Acta Informatica (2015) 52:497–524
DOI 10.1007/s00236-015-0223-4

ORIGINAL ARTICLE

Contextual hyperedge replacement

Frank Drewes · **Berthold Hoffmann**

Received: 4 February 2014 / Accepted: 3 February 2015 / Published online: 29 March 2015

Abstract Contextual hyperedge-replacement grammars (contextual grammars, for short) are an extension of hyperedge replacement grammars. They have recently been proposed as a grammatical method for capturing the structure of object-oriented programs, thus serving as an alternative to the use of meta-models like UML class diagrams in model-driven software design. In this paper, we study the properties of contextual grammars. Even though these grammars are not context-free, one can show that they inherit several of the nice properties of hyperedge replacement grammars. In particular, they possess useful normal forms and their membership problem is in NP.

Keywords Graph grammar · Hyperedge replacement · Context · Contextual grammar

Mathematics Subject Classification F.4.3 [formal languages] Classes defined by grammars—contextual graph grammars

1 Introduction

Graphs are ubiquitous in science and beyond, since they provide a mathematically sound basis for the study of all types of structural models that consist of entities and relationships between them. Furthermore, they can nicely be visualized as diagrams, with entities depicted as nodes, and relationships drawn as lines.

Let us just mention two fields in computer science where graphs are used:

- In the study of algorithms, graphs abstract from data structures with pointers as they occur in programming [16].

F. Drewes (✉)
Institutionen för datavetenskap, Umeå universitet, 901 87 Umeå, Sweden
e-mail: drewes@cs.umu.se

B. Hoffmann (✉)
FB 3 – Informatik, Universität Bremen, 28334 Bremen, Germany
e-mail: hof@informatik.uni-bremen.de

- In software engineering, software is nowadays usually described by structural models, which are often visualized as diagrams, e.g., of the Unified Modeling Language [24].

When studying graphs, one is often interested in graphs that have a certain structural property P, e.g, cyclicity, connectedness, or the existence of a unique root. This can be used for several purposes, e.g., to

- check whether a particular graph has the property P,
- define P by specifying the class of all graphs having this property,
- restrict attention to the class of graphs with this property,
- exploit P in order to derive algorithms for graphs having this property, and
- investigate whether transformations on these graphs preserve P.

Therefore, it is an important problem to devise algorithmically feasible methods that make it possible to specify sets of graphs having a certain desired property. Well-known specification mechanisms include logic (in particular monadic second-order logic [4]), meta-models (e.g., UML class diagrams), and graph grammars. While general graph grammars are Turing complete and thus too powerful to be handled algorithmically, context-free graph grammars based on node or hyperedge replacement have nice algorithmic properties. In contrast to meta-models, context-free graph grammars are generative devices. They derive sets of graphs constructively, by iteratively applying rules, beginning with a start graph. This kind of definition is strict, can easily produce sample graphs by derivation, and provides the generated graphs with a recursive structure that has many useful algorithmic applications. However, it must not be concealed that the membership problem, that is validating a given graph by parsing, is rather complex in general, namely NP-complete.

Unfortunately, context-free graph grammars are slightly too weak for modeling the structure of software. For instance, they cannot generate the classes of all acyclic graphs or of all connected graphs—not even the very elementary class of arbitrary graphs over a certain set of labels can be generated. Therefore, extensions such as adaptive star grammars [6,7,10,18] and contextual hyperedge replacement grammars [11,20] have been proposed. In the current paper, we study contextual hyperedge grammars mainly from a theoretical point of view, investigating their grammatical and algorithmic properties. Despite the fact that contextual grammars are not context-free, it turns out that they share some of the most important properties of hyperedge replacement grammars. In particular, we show that

- empty rules, chain rules, and useless rules can effectively be removed from contextual grammars without affecting the generated language, and
- as a consequence, the languages generated by contextual grammars belong to NP, and
- the Parikh images of their generated languages are semi-linear.

These results are based on a central normal-form proved in this paper: contextual grammars can be modified in such a way that, roughly speaking, the eventual applicability of a rule to a hyperedge depends only on the label in its left-hand side. If this label matches the label of the hyperedge to be replaced, the rule will eventually apply (except if the derivation does not terminate).

The remainder of this paper is structured as follows. In Sect. 2 we recall contextual grammars from [11] and give some examples. In particular, we discuss a grammar for program graphs. Normal forms for these grammars are proved in Sect. 3. In Sect. 4 we show some of their limitations w.r.t. language generation. We conclude with some remarks on related and future work in Sect. 5. This article extends the paper [11] considerably, by giving more (and more practical) examples, providing fully detailed proofs for the results in Sect. 3, and by introducing the context-safe form of contextual grammars.

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.

A^* denotes the set of all finite sequences over a set A; the empty sequence is denoted by ε. Given a sequence u, we denote by $[u]$ the smallest set A such that $[u] \in A^*$. 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 $a_i \in A$, $1 \leqslant i \leqslant n$, $n \geqslant 0$. If $g\colon B \to C$ is another function, the composition of f and g is denoted by $g \circ f$.

We consider labeling alphabets $\mathscr{C} = \dot{\mathscr{C}} \uplus \bar{\mathscr{C}} \uplus X$ that are sets whose elements are the *labels* (or "*colors*") of nodes, edges, and variables, and come with a *type* function $type\colon (\bar{\mathscr{C}} \uplus X) \to \dot{\mathscr{C}}^*$.

A *labeled hypergraph* $G = \langle \dot{G}, \bar{G}, att_G, \dot{\ell}_G, \bar{\ell}_G \rangle$ *over* $\mathscr{C}$ (a *graph*, for short) 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, and *labeling* functions $\dot{\ell}_G\colon \dot{G} \to \dot{\mathscr{C}}$ and $\bar{\ell}_G\colon \bar{G} \to \bar{\mathscr{C}} \uplus X$ so that $\dot{\ell}^*_G(att_G(e)) = type(\bar{\ell}_G(e))$ for every edge $e \in \bar{G}$. Edges are called *variables* if they carry a variable name as a label; the set of all graphs over $\mathscr{C}$ is denoted by $\mathscr{G}_{\mathscr{C}}$.

For a graph G and a set $E \subseteq \bar{G}$ of edges, we denote by $G - E$ the graph obtained by removing all edges in E from G. If E is a singleton $\{e\}$, we may write $G - e$ instead of $G - \{e\}$.

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:

$$\dot{\ell}_H \circ \dot{m} = \dot{\ell}_G,\ \bar{\ell}_H \circ \bar{m} = \bar{\ell}_G, \quad \text{and} \quad att_H \circ \bar{m} = \dot{m}^* \circ att_G.$$

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

Notation (*Drawing Conventions for Graphs*) Graphs are drawn as in Figs. 1 and 3 below. Circles and boxes represent nodes and edges, respectively. The text inside is their label from $\mathscr{C}$. If all nodes carry the same label, it is 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. 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. The empty graph is denoted as $\langle\rangle$.

Example 2.1 (*Program graphs*) In a program graph, syntactic entities – classes, variables, signatures and bodies of methods, expressions—are represented as nodes labeled with the first letter of the entity's kind. Edges establish relations between entities. Those drawn as straight arrows "$\rightarrow$" denote "*has-a*" relations (called *compositions* in UML terminology), and represent the abstract syntax of the program: classes consist of subclasses, and of declarations of features: variables, method signatures, and method bodies; method signatures have formal parameters (represented as variables), method bodies consist of expressions; expressions may have other expressions as subexpressions. Edges drawn as winding lines "$\rightsquigarrow$" relate every use of an entity with its declaration: the body of a method with the signature it implements, an expression with a variable that is used, or updated with the value of an argument expression, or with the signature of a method that is called with argument expressions.

```
class Cell is
  var cts: Any;
  method get() Any is
    return cts;
  method set(var n: Any) is
    cts := n

subclass ReCell of Cell is
  var backup: Any;
  method restore() is
    cts := backup;
  override set(var n: Any) is
    backup := cts;
    super.set(n)
```

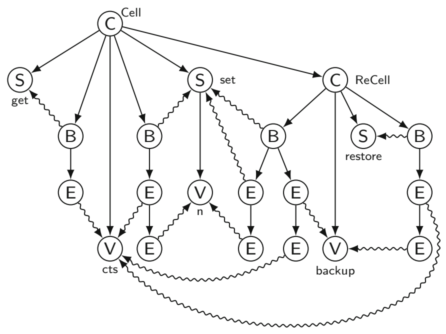


Fig. 1 An object-oriented program and its program graph

Figure 1 shows a simple object-oriented program from [1] and its representation as a program graph. Every program entity is represented by a unique node; the names of classes, variables, methods and parameters are irrelevant in the graph. (The names ascribed to the nodes in the graph shall just clarify the correspondence to the text.)

The program graphs introduced here are simplified w.r.t. the comprehensive definition in [26]: control flow of method bodies, types (of variables, parameters, and methods), and visibility rules for declarations have been omitted.

Not every graph over the labels appearing in Fig. 1 is a valid program graph. We shall define the class of valid program graphs by a contextual grammar, in Example 2.7. See [20] for a thorougher discussion of how program graphs can be defined by meta-models, with a UML class diagram and logical OCL constraints. □

The replacement of variables in graphs by graphs is performed by applying a special form of standard double-pushout rules [12].

Definition 2.2 (*Contextual Rule*) A *contextual rule* (*rule*, for short) $r = (L, R)$ consists of graphs L and R over $\mathscr{C}$ 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.

Let r be a contextual rule as above, and consider some graph G. An injective morphism $m : L \to G$ is called a *matching* for r in G. If such a matching exists, we say that r is *applicable* to the variable $m(x) \in \bar{G}$. The *replacement* of $m(x)$ by R (via m) is then given as 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 $H = G[R/m]$.

Context-free rules are known as hyperedge replacement rules in the graph grammar literature [14]. Note that contextual rules are equivalent to contextual star rules as introduced in [20], however without application conditions.

To simplify some constructions, we will make the following assumption throughout the paper.

Assumption In an application of a contextual rule $r = (L, R)$ as in Definition 2.2, the replacement of $m(x)$ by R in $G[R/m]$ is always made in such a way that fresh copies of the

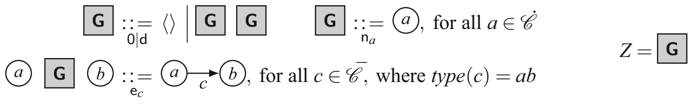

G ::= ⟨⟩ | G G
0|d
G ::= a, for all $a \in \dot{\mathscr{C}}$
n_a
a G b ::= a → b, for all $c \in \bar{\mathscr{C}}$, where $type(c) = ab$
e_c
c
Z = G

Fig. 2 A contextual grammar (generating the language of all graphs)

nodes and edges in R that are not in L are added to $G - m(x)$. Thus, the nodes and edges in G are still present in $G[R/m]$, except for the variable that has been replaced, and all nodes and edges that have been added are fresh copies of the corresponding ones in R.

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.

Definition 2.3 (*Contextual Grammar*) A *contextual hyperedge-replacement grammar* (*contextual grammar*, for short) is a triple $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$ consisting of a finite labeling alphabet $\mathscr{C}$, a finite set $\mathscr{R}$ of contextual rules, and a start graph $Z \in \mathscr{G}_{\mathscr{C}}$.

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

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

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

We note here that, by adding a new variable label S of type ε, every grammar can be turned into one in which the start graph consists of a single variable and no nodes. This easy modification is sometimes useful in formal constructions.

For individual contextual rules r, we abbreviate $G \Rightarrow_{\{r\}} H$ by $G \Rightarrow_r H$. Note that the assumption above extends to derivations: given any derivation $G \Rightarrow^*_{\mathscr{R}} H$, the nodes and edges in G are still present in H, except for variables that have been replaced, and all nodes and edges that have been added in the course of the derivation are fresh copies of those in the corresponding right-hand sides.

Notation (*Drawing Conventions for Rules*) A contextual rule $r = (L, R)$ is denoted as $L ::= R$, see, e.g., Figs. 2 and 4. Small numbers above nodes indicate identities of nodes in L and R. The notation $L ::= R_1 | R_2, \ldots$ is used as a shorthand 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 by which we can refer to rules in derivations.

