics 2, 11–26 (1978)