Example 2.4 (*The language of all graphs*) The contextual grammar in Fig. 2 generates the set of all loop-free graphs with binary edges over a labeling alphabet $\mathscr{C}$, and Fig. 3 shows a derivation with this grammar. Rules $\mathsf{0}$ and d generate $n \geqslant 0$ variables labeled with $\mathbf{G}$; for every node label a, the rule n_a generates a node labeled with a; similarly, for an edge label c, the rule e_c inserts an edge labeled with c between two nodes that are required to exist in the context. □

It is well known that the language of Example 2.4 cannot be generated by hyperedge replacement [14, Chapter IV, Theorem 3.12(1)]. The same holds for context-free node replacement, i.e., C-edNCE grammars; see [13, Theorem 4.17]. Thus, as CHR contains HR by definition, we have:

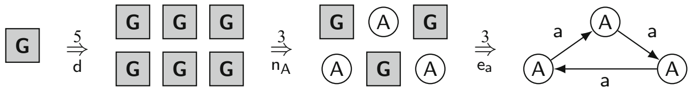


Fig. 3 A derivation with the rules in Fig. 2

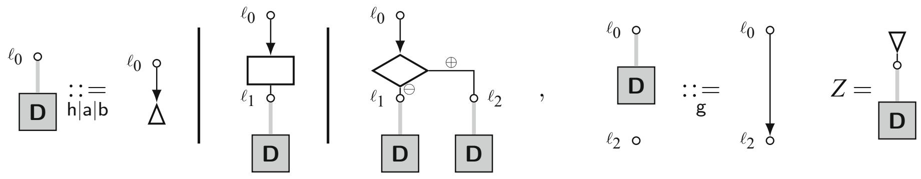


Fig. 4 Rules generating unrestricted control flow diagrams

Observation 2.5 HR $\subsetneq$ CHR and CHR $\not\subseteq$ C-edNCE.

Flow diagrams are another example for the strict inclusion of HR in CHR: In contrast to structured and semi-structured control flow diagrams, unrestricted control flow diagrams are not in HR, because they have unbounded tree-width [14, Chapter IV, Theorem 3.12(7)]. However, they can be generated by contextual grammars.

Example 2.6 (*Control flow diagrams*) Unrestricted control flow diagrams represent sequences of low-level instructions according to a textual syntax like this:

$$\begin{aligned}&\mathsf{Program} ::= \varepsilon \mid \ell\colon \mathsf{Instruction};\ \mathsf{Program}\\ &\mathsf{Instruction} ::= \mathbf{halt} \mid \mathsf{Assignment} \mid \mathbf{if}\ \mathsf{Condition}\ \mathbf{then\ goto}\ \ell \mid \mathbf{goto}\ \ell\end{aligned}$$

The rules in Fig. 4 generate unrestricted flow diagrams, wherein we omit the text inside the rectangles and diamonds representing assignments and conditions, respectively. The context-free rules h, a, and b generate control flow trees of the halt, assignment, and conditional jump, respectively, and the fourth rule, g, which is not context-free, inserts gotos to a program location in the context. In Fig. 5, these rules are used to derive a flow diagram that is not structured. □

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

Example 2.7 (*A contextual grammar for program graphs*) The rules in Fig. 6 define a contextual grammar PG = ($\mathscr{C}$, P, Z) for program graphs, where the start graph Z is the left-hand side of the first rule, hy.

Figure 7 shows snapshots in a derivation that could be completed to derive the program graph in Fig. 1. The second graph is obtained in nine steps that apply the rules hy once, cl* five times, cl^0 once, hy* once, and hy^0 once. The third graph is obtained by an application of rule at. The fourth graph is obtained in five steps that apply rules si twice, pa* once, and pa^0 twice. The fifth and last graph is obtained by two applications of rule im. □

3 Normal forms of contextual grammars

In this section, we study normal form properties of contextual grammars.

Recall that, for any class C of devices (such as grammars or automata), a subclass C' of C is said to be a normal form of C if every element of C can be turned into an element of

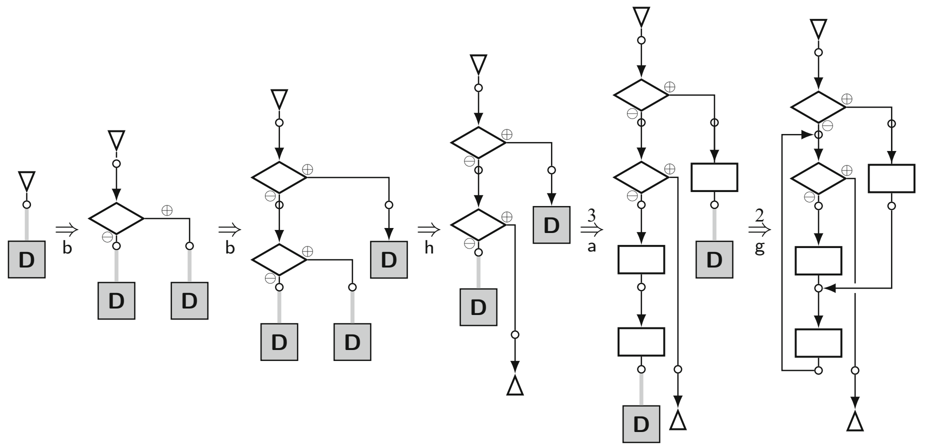


Fig. 5 A derivation of an unstructured control flow diagram

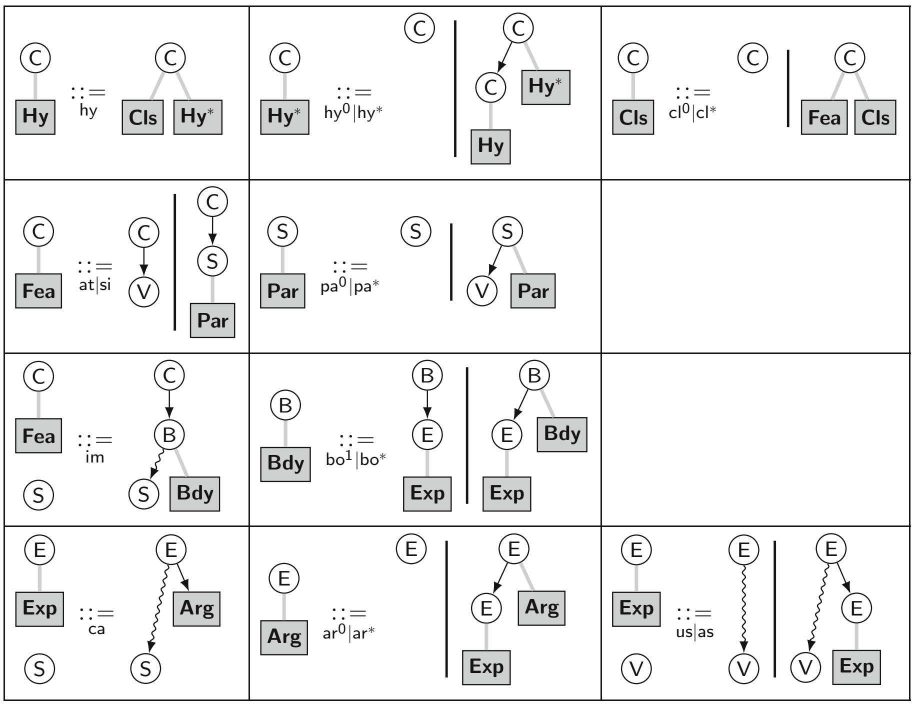


Fig. 6 Contextual rules deriving program graphs

C' that specifies the same set of objects as the original device. Consequently, we say that a restricted class of contextual grammars is a *normal form of contextual grammars* (or of a certain class of contextual grammars) if every contextual grammar (in the class considered) can effectively be transformed into an equivalent grammar belonging to the restricted class.

Many of the normal form properties studied here are inspired by corresponding transformations of context-free graph grammars (hyperedge replacement grammars, [14]), which, in

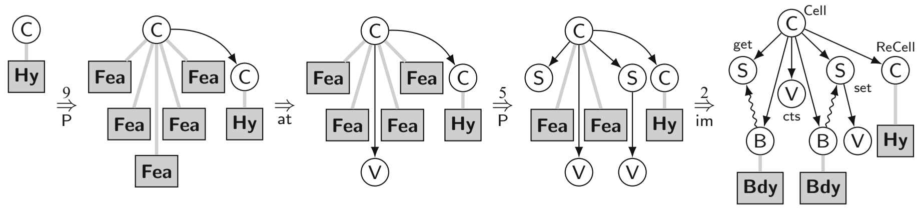


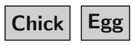
Fig. 7 Snapshots in a derivation of a program graph

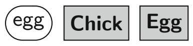
turn, have been inspired by transformations of context-free string grammars, namely by the removal of empty rules, of chain rules, and of rules that do not contribute to the language generated.

However, we first need another normal form property for contextual grammars, which we call context-safety. If a contextual grammar is in this form, the properties above can been shown to be normal form properties for contextual grammars, very much as in the context-free case. So, in a way, context-safety embodies the difference between contextual and context-free grammars. This normal form is defined in the next subsection, and afterwards we consider the normal forms mentioned above, in Sects. 3.2, 3.3, and 3.4.

3.1 Context-safety

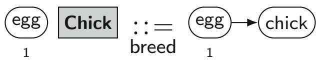
In the following, let us call the label of the (unique) variable in the left-hand side of a contextual rule its *lhs label*. A context-free rule can be applied to a variable x whenever its lhs label is equal to the label of x. In particular, derivation steps replacing different variables in a graph are independent of each other and can be re-ordered without restrictions. For instance, arbitrary context-free rules with lhs label **Chick** or **Egg** can be applied to the graphs

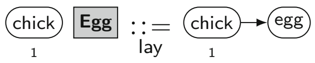
$Z_1 =$ 

and $Z_2 =$ 

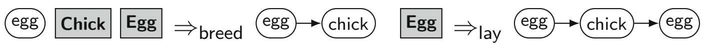
,

in any order. In the contextual case, this is not true any more, because one rule may create the contextual nodes that are needed to apply another rule. In particular, this may give rise to deadlock situations. Consider the contextual rules





(which model the mutual dependency of *chicken* on *eggs*) as an example. Then Rule breed must be applied to Z_2 before Rule lay can be used:

$Z_2 =$ 

None of these rule applies to the graph Z_1, as breed needs the contextual node generated by lay in order to generate the contextual node needed by lay, and vice versa.

We show now that contextual grammars can be turned into a normal form that avoids such deadlocks. The normal form guarantees a property close to the above-mentioned independence in hyperedge replacement grammars. Given a contextual grammar $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$, let us say that a *rule assignment* for a graph $G \in \mathscr{G}_{\mathscr{C}}$ is a mapping *ass* that assigns, to every variable $x \in \bar{G}$, a rule $ass(x) \in \mathscr{R}$ whose lhs label is equal to $\bar{\ell}_G(e)$. In the context-free case, we may choose a rule assignment and then apply the corresponding rules to the variables, and we can do so in any order. As seen above, this may not be true in the contextual case. Clearly, the property that necessarily has to be sacrificed is to be able to apply the chosen rules in any order. However, the more serious difficulty is that sticking to a certain choice of $ass(x)$ for a given variable x may eventually result in a deadlock, if other parts of the derivation terminate without providing the context required for $ass(x)$. For instance, consider adding the rules

$$\boxed{\textsf{Chick}} \underset{\mathsf{chain}}{::=} \boxed{\textsf{Chick}} \qquad\qquad \boxed{\textsf{chick}} \underset{\mathsf{empty}}{::=} \langle\rangle$$

to the example above. Use the start graph Z_2 with its two variables x_1 and x_2, labeled with **Chick** and **Egg**, respectively, and choose the rule assignment given by $ass(x_1) = \mathsf{chain}$ and $ass(x_2) = \mathsf{lay}$. Then $Z_2 \Rightarrow_{\mathsf{chain}} G_1 \Rightarrow_{\mathsf{empty}} G_2$ yields a graph containing x_2 as its unique variable, but $ass(x_2) = \mathsf{lay}$ is not applicable to it. (If there were more rules in the grammar, other rules could of course be applicable to x_2.) Grammars that are built in such a way that this cannot happen must have the following property: suppose that a graph G has been derived from Z, and that a rule assignment ass for G is chosen. After some more steps a graph H may have been obtained, perhaps containing both variables from G and some additional variables (from the right-hand sides of rules that have been applied). Since order is relevant, it may be necessary to apply rules to variables in $var(H)\backslash var(G)$. However, if this set happens to be empty, then we want one of the rules $ass(x)$, $x \in var(H) \subseteq var(G)$ to be applicable to x_1 in H, since otherwise we have a deadlock situation as above.

This gives rise to the following formal definition of context-safety. (Note that the definition makes use of the assumption made after Definition 2.2.)

Definition 3.1 (*Context-Safety*) Let $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z\rangle$ be a contextual grammar. For a graph $G \in \mathscr{G}_{\mathscr{C}}$, let $var(G)$ denote the set of variables occurring in G.

1. A rule assignment ass for a graph $G \in \mathscr{G}_{\mathscr{C}}$ is *context-safe* if the following holds for all derivations $G \Rightarrow^*_{\mathscr{R}} H$: If $\emptyset \neq var(H) \subseteq var(G)$, then there exists a variable $x \in var(H)$ such that $ass(x)$ is applicable to x.
2. Γ is context-safe if every rule assignment for every graph $G \in \mathscr{G}_{\mathscr{C}}$ is context-safe, provided that $Z \Rightarrow^*_{\mathscr{R}} G$.

Note that the derivation $G \Rightarrow^*_{\mathscr{R}} H$ in the first item of Definition 3.1 may be of length 0. Thus, in particular, at least one of the rules $ass(x)$ is applicable to a variable $x \in var(G)$, regardless of how we choose ass.

Example 3.2 (*Context-safety of example grammars*)

1. The grammar for all graphs in Example 2.4 is not context-safe. Assignments to the second graph in Fig. 3 are not context-safe if the a node insertion rule is assigned to one of the variables, and edge insertion rules are assigned to all others. The derivations according to this assignment generate just one node, whereas two nodes are needed for applying the edge insertion rules.
2. The grammar for control flow diagrams in Example 2.6 is not context-safe. Assigning Rule g to the variable in the first graph in Fig. 5 is not context-safe, since the graph does not contain a target node for the goto that shall be inserted. (Note that the match of Rule g must be injective!)
3. The program graph grammar in Example 2.7 is not context-safe. Assignments to the second graph in Fig. 7 are not context-safe if they assign Rule im to some variables and Rule at to all the others, since Rule at fails to generate the S-node needed as a contextual node for applying Rule im.

4. The contextual grammar $(\mathscr{C}, \{\mathsf{breed}, \mathsf{lay}\}, Z_2)$ in the introduction of this section is context-safe. In the only terminal derivation of this grammar shown above, the unique assignments to the variables are context-safe.
5. The contextual grammar $(\mathscr{C}, \{\mathsf{breed}, \mathsf{lay}\}, Z_1)$ is not context-safe. There is no context-safe assignment to the start graph Z_1 since the rules depend circularly on each other.

We will now formulate and prove the main technical result of this paper: every contextual grammar can be turned into an equivalent context-safe one. The idea behind this construction is to use a guess-and-verify strategy to keep track of the order in which the node labels will be introduced by the rules. For this, the variable labels are augmented with a sequence $sq \in \dot{\mathscr{C}}^*$ of those node labels that are not yet present in the graph, and with a set $M \subseteq [sq]$.[1] The set M is needed to record which of the labels in the sequence sq are supposed to be introduced by applying rules to the variable and its descendents. The labels in $[sq]\backslash M$ have been guessed to be introduced by other variables in the graph. Thus, rules are applicable if the labels of their contextual nodes do not occur in sq. When a rule is applied, symbols from the beginning of sq which occur in the right-hand side (and are also in M) are removed from sq and M, and the remaining ones are "distributed" to the descendant variables. While we cannot really guarantee that the labels in sq are indeed introduced in the exact order in which they occur in sq, the control we achieve in this way is enough to ensure context-safety.

Theorem 3.3 (Context-Safe Normal Form) *Context-safe contextual grammars are a normal form of contextual grammars.*

Proof Consider a contextual grammar $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$, where we may without loss of generality assume that Z consists of a single variable x such that $\bar{\ell}_Z(x) = S$ and $type(S) = \varepsilon$. Moreover, to simplify the construction, let us assume that, for every rule $(L, R) \in \mathscr{R}$, the label of each contextual node in $\dot{L}$ is distinct from the labels of all other nodes in $\dot{L}$. Dropping this assumption is easy but tedious: In the construction below, the sequences sq and sets M could contain repeated elements (thus turning M into a multiset), the multiplicity being bounded by the maximum number of occurrences of a single contextual node label in a left-hand side of Γ.

To prove the theorem, it suffices to show that there is a context-safe contextual grammar Γ' such that $\mathscr{L}(\Gamma') = L$, where $L = \{G \in \mathscr{L}(\Gamma) \mid \dot{\ell}_G(\dot{G}) = \dot{\mathscr{C}}\}$. In other words, Γ' generates only those graphs in $\mathscr{L}(\Gamma)$ in which all node labels occur. This is because we may apply the construction to all sub-grammars of Γ obtained by deleting some of the node labels (as well as the rules containing them), and taking the union of the resulting grammars (after having made their sets of variable labels disjoint except for the one that labels the variable in the start graph).

Let X' contain all augmented symbols $A\langle M, sq \rangle$ such that $A \in X$, $sq \in \dot{\mathscr{C}}^*$ is repetition-free, and $M \subseteq [sq]$ is such that sq has a nonempty prefix in M^* unless $sq = \varepsilon$.

We let the set of labels of Γ' be $\mathscr{C}' = (\mathscr{C}\backslash X) \cup X'$. For a graph $G \in \mathscr{G}_{\mathscr{C}'}$ we let $strip(G) \in \mathscr{G}_{\mathscr{C}}$ denote the graph obtained from G by turning each variable label $A\langle M, sq \rangle$ into A.

Let $\mathscr{R}'$ be the set of all rules $r = (L, R)$ over augmented symbols with $strip(r) = (strip(L), strip(R)) \in \mathscr{R}$ which, in addition, satisfy the following. Suppose that the lhs label of r is $A\langle M, sq \rangle$, and $var(R) = \{x_1, \ldots, x_m\}$ with $\bar{\ell}_G(x_i) = A_i\langle M_i, sq_i \rangle$ for $i = 1, \ldots, m$. Then the condition for including r in $\mathscr{R}'$ is that sq can be decomposed into $sq = sq_0 sq'$ such that

[1] Recall that $[sq]$ denotes the set of symbols that occur in sq.

(a) $[sq] \cap \dot{\ell}_L(\dot{L}) = \emptyset$,
(b) $[sq_0] = M \cap \dot{\ell}_R(\dot{R})$,
(c) $M_1, \dots, M_m$ is a partition of $M \backslash \dot{\ell}_R(\dot{R})$, and
(d) for $i = 1, \dots, m$, sq_i is the shortest suffix of sq' such that $M_i \subseteq [sq_i]$.

Note that the augmented rule r is uniquely determined by M, sq, and the assignment of the sets $M_1, \dots, M_m$ to the variables of R; below, we express the latter by saying that x_i *is assigned the responsibility* M_i. Intuitively, condition (a) means that the left-hand side must not contain node labels (and, in particular, contextual node labels) that are not yet assumed to be available in the graph, (b) means that the labels in M that are generated by the rule are those which were guessed to be generated next, (c) means that we distribute the remaining responsibilities for generating node labels in M to the variables in the right-hand side, and (d) means that the sq_i are obtained from the remainder of sq by removing the prefix of labels that are not in M_i. This ensures that $A_i\langle M_i, sq_i\rangle \in X'$. Intuitively, the removal of this prefix is justified because, by (c), such node labels are in the responsibility of some other variable, and have been guessed to be created by that variable before the first node label in sq_i is generated.

Now let $\Gamma' = \langle \mathscr{C}' \cup \{S\}, \mathscr{R}_0 \cup \mathscr{R}', Z\rangle$, where $\mathscr{R}_0$ consists of all rules (Z, Z') such that Z' is obtained by relabeling the variable in Z to the augmented variable name $S\langle \dot{\mathscr{C}}, sq\rangle$, for some ordering sq of $\dot{\mathscr{C}}$ (i.e., sq is a sequence in $\dot{\mathscr{C}}^*$ that contains every label in $\dot{\mathscr{C}}$ exactly once).[2] In the following, we assume that every variable label in $\mathscr{C}'$ is the lhs label of at least one rule in $\mathscr{R}'$. (Obviously, variable labels that do not satisfy this assumption may be removed, together with the rules in whose right-hand sides they occur.)

Claim 1. $\mathscr{L}(\Gamma') \subseteq \mathscr{L}(\Gamma)$.
We have $strip(r) \in \mathscr{R}$ for every rule $r \in \mathscr{R}'$. Hence, for every derivation

$$Z \Rightarrow Z' = G_0 \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_n} G_n \in \mathscr{G}_{\mathscr{C}\backslash X}$$

in Γ' it holds that

$$Z = strip(G_0) \Rightarrow_{strip(r_1)} strip(G_1) \Rightarrow_{strip(r_2)} \cdots \Rightarrow_{strip(r_n)} strip(G_n) = G_n.$$

Claim 2. $L \subseteq \mathscr{L}(\Gamma')$.
Consider a derivation $Z = G_0 \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_n} G_n \in L$ with $r_1, \dots, r_n \in \mathscr{R}$. For every $a \in \dot{\mathscr{C}}$, let $p(a)$ be the least $i \in \{1, \dots, n\}$ such that a occurs in G_i. Note that the nodes labeled with a in $G_{p(a)}$ belong to the right-hand side of the rule $r_{p(a)}$. Hence, for every $j < p(a)$, there is a unique variable in G_j from which these nodes have been generated (directly or indirectly). We call such a variable an *ancestor of* a.

Let $sq = a_1 \cdots a_k$ be any ordering of $\dot{\mathscr{C}}$ such that $p(a_1) \leqslant \cdots \leqslant p(a_k)$. For $i = 0, \dots, n$, we turn G_i into $H_i \in \mathscr{G}_{\mathscr{C}'}$ by relabeling each variable x of G_i to the augmented variable name $\bar{\ell}_{G_i}(x)\langle M(x), sq(x)\rangle$, as follows. $M(x)$ is the set of all node labels of which x is an ancestor, and $sq(x)$ is the shortest suffix of sq such that $M(x) \subseteq [sq(x)]$. In particular, for $i = 0$, since the unique variable x in G_0 is an ancestor of every node label, we have $sq(x) = sq$. It is now straightforward to check that

$$Z \Rightarrow H_0 \Rightarrow_{r'_1} H_1 \Rightarrow_{r'_2} \cdots \Rightarrow_{r'_n} H_n = G_n$$

in Γ', for rules $r'_1, \dots, r'_n \in \mathscr{R}'$ such that $strip(r'_i) = r_i$. If x is the variable in G_{i-1} to which r_i is applied, the rule r'_i is obtained from r_i by

- turning the lhs label or r_i into $\bar{\ell}_{G_{i-1}}(x)\langle M(x), sq(x)\rangle$ and

[2] We can use $\dot{\mathscr{C}}$ in $S\langle \dot{\mathscr{C}}, sq\rangle$ rather than having to guess a subset of $\dot{\mathscr{C}}$, because we want Γ' to generate L rather than the whole language $\mathscr{L}(\Gamma)$.

- assigning every variable x_i in the right-hand side the responsibility $M(x_i)$.

Claim 3. If $Z \Rightarrow^+ G$ in Γ', where $var(G) = \{x_1, \ldots, x_m\}$ and $\bar{\ell}_G(x_i) = A_i\langle M_i, sq_i\rangle$ for $i = 1, \ldots, m$, then $\dot{\ell}_G(\dot{G}) \supseteq \dot{\mathscr{C}} \setminus \bigcup_{i=1}^m M_i$. Moreover, $\bigcup_{i=1}^m M_i \subseteq \bigcup_{i=1}^m [sq_i]$ and thus $\dot{\ell}_G(\dot{G}) \supseteq \dot{\mathscr{C}} \setminus \bigcup_{i=1}^m [sq_i]$.
For one-step derivations $Z \Rightarrow G$ in Γ', this holds by the construction of the rules whose lhs label is S. Moreover, the property $\dot{\ell}_G(\dot{G}) \supseteq \dot{\mathscr{C}} \setminus \bigcup_{i=1}^m M_i$ is preserved by the rules in $\mathscr{R}'$, thanks to (c), and the property $\bigcup_{i=1}^m M_i \subseteq \bigcup_{i=1}^m [sq_i]$ is preserved thanks to (b)–(d). This completes the proof of Claim 3.

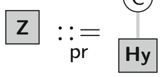
We can finally prove the statement of the theorem: if $Z \Rightarrow^+ G$ in Γ', then all rule assignments for G are context-safe. Consider a rule assignment ass for G and a derivation $G \Rightarrow^n_{\mathscr{R}'} H$ such that $\emptyset \neq var(H) \subseteq var(G)$. We proceed by induction on n to show that there exists a variable $x \in var(H)$ such that $ass(x)$ is applicable to x.

($n = 0$) For a graph G and a variable x occurring in G, let $sq_G(x)$ denote the sequence of node labels such that $\bar{\ell}_G(x) = A\langle M, sq_G(x)\rangle$ for some $A \in X$ and $M \subseteq \dot{\mathscr{C}}$. By the construction of the rules $r = (L, R)$ in $\mathscr{R}'$, if x is the variable in L and y is any variable in R, then $sq_R(y)$ is a suffix of $sq_L(x)$. By an obvious induction on the length of the derivation yielding G, this yields the following: If $x_1, \ldots, x_m$ are the variables in G, then $sq_G(x_1), \ldots, sq_G(x_m)$ are suffixes of one and the same sequence (namely the one nondeterministically chosen in the first step of the derivation $Z \Rightarrow^+ G$). Consequently, there is an $h \in \{1, \ldots, m\}$, such that each $sq_G(x_i)$ is a suffix of $sq_G(x_h)$. By Claim 3, $\dot{\ell}_G(\dot{G}) \supseteq \dot{\mathscr{C}} \setminus \bigcup_{i=1}^m [sq_G(x_i)] = \dot{\mathscr{C}} \setminus [sq_G(x_h)]$. Since the rule $ass(x_h)$ fulfills condition (a), this means that the label of each contextual node in its left-hand side appears in G. Thus, $ass(x_h)$ is applicable to x_h.

($n \to n + 1$) For the inductive step, let $n \geqslant 1$ and $G \Rightarrow_{\mathscr{R}'} G_1 \Rightarrow^{n-1}_{\mathscr{R}'} H$. Let ass_1 be an arbitrary rule assignment for G_1 such that $ass_1(x) = ass(x)$ for all variables $x \in var(G) \cap var(G_1)$. Note that ass_1 exists, because of our assumption that every variable label is the label of at least one rule. Now, applying the induction hypothesis to the derivation $G_1 \Rightarrow^{n-1}_{\mathscr{R}'} H$ yields a variable x in H such that $ass_1(x)$ applies to x. However, since $var(H) \subseteq var(G)$, we have $x \in var(G)$ and $ass(x) = ass_1(x)$. □

We note here that, from a practical point of view, the construction of Γ' in the preceding proof may be optimized with respect to the number of variable labels and rules. This is because the annotations M and sq can be restricted to the subset of those labels which, in Γ, are node labels of contextual nodes.

Example 3.4 (*Context-safe form of the program graph grammar*) For the context-safe form of the program graph grammar PG in Example 2.7, we add a start rule pr with a nullary variable named **Z**:



Since only the labels S and V label contextual nodes (in rules im, ca, us, and as), we restrict augmentations to these labels.

Then the new variable names are of the form $A\langle M, sq\rangle$, where $A \in \{$**Z**, **Hy**, **Hy***, **Cls**, **Fea**, **Par**, **Bdy**, **Exp**, **Arg**$\}$, $M \subseteq \{\mathsf{S}, \mathsf{V}\}$ and $sq \in \{\varepsilon, \mathsf{S}, \mathsf{V}, \mathsf{SV}, \mathsf{VS}\}$. The requirement (in the proof of Theorem 3.3) that "$M \subseteq [sq]$ *is such that sq has a nonempty prefix in* M^* *unless* $sq = \varepsilon$" allows the following augmentations $\langle M, sq\rangle$:

$$\{\langle\emptyset, \varepsilon\rangle, \langle\{\mathsf{S}\}, \mathsf{S}\rangle, \langle\{\mathsf{S}\}, \mathsf{SV}\rangle, \langle\{\mathsf{V}\}, \mathsf{V}\rangle, \langle\{\mathsf{V}\}, \mathsf{VS}\rangle, \langle\{\mathsf{S}, \mathsf{V}\}, \mathsf{SV}\rangle, \langle\{\mathsf{S}, \mathsf{V}\}, \mathsf{VS}\rangle\}$$

The program graph rules in Fig. 6 have up to two variables on their right-hand sides. Each rule gives rise to one or more rules obtained by relabeling the variables in its left-hand side and in its right-hand side in all possible ways that satisfy the requirements (a)–(d) in the proof.

Table 1 summarizes the augmentation of the variable names of selected context-safe rules in the program graph example. For the start rule pr, the left-hand side symbol Z stays as it was, and we get two augmented rules, with variable names $\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$ and $\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$ on the right-hand side. Rule hy has 17 augmented variations, for all augmentations of the left-hand side variable, and all distributions of these augmentations to the right-hand side variables; the augmentations for rules hy*, cl* bo*, and ar* are built analogously. The rule

Table 1 Augmentated variables of the context-safe program graph grammar

P	lhs label	rhs label(s)	
pr_1	$\mathbf{Z}$	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	N/A
pr_2	$\mathbf{Z}$	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	N/A
hy_1	$\mathbf{Hy}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Cls}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Hy}^*\langle\emptyset,\varepsilon\rangle$
hy_2	$\mathbf{Hy}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	$\mathbf{Cls}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{V}\},\mathsf{V}\rangle$
hy_3	$\mathbf{Hy}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	$\mathbf{Cls}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	$\mathbf{Hy}^*\langle\emptyset,\varepsilon\rangle$
hy_4	$\mathbf{Hy}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Cls}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{V}\},\mathsf{VS}\rangle$
hy_5	$\mathbf{Hy}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Cls}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Hy}^*\langle\emptyset,\varepsilon\rangle$
hy_6	$\mathbf{Hy}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	$\mathbf{Cls}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{S}\},\mathsf{S}\rangle$
hy_7	$\mathbf{Hy}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	$\mathbf{Cls}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	$\mathbf{Hy}^*\langle\emptyset,\varepsilon\rangle$
hy_8	$\mathbf{Hy}\langle\{\mathsf{S}\},\mathsf{SV}\rangle$	$\mathbf{Cls}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{S}\},\mathsf{SV}\rangle$
hy_9	$\mathbf{Hy}\langle\{\mathsf{S}\},\mathsf{SV}\rangle$	$\mathbf{Cls}\langle\{\mathsf{S}\},\mathsf{SV}\rangle$	$\mathbf{Hy}^*\langle\emptyset,\varepsilon\rangle$
hy_{10}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	$\mathbf{Cls}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$
hy_{11}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	$\mathbf{Cls}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{S}\},\mathsf{SV}\rangle$
hy_{12}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	$\mathbf{Cls}\langle\{\mathsf{S}\},\mathsf{SV}\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{V}\},\mathsf{V}\rangle$
hy_{13}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	$\mathbf{Cls}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	$\mathbf{Hy}^*\langle\emptyset,\varepsilon\rangle$
hy_{14}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Cls}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$
hy_{15}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Cls}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{S}\},\mathsf{S}\rangle$
hy_{16}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Cls}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	$\mathbf{Hy}^*\langle\{\mathsf{V}\},\mathsf{VS}\rangle$
hy_{17}	$\mathbf{Hy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Cls}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Hy}^*\langle\emptyset,\varepsilon\rangle$
hy^0_1	$\mathbf{Hy}\langle\emptyset,\varepsilon\rangle$	N/A	N/A
at_1	$\mathbf{Fea}\langle\emptyset,\varepsilon\rangle$	N/A	N/A
at_2	$\mathbf{Fea}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	N/A	N/A
at_3	$\mathbf{Fea}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	N/A	N/A
si_1	$\mathbf{Fea}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Par}\langle\emptyset,\varepsilon\rangle$	N/A
si_2	$\mathbf{Fea}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	$\mathbf{Par}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	N/A
si_3	$\mathbf{Fea}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Par}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	N/A
si_4	$\mathbf{Fea}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	$\mathbf{Par}\langle\emptyset,\varepsilon\rangle$	N/A
si_5	$\mathbf{Fea}\langle\{\mathsf{S}\},\mathsf{SV}\rangle$	$\mathbf{Par}\langle\emptyset,\varepsilon\rangle$	N/A
si_6	$\mathbf{Fea}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	$\mathbf{Par}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	N/A
bo^1_1	$\mathbf{Bdy}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Exp}\langle\emptyset,\varepsilon\rangle$	N/A
bo^1_2	$\mathbf{Bdy}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	$\mathbf{Exp}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	N/A
bo^1_3	$\mathbf{Bdy}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Exp}\langle\{\mathsf{V}\},\mathsf{VS}\rangle$	N/A
bo^1_4	$\mathbf{Bdy}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	$\mathbf{Exp}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	N/A
bo^1_5	$\mathbf{Bdy}\langle\{\mathsf{S}\},\mathsf{SV}\rangle$	$\mathbf{Exp}\langle\{\mathsf{S}\},\mathsf{SV}\rangle$	N/A
bo^1_6	$\mathbf{Bdy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	$\mathbf{Exp}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{SV}\rangle$	N/A
bo^1_7	$\mathbf{Bdy}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	$\mathbf{Exp}\langle\{\mathsf{S},\mathsf{V}\},\mathsf{VS}\rangle$	N/A
im_1	$\mathbf{Fea}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Bdy}\langle\emptyset,\varepsilon\rangle$	N/A
im_2	$\mathbf{Fea}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	$\mathbf{Bdy}\langle\{\mathsf{V}\},\mathsf{V}\rangle$	N/A
as_1	$\mathbf{Exp}\langle\emptyset,\varepsilon\rangle$	$\mathbf{Exp}\langle\emptyset,\varepsilon\rangle$	N/A
as_2	$\mathbf{Exp}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	$\mathbf{Exp}\langle\{\mathsf{S}\},\mathsf{S}\rangle$	N/A

hy^0 has a single augmentation; this is the same for the rules cl^0, pa^0, and us, which have no variable on their right-hand side and do not generate any node labeled with S or V. Rule at is similar, but since it generates a node labeled with V, there are three possible augmentations: the left-hand side may be labeled with $\mathbf{Fea}\langle\emptyset, \varepsilon\rangle$, with $\mathbf{Fea}\langle\{\mathsf{V}\}, \mathsf{V}\rangle$, or with $\mathbf{Fea}\langle\{\mathsf{V}\}, \mathsf{VS}\rangle$. Rule si has six augmentations; whenever the left-hand side variable "promises" to generate an S-node ($\mathsf{S} \in M$), this node will be generated first (is the head of sq). Rule bo^1 has a single variable on the right-hand side, and needs seven augmentations, for all possible augmentations of variables. Rule im has only two augmentations, as the node label S does occur on their left-hand side (like ca, which is not shown). Finally, the augmentations of rule as are analoguous, because V occurs on its left-hand side. Altogether, the context-safe form PG' of the program graph grammar PG has 113 augmented rules, for 18 rules in the original grammar. □

It is worthwhile observing that the mapping *strip* in the proof of Theorem 3.3 turns derivations of the context-safe grammar Γ' into derivations of the original grammar Γ. We note this slightly stronger form of the theorem as a corollary. For this, let us say that an *edge relabeling* is a mapping *rel* on edge labels. Such an edge relabeling is extended to a mapping on graphs and rules in the obvious way: for an edge relabeling $rel\colon \bar{\mathscr{C}} \to \bar{\mathscr{C}}'$ and a graph $G \in \mathscr{G}_{\mathscr{C}}$ we let $rel(G) = \langle \dot{G}, \bar{G}, att_G, \dot{\ell}_G, rel \circ \bar{\ell}_G\rangle$. For a rule $r = (L, R)$, $rel(r) = (rel(L), rel(R))$.

Corollary 3.5 *For every contextual grammar Γ one can effectively construct an equivalent context-safe contextual grammar Γ' together with an edge relabeling rel such that $rel(Z') \Rightarrow_{rel(r_1)} rel(G_1) \Rightarrow_{rel(r_2)} \cdots \Rightarrow_{rel(r_n)} rel(G_n)$ is a derivation in Γ for every derivation $Z' \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_n} G_n$ in Γ'.*

To be precise, we note here that the construction in the proof of Theorem 3.3 does not entirely fulfil Corollary 3.5 (with *strip* as *rel*), because of the initial rules (Z, Z'). However, these rules can easily be removed by composing them with the rules applying to Z', as proposed in Definition 3.12.

Corollary 3.5 will be used in Sect. 3.4 to show that contextual grammars can effectively be reduced. However, let us first show (in the next two subsections) that both empty and chain rules can be removed from contextual grammars.

3.2 Removing empty rules

We say that a rule (L, R) with $\bar{L} = \{x\}$ is an *empty rule* if $R = L - x$, and a *chain rule* if $R - y = L - x$ for a variable $y \in \bar{R}$. In the case of chain rules, we say that $\bar{\ell}_R(y)$ is the *rhs label* of the rule. Note that both empty and chain rules are more general than in the context-free case, because L may contain contextual nodes. Hence, the applicability of these rules may be subject to the existence of nodes with certain labels elsewhere in the graph. Moreover, in the case of chain rules it is not required that the variable y is attached to the same nodes as x. Hence, chain rules can "move" a variable through a graph.

Similar to the context-free case [14, Section IV.1], the overall strategy for removing empty and chain rules is to compose them with other rules. In the case of empty rules, no real composition is required. We just determine the labels of those variables that can, possibly via a sequence of derivation steps, be removed without generating any terminal node or edge. Then we build new rules by removing some of these variables from the right-hand sides of the original rules, thus anticipating the application of empty rules. Collecting the variables that can be removed works precisely as in the context-free case, i.e., we do not take the contextual nodes into account at all. For this, consider a contextual grammar $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$. Let $P_{\mathscr{R}}$ be the set of ordinary context-free Chomsky rules given as follows: if $\mathscr{R}$ contains a rule $r = (L, R)$ such that $\dot{L} = \dot{R}$ and $\bar{R} = \{x_1, \ldots, x_k\} \subseteq var(R)$ (i.e., an application of r adds neither nodes nor terminal edges to the graph) then $\tilde{\mathscr{R}}$ contains the Chomsky rule $p_r = (A \to w)$ where A is the lhs label of r and $w = \bar{\ell}_R(x_1) \cdots \bar{\ell}_R(x_k)$, arranging the variables in R in some arbitrary order. Now, define X_ε^Γ to be the set of all variable labels $A \in X$ such that $A \to^*_{\tilde{\mathscr{R}}} \varepsilon$. Note that X_ε^Γ can be computed by the usual iterative procedure. For $A \in X_\varepsilon^\Gamma$ we denote by $depth(A)$ the length d of the shortest derivation $A \to^d_{\tilde{\mathscr{R}}} \varepsilon$.

As mentioned, the basic idea for the removal of empty rules from contextual grammars is the same as for hyperedge replacement grammars: We add new rules that are obtained by removing variables named by X_ε^Γ from their right-hand sides. Let us illustrate this using the program graph grammar as an example.

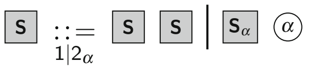
Example 3.6 (*Removing empty rules from the program graph grammar*) In the program graph grammar PG of Example 2.7, we have

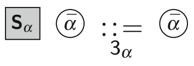
$$\tilde{\mathsf{P}} = \{\textbf{Hy} \to \textbf{Cls}\,\textbf{Hy}^*, \textbf{Hy}^* \to \varepsilon, \textbf{Cls} \to \varepsilon, \textbf{Cls} \to \textbf{Fea}\,\textbf{Cls}, \textbf{Par} \to \varepsilon, \textbf{Arg} \to \varepsilon\}$$

which yields the set $X_\varepsilon^{\mathsf{PG}} = \{\textbf{Hy}, \textbf{Hy}^*, \textbf{Cls}, \textbf{Par}, \textbf{Arg}\}$ of variables generating ε. The set $P_\delta = \{\mathsf{pr}, \mathsf{hy}, \mathsf{hy}^*, \mathsf{cl}^*, \mathsf{si}, \mathsf{pa}^*, \mathsf{ca}, \mathsf{ar}^*\} \subseteq \mathsf{P}$ contains the rules where variables with names in $X_\varepsilon^{\mathsf{PG}}$ occur on the right-hand side. We introduce variants of these rules where some of these variables are removed from the right-hand sides, and delete the original empty rules as well as the empty rule that is introduced by removing the variables named **Cls** and **Hy*** from the right-hand side of rule hy. We get the set of rules shown in Fig. 8. So 17 rules of P, plus the start rule pr, are replaced with 25 non-empty rules. □

In Example 3.6, removal of empty rules happens to work correctly. However, to make it work in general, it turns out that we have to assume that the grammar is context-safe. This is illustrated by the following example.

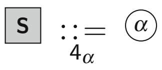
Example 3.7 (*Removal of empty rules*) Consider two node labels a, ā with $\bar{\bar{\mathsf{a}}} = \mathsf{a}$, and the following rules, where $\alpha \in \{\mathsf{a}, \bar{\mathsf{a}}\}$:





The grammar generates the language of all discrete graphs over $\{\mathsf{a}, \bar{\mathsf{a}}\}$ that contain both labels. (Recall that a discrete graph is a graph without edges.)

When we apply the construction of Lemma 3.8 to this grammar, we obtain $X_\varepsilon^\Gamma = \{\mathsf{S_a}, \mathsf{S_{\bar{a}}}\}$; removal of empty rules from this grammar deletes the empty rules 3_a and $3_{\bar{\mathsf{a}}}$, and adds the following rules:



However, the original contextual grammar does not generate graphs with a single node: whereas the original rule 2_a generates the graph $\boxed{\mathsf{S_a}}$ $\textcircled{a}$, rule 3_a is not applicable to this graph as its contextual node labeled ā is missing.

This shows that the construction does not work without context-safety. One may attempt to circumvent this problem by not simply removing variables with labels in X_ε^Γ, but composing

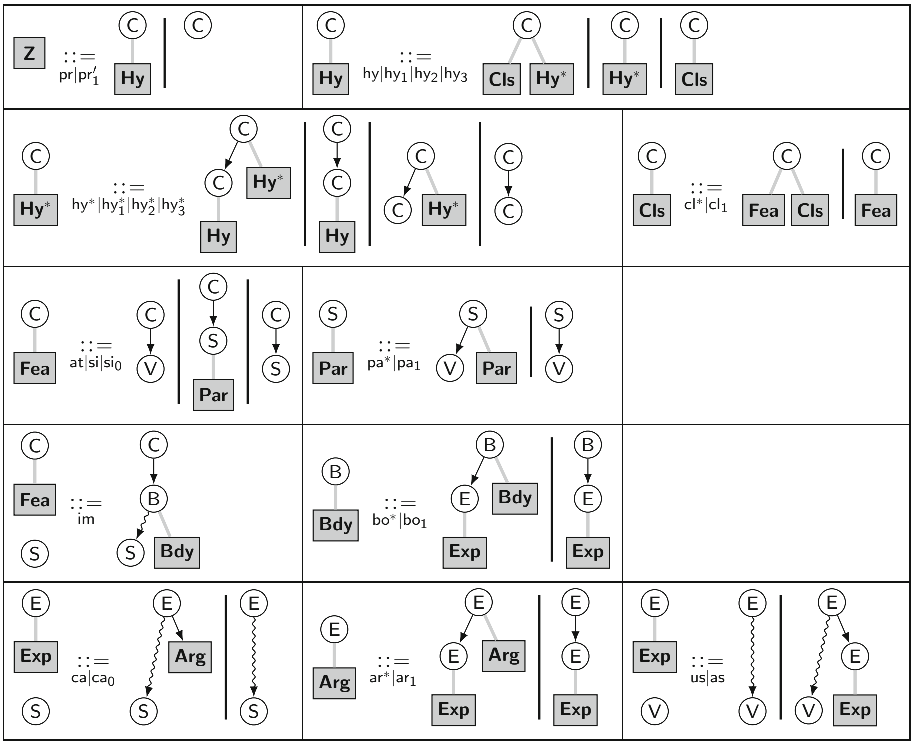
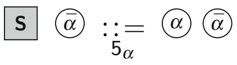
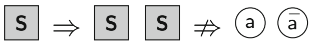


Fig. 8 Removing empty rules from the program graph grammar in Fig. 6

the original rules with empty rules in such a way that the contextual nodes of those empty rules are added to the left-hand side of the resulting rule. This notion of rule composition will be formalized in Definition 3.12. However, for the removal of empty rules, it does not yield the desired result. The composition of 2_{a} and $2_{\bar{\mathsf{a}}}$ with the empty rules 3_{a} and $3_{\bar{\mathsf{a}}}$, resp., yields the following new rules:



Unfortunately, adding these rules and removing the empty rules yields



due to a deadlock: Rule 5_{a} cannot be applied to the graph in the middle because the graph does not contain a node labeled a, and vice versa for rule $5_{\bar{\mathsf{a}}}$. So the language of the new grammar is empty.

Thus we have to turn the original grammar into a context-safe one (in simplified form as the subscript *sq* is not needed here). This yields the following rules:

$$\boxed{\mathsf{S}^M} ::= \boxed{\mathsf{S}^{M_1}} \; \boxed{\mathsf{S}^{M_2}} \;\Big|\; \boxed{\mathsf{S}_\alpha^{M\setminus\{\alpha\}}} \; \textcircled{\alpha} \quad \text{where } M \subseteq \{\mathsf{a}, \bar{\mathsf{a}}\}, M_1 \uplus M_2 = M$$

$$\boxed{\mathsf{S}_\alpha^M} \; \textcircled{\bar\alpha} ::= \textcircled{\bar\alpha} \quad \text{where } M \subseteq \{\alpha\}, \text{i.e, } \bar\alpha \notin M$$

Now, $X_\varepsilon^\Gamma = \left\{ \mathsf{S}_\alpha^M \mid M \subseteq \{\alpha\} \right\}$, and the removal of empty rules yields $\boxed{\mathsf{S}^M} ::= \textcircled{\alpha}$ for $M \subseteq \{\alpha\}$. This rule is correct since S^M with $\bar\alpha \notin M$ only appears in the context of another $\boxed{\mathsf{S}^{\bar M}}$ with $\bar\alpha \in \bar M$, which gives rise to a node labeled $\bar\alpha$. □

We can now show that the removal of empty rules indeed works correctly, under the condition that it is applied to a context-safe grammar.

Lemma 3.8 (Removal of Empty Rules) *Let $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$ be a context-safe contextual grammar with $\bar Z = \{x\}$ and $\dot Z = \emptyset$.*[3] *If Γ' is the contextual grammar obtained from Γ by*

1. *replacing every rule $(L, R) \in \mathscr{R}$ by the set of all rules of the form $(L, R - E)$ such that $E \subseteq \{x \in var(R) \mid \bar\ell_R(x) \in X_\varepsilon^\Gamma\}$ and*
2. *removing all empty rules from the resulting set of rules*

then $L(\Gamma') = L(\Gamma) \setminus \{\langle\rangle\}$.

Proof By very much the same arguments as in the context-free case, it follows that $L(\Gamma)\setminus\{\langle\rangle\} \subseteq L(\Gamma')$. Thus, it remains to be shown that $L(\Gamma') \subseteq L(\Gamma)$. For this, we need context-safety. By induction on the length of derivations, given any derivation $Z \Rightarrow_{\mathscr{R}'}^* G$, there exist a derivation $Z \Rightarrow_{\mathscr{R}}^* H$ and a set $E \subseteq \{x \in var(H) \mid \bar\ell_H(x) \in X_\varepsilon^\Gamma\}$ such that $G = H - E$. We have to consider the case where $G \in \mathscr{G}_{\mathscr{C}\setminus X}$, which means that $E = var(H)$. Thus, denoting $H - var(H)$ by $\underline{H}$, we have to show that $H \Rightarrow_{\mathscr{R}}^* \underline{H}$. For this, we show by induction on $n = \sum_{x \in var(H)} depth(\bar\ell_H(x))$ that $H \Rightarrow_{\mathscr{R}}^n \underline{H}$ for every graph H such that $Z \Rightarrow_{\mathscr{R}}^* H$ and $\bar\ell_H(var(H)) \subseteq X_\varepsilon^\Gamma$.

For $n = 0$ we have $var(H) = \emptyset$, and hence there is nothing to show. Now, assume that $n > 0$. For every variable $x \in var(H)$ with $depth(\bar\ell_H(x)) = d$ there is a derivation $\bar\ell_H(x) \rightarrow_{\tilde{\mathscr{R}}}^d \varepsilon$. Let ass be the assignment that assigns to each $x \in var(H)$ the rule r such that p_r is the first rule applied in the derivation $\bar\ell_H(x) \rightarrow_{\tilde{\mathscr{R}}}^d \varepsilon$. Then, by context-safety, one of the rules $ass(x)$ (for $x \in var(H)$) applies to x, yielding $H \Rightarrow_{\mathscr{R}} H'$ with $\underline{H'} = \underline{H}$ and $\sum_{x \in var(H')} depth(\bar\ell_{H'}(x)) = \sum_{x \in var(H)} depth(\bar\ell_H(x)) - 1$. Thus, the induction hypothesis applies to H', yielding $H \Rightarrow_{\mathscr{R}} H' \Rightarrow_{\mathscr{R}}^{n-1} \underline{H'} = \underline{H}$, as claimed. □

3.3 Removing chain rules

Just as the removal of empty rules, the removal of chain rules follows the pattern known from the theory of context-free grammars. Roughly speaking, an arbitrary sequence of applications of chain rules, followed by an application of a rule that is not a chain rule, is composed into a single rule. Composing two rules r and r' means to apply r' to the right-hand side of r, which yields the right-hand side of the composed rule. Unfortunately, things are not quite as simple for contextual grammars, because an application of r' may make use of contextual nodes that do not belong to the right-hand side of r. Thus, to make sure that the composed rules faithfully implement r followed by r', the left-hand side may have to be extended by additional contextual nodes. We also have to take into account that an application of r may

[3] Recall that, by the remark after Definition 2.3, the requirement $\dot Z = \emptyset$ is no limitation.

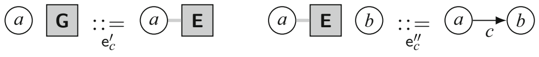


Fig. 9 Rules with singleton contexts for rule e_c in Fig. 2

not immediately be followed by an application of r'. It may be the case that contextual nodes required by r' are not yet available, as they are still to be generated in another branch of the derivation. Fortunately, this problem can be solved if r is a chain rule: As the following lemma shows, we can reorder derivation steps in such a way that every application of a chain rule is immediately followed by an application of a rule that replaces the variable in the right-hand side of that chain rule. For technical convenience, given two arbitrary contextual rules r and r', let us say that $G \Rightarrow_{rr'} H$ *via* I if $G \Rightarrow_r I \Rightarrow_{r'} H$, where the second step replaces one of the variables coming from the right-hand side of r. We simply write $G \Rightarrow_{rr'} H$ without specifying I if I is of no particular interest.

Lemma 3.9 *Let $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z\rangle$ be a contextual grammar and $G \in \mathscr{L}(\Gamma)$. Then there is a derivation $Z = G_0 \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_n} G_n = G$ such that, for all $i \in \{1, \ldots, n-1\}$, if r_i is a chain rule then $G_{i-1} \Rightarrow_{r_i r_{i+1}} G_{i+1}$ via G_i.*

Proof For graphs G, H, and I, if $G \Rightarrow_r I$ using a chain rule r then $\dot{I} = \dot{G}$. Hence, if $I \Rightarrow_{r'} H$ via a matching m, replacing a variable in $\bar{G} \cap \bar{I}$, then r' applies to G as well, using the same matching m. Consequently, the steps can be switched yielding $G \Rightarrow_{r'} I' \Rightarrow_r H$ for a graph I'. This shows that an application of a chain rule can be shifted to the right as long as the next derivation step replaces a variable other than the one in the right-hand side of this chain rule. Hence, the property asserted in the lemma can be guaranteed by shifting all applications of chain rules to the right as far as possible. □

As a consequence of the previous lemma let us note in passing that contextual grammars require at most one contextual node in their left-hand sides.

Lemma 3.10 *Contextual grammars in which each rule contains at most one contextual node are a normal form of contextual grammars.*

Proof Using Lemma 3.9 this is straightforward. Suppose we wish to implement a rule r whose left-hand side contains a variable with k attached nodes and $l \geqslant 1$ contextual nodes. We use l chain rules $r_1, \ldots, r_l$ 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 r'. Clearly, every derivation step $G \Rightarrow_r H$ can be turned into a derivation $G \Rightarrow_{r_1 \cdots r_l r'} H$ of length $l+1$. (Here we extend the notation $\Rightarrow_{r_1 r_2}$ to l chain rules followed by one arbitrary rule in the obvious way.) Conversely, by Lemma 3.9 every terminating derivation in the new grammar can be rewritten in such a way that $r_1, \ldots, r_l$ and r' only occur in subderivations of the form $G \Rightarrow_{r_1 \cdots r_l r'} H$, which means that each such subderivation can be replaced by $G \Rightarrow_r H$ yielding a derivation in the original grammar. □

Example 3.11 (*Rules with singleton contexts*) In the grammar in Example 2.4 generating all graphs, the edge-inserting rules have two contextual nodes. We can replace the rules e_c (for $c \in \bar{\mathscr{C}}$) by two rules e'_c and e''_c for a fresh variable name $\mathbf{E}$ with $type(\mathbf{E}) = a$, as shown in Fig. 9. □

Let us now formalize the notion of rule composition discussed informally above.

Definition 3.12 (*Composition of Contextual Rules*) Let r and r' be contextual rules. A *composition of r and r'* is a contextual rule (L, R) such that $L \Rightarrow_{rr'} R$.

In the following, we denote the set of all compositions of contextual rules r and r' by $r\,;r'$. Note that $r\,;r'$ is an infinite set, as contextual rules may have any number of contextual nodes. We will soon take care of this problem, but let us first note that rule composition works as expected. For a detailed treatment including proofs (in a much more general case) see [12, Sections 3.4.1and 5.4].

Fact 3.13 Let r and r' be contextual rules, and let G and H be graphs. Then $G \Rightarrow_{rr'} H$ if and only if there is a rule $p \in r\,;r'$ such that $G \Rightarrow_p H$.

To circumvent the problem that composition creates infinitely many rules, we define a partial order on contextual rules, as follows. For contextual rules $r = (L, R)$ and $r' = (L', R')$, we let $r \sqsubseteq r'$ if there is a discrete graph D such that $L' = L \uplus D$ and $R' = R \uplus D$ (up to isomorphism, as usual).

Observation 3.14 For graphs $G, H \in \mathscr{G}_{\mathscr{C}}$ and contextual rules $r \sqsubseteq r'$, if $G \Rightarrow_{r'} H$ then $G \Rightarrow_r H$.

Lemma 3.15 *Let $\mathscr{R}$ be a finite set of rules over a finite labeling alphabet $\mathscr{C}$ and let $\overline{\mathscr{R}}$ be the set of all rules $\overline{r}$ over $\mathscr{C}$ such that $r \sqsubseteq \overline{r}$ for some $r \in \mathscr{R}$. Then $(\overline{\mathscr{R}}, \sqsubseteq)$ is a well partial order*[4] *(wpo, for short).*

Proof Let $\dot{\mathscr{C}} = \{a_1, \ldots, a_n\}$ and assume without loss of generality that the rules in $\mathscr{R}$ are pairwise incomparable w.r.t. $\sqsubseteq$. Every rule $\overline{r} \in \overline{\mathscr{R}}$ has the form $(L \uplus D, R \uplus D)$ for a unique rule $r = (L, R) \in \mathscr{R}$ and a unique discrete graph D. Furthermore, D can be represented as $d = (d_1, \ldots, d_k) \in \mathbb{N}^k$, where $d_i = |\{v \in \dot{D} \mid \dot{\ell}_D(v) = a_i\}|$ for $1 \leqslant i \leqslant n$. Hence, we may denote $\overline{r}$ by $r + d$. Clearly, using this notation, given two rules $r + d$ and $r' + d'$ in $\overline{R}$, we have $r + d \sqsubseteq r' + d'$ if and only if $r = r'$ and $d \leqslant d'$, where $\leqslant$ is the usual partial order on $\mathbb{N}^k$. This proves the statement since $(\mathbb{N}^k, \leqslant)$ is a wpo and $\mathscr{R}$ is finite (using the fact that the union of finitely many wpos is a wpo). □

The basic idea for removing a chain rule is to compose it with the rules for the variable on its right-hand side. For the program graph grammar, this is easily achieved.

Example 3.16 (*Removing chain rules from the program graph grammar*) The program graph grammer in Example 2.7 does not contain a chain rule, but the removal of empty rules in Example 3.6 introduces chain rules hy_2, hy_3, and cl_1, shown in Fig. 8. Composing these rules with all rules for the variable names on their right-hand sides yields composed rules cl_2 to cl_5, and hy_4 to hy_{12}. Two of these rules are contextual (by composition with rule im). The new rules are shown in Fig. 10. The resulting grammar has 31 instead of 26 rules altogether. □

We can now show how to remove chain rules from contextual grammars.

Lemma 3.17 (Removal of Chain Rules) *For every contextual grammar, one can effectively construct an equivalent contextual grammar that does not contain chain rules.*

Proof Let $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$ be a contextual grammar. We iteratively add compositions of rules to $\mathscr{R}$, as follows: Let $\mathscr{R}_0 = \mathscr{R}$. For each $i = 0, 1, \ldots$, choose a chain rule $r \in \mathscr{R}$ and a rule $r' \in \mathscr{R}_i$ which is not a chain rule. Now, let $\mathscr{R}_{i+1} = \mathscr{R}_i \cup \{p\}$ for a minimal rule

[4] Recall that a wpo is a partial order that is well-founded (there are no infinite descending chains) and has no infinite antichains (every infinite subset contains comparable elements).

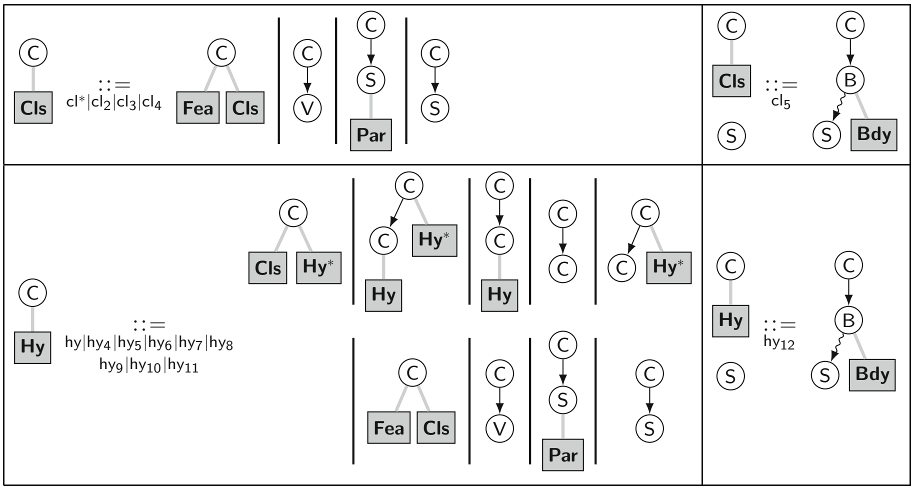


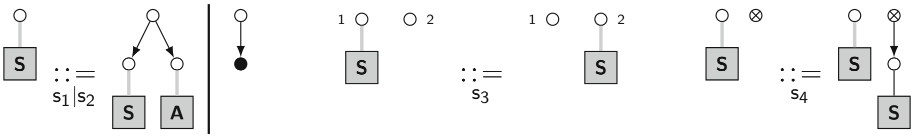
Fig. 10 Removing chain rules from the program graph grammar of Fig. 8

$p \in r\,;r'$ (w.r.t. $\sqsubseteq$) such that $q \not\sqsubseteq p$ for all $q \in \mathscr{R}_i$. Clearly, such a rule p can effectively be found if it exists. The process stops when there are no more rules $r \in \mathscr{R}$, $r' \in \mathscr{R}_i$, and $q \in r\,;r'$ of the kind required. Let i_0 be the index i at which this happens and define $\mathscr{R}' = \{r \in \mathscr{R}_{i_0} \mid r \text{ is not a chain rule}\}$. We claim that i_0 exists (i.e., the iteration does eventually stop) and that $\Gamma' = \langle \mathscr{C}, \mathscr{R}', Z\rangle$ is equivalent to Γ.

To see that i_0 exists, notice that all rules in $\mathscr{R}_i$ are of the form $(L \uplus D, R \uplus D')$, where L and R are left- and right-hand sides of rules in $\mathscr{R}$ and D and D' are discrete graphs. Since $\mathscr{R}$ is finite, it follows that there is a finite set $\mathscr{R}''$ of rules over $\mathscr{C}$ such that, for every rule $r \in \mathscr{R}_i$, there is a rule $r_0 \in \mathscr{R}''$ with $r_0 \sqsubseteq r$. Hence, taking $\mathscr{R}''$ as $\mathscr{R}$ in Lemma 3.15, all $\mathscr{R}_i$ are subsets of the well partial order $(\overline{\mathscr{R}''}, \sqsubseteq)$. By construction, the rules $r_1, r_2, \ldots$ added to the sets $\mathscr{R}_i$ satisfy $r_1 \not\sqsubseteq r_2 \not\sqsubseteq \cdots$. Hence, if this sequences were infinite, it would either contain an infinite descending chain or an antichain, both of which is impossible. Hence, the process must eventually terminate.

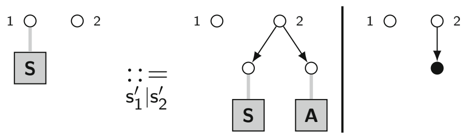
By the fact that $\mathscr{R} \subseteq \mathscr{R}_1 \subseteq \cdots \subseteq \mathscr{R}_{i_0}$ and the *if* direction of Fact 3.13, the contextual grammar $\Gamma_{i_0} = \langle \mathscr{C}, \mathscr{R}_{i_0}, Z\rangle$ is equivalent to Γ. It remains to be shown that every derivation in Γ_{i_0} can be turned into a derivation that does not make use of chain rules, i.e., into a derivation in Γ'. By Lemma 3.9 and an obvious induction on the number of steps using chain rules, it suffices to verify that $G \Rightarrow_{rr'} H$ implies $G \Rightarrow_{\mathscr{R}'} H$, for every chain rule $r \in \mathscr{R}$ and every rule $r' \in \mathscr{R}_{i_0}$ that is not a chain rule. Fact 3.13 yields a rule $p \in r\,;r'$ such that $G \Rightarrow_p H$. By Observation 3.14 we can assume that p is a minimal element of $r\,;r'$. However, this means that there exists a rule $q \in \mathscr{R}_{i_0}$ such that $q \sqsubseteq p$ (because otherwise the construction would not have terminated at step i_0). Again using Observation 3.14, $G \Rightarrow_p H$ and $q \sqsubseteq p$ implies that $G \Rightarrow_q H$, which finishes the proof. □

Example 3.18 (*Removal of chain rules*) Consider the (incomplete) contextual grammar with the following rules:

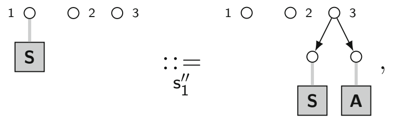


where $\circ$, $\bullet$, and $\otimes$ indicate three distinct node labels. The rules for the variable name A will be added in Example 3.23; they should generate $\otimes$-labeled nodes in particular. The variable S generates a tree having a unique leaf labeled with $\bullet$ if only rules s_1, s_2, and s_3 are used, provided that A generates trees over $\circ$. Note that trees need not be binary, because the S-labeled variable may "jump back" to any node that has been previously generated by rule s_3. If there happens to be a $\otimes$-labeled node at some point (presumably be generated by A), then the generation may "spread out" to this node by means of rule s_4.

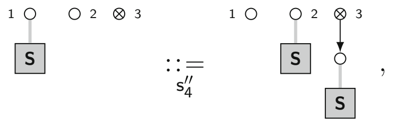
Rule s_3 is a chain rule. Composing it with the rules s_1 and s_2 yields rules



Another composition with, e.g., s'_1 yields a rule



but $s'_1 \sqsubseteq s''_1$, which means that this rule is not needed: whenerver s''_1 applies to a graph, s'_1 does apply as well, with the same result. Similarly, composing s_3 with itself yields only a rule s'_3 such that $s_3 \sqsubseteq s'_3$. However, composing s_3 with s_4 yields the rule



which is not subsumed by another rule. Further iteration does not yield more rules to be included, which means that rule s_3 can now be removed. □

Theorem 3.19 *Contextual grammars with neither empty nor chain rules are a normal form of those contextual grammars that do not generate the empty graph.*

Proof Use Lemma 3.8 followed by Lemma 3.17. Obviously, if applied to a grammar without empty rules, the construction used to prove Lemma 3.17 does not create empty rules. □

Note that it seems that Theorem 3.19 cannot be combined with Lemma 3.10, because the proof of the latter creates chain rules.

3.4 Reducedness

In the case of context-free grammars (both on strings and graphs), reducedness is a very useful property. As we shall show, even contextual grammars can effectively be reduced without affecting their generated language. In context-free grammars, reducedness usually refers to the usefulness of all nonterminals (which correspond to our variable labels), where a nonterminal is useful if it occurs in at least one terminating derivation. In the case of contextual grammars, it is more appropriate to speak about the usefulness of rules rather than of variable labels, because two rules with the same lhs label do not necessarily match the same graphs. Thus, even if all variable labels are useful in the sense mentioned above, the grammar could contain rules that can never be applied.

Definition 3.20 (*Reduced Contextual Grammar*) In $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$, a rule $r \in \mathscr{R}$ is *useful* if there is a derivation of the form $Z \Rightarrow^*_{\mathscr{R}} G \Rightarrow_r G' \Rightarrow^*_{\mathscr{R}} H$ such that $H \in \mathscr{G}_{\mathscr{C} \setminus X}$. Γ is *reduced* if every rule in $\mathscr{R}$ is useful.

In order to prove that reducedness can always be achieved, we need a lemma regarding the following notion of *derivation tree skeletons* of a contextual grammar $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$. Intuitively, these derivation tree skeletons are derivation trees that are obtained by forgetting about the contextual nodes of rules. For the formal definition, let us order the variables in Z and in the right-hand side of every rule in $\mathscr{R}$ in an arbitrary but fixed way. Then the recursive definition of derivation tree skeletons reads as follows:

- If $var(Z) = \{x_1, \ldots, x_k\}$ then $Z[\bar{\ell}_Z(x_1), \ldots, \bar{\ell}_Z(x_k)]$, the tree consisting of a root labeled with Z and ordered children labeled with $\bar{\ell}_Z(x_1), \ldots, \bar{\ell}_Z(x_k)$, is a derivation tree skeleton of Γ.
- If $r = (L, R)$ is a rule in $\mathscr{R}$ with $var(R) = \{x_1, \ldots, x_k\}$, and t is a derivation tree skeleton of Γ that contains a leaf labeled with the lhs label of r, then the tree t' obtained from t by replacing this leaf by the subtree $r[\bar{\ell}_R(x_1), \ldots, \bar{\ell}_R(x_k)]$ is a derivation tree skeleton of Γ.

Thus, derivation tree skeletons of Γ are defined as (one may define them) in the context-free case; see, e.g., [14, Section II.3] and [5, Section 2.3.2]. A node of a derivation tree skeleton is *internal* if its label is a rule. Given a derivation D, we denote its derivation tree skeleton by t_D. We omit the explicit formal definition of t_D here, because it should be obvious. Clearly, if D has length n then t_D has n internal nodes. Note that, in contrast to the contex-free case, t_D does not uniquely determine the graph derived by D, because it lacks information about the matching of contextual nodes. In fact, a derivation tree skeleton may not correspond to any valid derivation at all. However, in the context-safe case this is always guaranteed. To state this result in a proper way, let us say that a derivation tree skeleton is *terminal* if all nodes except the root are internal ones.

Lemma 3.21 *If Γ is a context-safe contextual grammar then each terminal derivation tree skeleton of Γ is the derivation tree skeleton of at least one derivation in Γ.*

Proof Suppose t is a terminal derivation tree skeleton of Γ, and let us say that the lhs label of a proper subtree of t is the lhs label of the rule labeling its root. A derivation tree skeleton t' is a *predecessor of t* if it can be obtained from t by replacing any number of proper subtrees of t by their lhs labels. Now, assume that t contains $n + 1$ nodes (i.e., n internal nodes plus the root). To prove the claim, we show by induction on $i \leqslant n$ that there is a derivation D of length i such that t_D is a predecessor of t. This proves the claim because, for $i = n$, t_D has n internal nodes and is thus equal to t.

For $i = 0$, note that the derivation tree skeleton of the derivation of length 0 is $Z[\bar{\ell}_Z(x_1), \ldots, \bar{\ell}_Z(x_k)]$, which is a predecessor of t. For $1 \leqslant i \leqslant n$, let D be a derivation of length $i - 1$ such that t_D is a predecessor of t, and let G be the graph derived by D. The variables $x_1, \ldots, x_m$ in G correspond to the leaves of t_D that are labeled with $\bar{\ell}_G(x_1), \ldots, \bar{\ell}_G(x_m)$. These nodes are internal nodes in t (since t is terminal), and labeled with rules $r_1, \ldots, r_m$ whose lhs labels are $\bar{\ell}_G(x_1), \ldots, \bar{\ell}_G(x_m)$. Let ass be the rule assignment for G given by $ass(x_i) = r_i$. Since Γ is context-safe, one of the rules r_i applies to the corresponding variable x_i. Hence, D can be prolonged by a step $G \Rightarrow_{r_i} G'$ replacing x_i. By construction, the derivation tree skeleton of the resulting derivation D' is a predecessor of t (obtained by turning the node labeled $\bar{\ell}_G(x_i)$ into an internal node labeled r_i). This proves the lemma. □

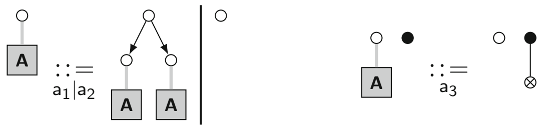
We can now show how reducedness can be achieved.

Theorem 3.22 *Reduced contextual grammars are a normal form of contextual grammars.*

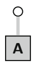
Proof For a contextual grammar $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$, let $\lfloor \Gamma \rfloor$ denote Γ with all useless rules removed. Clearly, $\lfloor \Gamma \rfloor$ is reduced and $\mathscr{L}(\Gamma) = \mathscr{L}(\lfloor \Gamma \rfloor)$. Hence, by Theorem 3.3 it suffices to prove that the set of useful rules of a context-safe contextual grammar $\Gamma = \langle \mathscr{C}, \mathscr{R}, Z \rangle$ can effectively be determined.

By Lemma 3.21, a rule in Γ is useful if and only if it appears in a terminal derivation tree skeleton of Γ. Since contextual nodes are irrelevant for the definition of derivation tree skeletons, this means that usefulness of rules can be checked in the same way as in the context-free case (after applying Theorem 3.3 to make Γ context-safe). More precisely, we can turn Γ into a hyperedge-replacement grammar Γ' by replacing every rule (L, R) by (L', R), where L' is obtained from L by removing all contextual nodes. (Thus, their counterparts in R become nodes that are generated by the rule rather than being taken from the context.) Clearly, Γ' has the same derivation tree skeletons as Γ, which means that usefulness can be checked as in the context-free case. □

Example 3.23 (*Removal of useless rules*) Let us now add the following rules for A to the rules in Example 3.18:

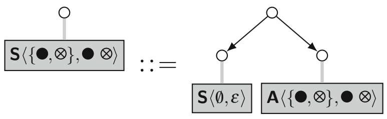


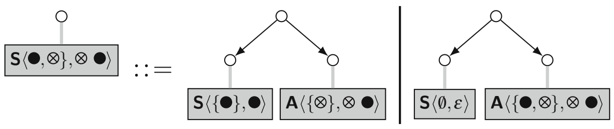
Thus rule a_3 generates the ⊗-node that can be used as a context by rule s_4 of Example 3.18. Note, however, that is enabled only if there is a node •, and s_2 is the only rule creating such a node. Hence there will not be any variable named S left when rule a_3 is applicable. The conclusion is that s_4 is, in fact, useless. There are, however, terminal derivation tree skeletons containing occurrences of s_4, the smallest example being

$Z[\mathsf{s}_4[\mathsf{s}_2[\,], \mathsf{s}_2[\,]]]$ where $Z =$ 

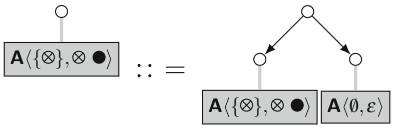
This is because the grammar is not context-safe: We can assign s_4 to the unique variable of Z although is not applicable to it.

Turning the grammar into the context-safe form remedies this deficiency. Consider the annotation of S to be $\langle\{\bullet,\otimes\},\bullet\otimes\rangle$ or $\langle\{\bullet,\otimes\},\otimes\bullet\rangle$ (because it suffices to consider the labels • and ⊗ of contextual nodes in the annotations). All annotated variants of s_4 have a lhs label $\mathsf{S}\langle M, sq\rangle$ such that $[sq] \in \{\varepsilon, \bullet\}$. Such a variable can only be generated by three rules, namely these:





Here, the first and the third rule cannot appear in any terminal derivation tree skeleton, because no rule for A generates a •-labeled node, which means that the annotation makes termination impossible. However, if a derivation tree skeleton contains the second rule, it cannot be terminated either, since the only rule that has the lhs label $\mathsf{A}\langle\{\otimes\},\otimes\bullet\rangle$ is


$\mathbf{A}\langle\{\otimes\},\otimes\bullet\rangle$
::=
$\mathbf{A}\langle\{\otimes\},\otimes\bullet\rangle$
$\mathbf{A}\langle\emptyset,\varepsilon\rangle$

In particular, a_3 does not have an annotated variant with this lhs label, because of the contextual node labeled with •.

Therefore, none of the annotated variants of s_4 appears in a terminal derivation tree skeleton; they are all correctly identified as being useless. □

Example 3.24 (*Reducedness of the program graph grammar*) In the context-safe form the program graph grammar discussed in Example 3.4, the following augmentations shown in Table 1 are useless:

- 4-5 of rules hy, hy*, and cl*.
- 3 of rule si.
- 2-7 of rule bo^1.
- 2-17 of rules bo* and ar*.
- 2 of rules as and ca.

Thus 69 of the 113 rules of the context-safe form are useful. Note that, for each of the original program graph rules, at least one of its context-safe variants is useful. Hence, since the relation between the two grammars is the one expressed in Corollary 3.5, each of the rules of the original grammar is useful. □

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

Corollary 3.25 *For a contextual grammar* Γ*, it is decidable whether* $\mathscr{L}(\Gamma) = \emptyset$*.*

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 3.8: as in the context-free case, the languages generated by contextual grammars are in NP, and there are NP-complete ones among them.

Theorem 4.1 *For every contextual grammar* Γ*, it holds that* $\mathscr{L}(\Gamma) \in NP$*. Moreover, there is a contextual grammar* Γ *such that* $\mathscr{L}(\Gamma)$ *is NP-complete.*

Proof The second part follows from the well-known fact that there are NP-complete hyperedge-replacement languages [21]. For the first part, by Lemma 3.8, it may be assumed that Γ contains neither empty 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 result 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 3.8 may, in the worst case, lead to an exponential size increase of Γ.

For the next result, also known from the theory of hyperedge-replacement languages, let us recall the standard notions of Parikh images and semilinearity. In the following, let us assume that our alphabets $\mathscr{C}$ are implicitly provided with an arbitrary but fixed order $a_1, \ldots, a_k$ (where $\mathscr{C} = \{a_1, \ldots, a_k\}$). Given a graph $G \in \mathscr{G}_{\mathscr{C}}$, its *Parikh image* is the k-tuple $\psi(G) = (n_1, \ldots, n_k)$ such that n_i is the number of nodes or edges in G whose label is a_i. Thus ψ simply counts the numbers of occurrences of the different labels in G. As usual, the Parikh image of a graph language $L \subseteq \mathscr{G}_{\mathscr{C}}$ is $\psi(L) = \{\psi(G) \mid G \in L\}$.

A subset Ξ of $\mathbb{N}^k$ is *linear* if there are $\xi, \xi_1, \ldots, \xi_m \in \mathbb{N}^k$ such that

$$\Xi = \left\{ \xi + \sum_{i=1}^{m} s_i \xi_i \mid s_1, \ldots, s_m \in \mathbb{N} \right\}$$

(using scalar multiplication and componentwise addition), and it is *semilinear* if it is a finite union of linear sets. The famous theorem by Parikh [23] states that the Parikh image of a context-free language (with respect to the terminal alphabet of the grammar) is semilinear. It is known (and not very difficult to see, using Parikh's theorem) that this holds for hyperedge-replacement languages as well [5,14]. We can now extend this result to the contextual case.

Theorem 4.2 *For every language $L \in$ CHR, the Parikh image $\psi(L)$ is semilinear.*

Proof To prove the theorem, it suffices to show that $\psi'(L)$ is semi-linear, where ψ' is defined like ψ but counts only edge labels. To see this, notice that we can attach a single edge labeled $\tilde{a}$ to every node labeled a when this node is generated. Then counting nodes labeled a is the same as counting edges labeled $\tilde{a}$.

Now, we use Lemma 3.21, in a way similar to the proof of Theorem 3.22. By Theorem 3.3, we may assume that Γ is context safe. Now, consider again the hyperedge-replacement grammar Γ' obtained from Γ by removing all contextual nodes from the left-hand sides of its rules. Again, the derivation tree skeletons of Γ' and Γ coincide. Hence, in particular, $\psi'(\mathscr{L}(\Gamma')) = \psi'(\mathscr{L}(\Gamma))$. As mentioned above, it is known that the Parikh image of hyperedge-replacement languages is semilinear (see, e.g., [14]), which completes the proof. □

As a well-known, weaker but sometimes useful consequence, we obtain the result that the size of graphs in $\mathscr{L}(\Gamma)$ grows only linearly, no matter whether we count nodes, edges, or both.

Corollary 4.3 *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 $\mathscr{L}(\Gamma) = \{H_1, H_2, \ldots\}$ with $|H_1| \leqslant |H_2| \leqslant \cdots$, there is a constant k such that $|H_{i+1}| - |H_i| \leqslant k$ for all $i \in \mathbb{N}$.*

The corollary shows, for example, that the language of all complete graphs cannot be generated by any contextual grammar (because the number of edges grows quadratically in this case).

Corollary 4.4 *The language of all complete graphs is not in CHR.*

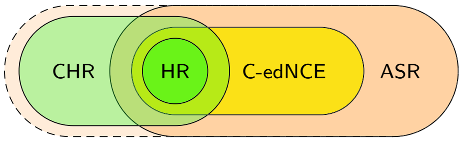


Fig. 11 Relation of languages generated by hyperedge replacement (HR), contextual hyperedge replacement (CHR), adaptive star replacement (ASR), and confluent node replacement (C-edNCE)

Note that the set of all complete graphs is well known to be a C-edNCE graph language. Hence, together with Observation 2.5, the corollary shows that contextual grammars and C-edNCE grammars are incomparable with respect to generative power.

5 Conclusions

In this paper we have studied grammatical and algorithmic properties of contextual grammars. It turned out that 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. In particular, contextual grammars have useful normal forms, namely rules with at most one contextual node, grammars without empty and chain rules, and reduced grammars. Indeed, they share certain algorithmic properties with context-free grammars, (i.e., effective removability of empty and chain rules, decidability of reducedness and emptiness, as well as an NP-complete membership problem) and their languages exhibit at most linear growth. Nevertheless, contextual grammars are more powerful than context-free ones, as illustrated in Fig. 11. Let HR, C-edNCE, ASR, and CHR denote the classes of graph languages generated by hyperedge replacement [14], confluent node replacement [13], adaptive star replacement [6], and contextual hyperedge replacement, respectively. HR is properly included in C-edNCE [13, Section 4.3], as is C-edNCE in ASR [6, Corollary 4.9]. The proper inclusion of HR in CHR and the non-inclusion of CHR in C-edNCE are stated in Observation 2.5. As C-edNCE does contain the language of all complete graphs, this and Corollary 4.4 show that CHR and C-edNCE are uncomparable. [5] We do not yet know whether ASR includes CHR. Example 2.6 indicates that contextual grammars allow for a finer definition of structural properties of models than class diagrams.

5.1 Related work

Some work related to the concepts studied in this paper shall be mentioned here. Context-exploiting rules [9] correspond to contextual rules with a positive application condition, and are equivalent to the context-embedding rules used to define diagram languages in DiaGen [22]. The context-sensitive hypergraph grammars discussed in [14, ChapterVIII] 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* of Bakewell et al. [2] are based on standard graph transformation rules (as considered in [12], but without application conditions),

[5] As Dirk Janssens has pointed out in private communication, this is not sure for the relation of CHR and non-confluent edNCE grammars, which do derive the language of all graphs, thanks to their ability to delete terminal edges during derivations. Then the node replacement grammar generating complete graphs can be extended by (non-confluent) rules that delete arbitrary sets of edges.

and require the specifications to define confluent and terminating reductions. As these rules may delete previously generated terminal nodes and edges, it is difficult to compare them to contextual grammars with respect to generative power. It is clear, however, that all example languages considered in this paper can be defined by graph reduction specifications as well.

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 the operations on the structures. Several logical formalisms have been proposed for this purpose [25]. The graph grammars mentioned in the last paragraph can also be used for defining admissable shapes of graphs, and for analyzing whether graph transformations do preserve shapes. In their paper mentioned above, A. Bakewell, D. Plump, and C. Runciman have studied whether standard graph transformation rules do preserve shapes defined by graph reduction specifications. For more expressive graph transformation rules wherein variables are placeholders for graphs of varying size and shape, conditions for shape-preservation have been studied for the case where the shape of the host graphs, of the pattern and replacements graphs of rules, and of the substitutions of the variables are defined by context-free grammars [17], and by adaptive star grammars [8], respectively.

5.2 Future work

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 [15] that has been started in [20]. Furthermore, we aim at an improved parsing algorithm for contextual grammars that are unambiguous modulo associativity and commutativity of certain replicative rules. And, we want to study shape analysis for the even more expressive graph transformation rules implemented in the GRGEN rewriting tool [3], which can be recursively refined by applying contextual meta-rules [19]. We hope that we can then show that refactoring operations on program graphs preserve the shape defined in the contextual grammar of Example 2.7.

Acknowledgments We wish to thank several people: Annegret Habel for numerous useful comments on the contents of previous versions of this paper, Mark Minas for extensive discussions on many aspects of this work, in particular for his promise to re-join us when parsing of contextual grammars shall be considered at depth, and the two anonymous referees who read the manuscript carefully and provided various useful comments.

References

1. Abadi, M., Cardelli, L.: A Theory of Objects. Monographs in Computer Science. Springer, New York (1996)
2. Bakewell, A., Plump, D., Runciman, C.: Specifying pointer structures by graph reduction. In: Nagl, M., Pfaltz, J., Böhlen, B. (eds.) Applications of Graph Transformation with Industrial Relevance (AGTIVE'03), No. 3062 in Lecture Notes in Computer Science, pp. 30–44. Springer, Berlin (2004)
3. Blomer, J., Geiß, R.: GrGen.net: A generative system for graph-rewriting, user manual. www.grgen.net (2006–2011)
4. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic—A Language-Theoretic Approach, Encyclopedia of Mathematics and Its Applications, vol. 138. Cambridge University Press, Cambridge (2012)
5. Drewes, F., Habel, A., Kreowski, H.J.: Hyperedge replacement graph grammars. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation Vol. I: Foundations, chap. 2, pp. 95–162. World Scientific, Singapore (1997)
6. Drewes, F., Hoffmann, B., Janssens, D., Minas, M.: Adaptive star grammars and their languages. Theor. Comput. Sci. **411**, 3090–3109 (2010)

7. Drewes, F., Hoffmann, B., Janssens, D., Minas, M., Van Eetvelde, N.: Adaptive star grammars. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) 3rd International Conference on Graph Transformation (ICGT'06), No. 4178 in Lecture Notes in Computer Science, pp. 77–91. Springer, Berlin (2006)
8. Drewes, F., Hoffmann, B., Janssens, D., Minas, M., Van Eetvelde, N.: Shaped generic graph transformation. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) Applications of Graph Transformation with Industrial Relevance (AGTIVE'07), No. 5088 in Lecture Notes in Computer Science, pp. 201–216. Springer, Berlin (2008)
9. Drewes, F., Hoffmann, B., Minas, M.: Context-exploiting shapes for diagram transformation. Mach. Graph. Vis. **12**(1), 117–132 (2003)
10. Drewes, F., Hoffmann, B., Minas, M.: Adaptive star grammars for graph models. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) 4th International Conference on Graph Transformation (ICGT'08), No. 5214 in Lecture Notes in Computer Science, pp. 201–216. Springer, Berlin (2008)
11. Drewes, F., Hoffmann, B., Minas, M.: Contextual hyperedge replacement. In: Schürr, A., Varró, D., Varró, G. (eds.) Proceedings of Applications of Graph Transformation with Industrial Relevance 2011 (AGTIVE 2011), No. 7233 in Lecture Notes in Computer Science, pp. 182–197. Springer, Berlin (2012)
12. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, EATCS Monographs on Theoretical Computer Science (2006)
13. Engelfriet, J.: Context-free graph grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3: Beyond Words, chap. 3, pp. 125–213. Springer, Berlin (1999)
14. Habel, A.: Hyperedge Replacement: Grammars and Languages. No. 643 in Lecture Notes in Computer Science. Springer, Berlin (1992)
15. Habel, A., Radke, H.: Expressiveness of graph conditions with variables. In: Electronic Communications of the EASST 30 (2010). International Colloquium on Graph and Model Transformation (GraMoT'10)
16. Harary, F.: Graph Theory. Addison-Wesley, Reading, MA (1969)
17. Hoffmann, B.: Shapely hierarchical graph transformation. In: Proceedings of IEEE Symposia on Human-Centric Computing Languages and Environments, pp. 30–37. IEEE Computer Press, Silver Spring, MD (2001)
18. Hoffmann, B.: Conditional adaptive star grammars. Electronic Communications of the EASST 26 (2010)
19. Hoffmann, B.: More on graph rewriting with contextual refinement. In: A. Habel, M. Mosbah, R. Echahed (eds.) Fifth International Workshop on Graph Computational Models (GCM 2014) (2014). Downloadable at http://gcm2014.imag.fr/proceedingsGCM2014.pdf
20. Hoffmann, B., Minas, M.: Defining models—meta models versus graph grammars. Elect. Comm. of the EASST 29 (2010). In Proceedings of 6th Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'10), Paphos, Cyprus
21. Lange, K.J., Welzl, E.: String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. Discrete Appl. Math. **16**, 17–30 (1987)
22. Minas, M.: Concepts and realization of a diagram editor generator based on hypergraph transformation. Sci. Comput. Program. **44**(2), 157–180 (2002)
23. Parikh, R.J.: On context-free languages. J. ACM **13**, 570–581 (1966)
24. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Object Technology Series, 2nd edn. Addison Wesley, Reading, MA (2004)
25. Sagiv, M., Reps, T., Wilhelm, R.: Solving shape-analysis problems in languages with destructive updating. ACM Trans. Program. Lang. Syst. **20**(1), 1–50 (1998)
26. Van Eetvelde, N.: A graph transformation approach to refactoring. Doctoral thesis, Universiteit Antwerpen (2007)