HYPEREDGE REPLACEMENT GRAPH GRAMMARS

FRANK DREWES, HANS-JÖRG KREOWSKI
Universität Bremen, Fachbereich 3, D–28334 Bremen (Germany)
E-mail: {drewes,kreo}@informatik.uni-bremen.de

ANNEGRET HABEL
Universität Hildesheim, D–31141 Hildesheim (Germany)
E-mail: habel@informatik.uni-hildesheim.de

In this survey the concept of hyperedge replacement is presented as an elementary approach to graph and hypergraph generation. In particular, hyperedge replacement graph grammars are discussed as a (hyper)graph-grammatical counterpart to context-free string grammars. To cover a large part of the theory of hyperedge replacement, structural properties and decision problems, including the membership problem, are addressed.

Contents

1 Introduction

Graph grammars have been developed as an extension of the concept of formal grammars on strings to grammars on graphs. Among string grammars, context-free grammars have proved extremely useful in practical applications and powerful enough to generate a wide spectrum of interesting formal languages. It is therefore not surprising that analogous notions have been developed also for graph grammars. Corresponding to the different graph-grammar formalisms in the literature, and to the differing opinions about what the term "context-free" means, a number of different types of graph grammars have been given this attribute (cf. Feder [43], Pavlidis [78], Della Vigna and Ghezzi [19], Janssens and Rozenberg [61], Slisenko [81], Bauderon and Courcelle [7], Habel and Kreowski [50,51], Montanari and Rossi [76], Lautemann [67], Engelfriet [29,30], and Lengauer and Wanke [72]). Among the most general, there are the C-edNCE graph grammars surveyed in Chapter [38] and the hyperedge replacement grammars dealt with in this chapter.

Hyperedge replacement is an elementary approach of graph and hypergraph rewriting. It was introduced in the early seventies by Feder [43] and Pavlidis [78] (under other names) and has been intensively studied since the late seventies (starting with the special case of edge replacement) by Bauderon, Courcelle, and Engelfriet [7,10,11,12,14], Engelfriet, Heyker, Leih, and Rozenberg [37,33,34,31,36,32], Drewes, Habel, Kreowski, Lautemann, and Vogler [62,63,50,52,67,56,53,68,57,69,20,21,55], Lengauer and Wanke [70,71,72], and others.

A *hyperedge* is an atomic item with a fixed number of tentacles, called the *type* of the hyperedge. It can be attached to any kind of structure coming with a set of nodes by attaching each of its tentacles to a node. The hyperedge controls the sequence of these *attachment nodes* and can play the role of a place holder, which may be replaced with some other structure eventually. In such a *hyperedge replacement*, the hyperedge is removed, and the replacing structure R is embedded into the original structure. For this purpose, the considered structures are equipped with sequences of *external nodes*. R is then glued to the remainder of the original structure by fusing each external node with the corresponding attachment node. For this, the number of external nodes in R is required to equal the type of the replaced hyperedge. Thus, referring to the distinction between *gluing* and *connecting* approaches discussed in the introduction of Chapter [38], hyperedge replacement belongs to the gluing approaches.

The replacement of a hyperedge by some structure can be iterated if the original structure or the replacing one is equipped with further hyperedges. If the hyperedges are labelled we may define *productions*. They consist of a label as left-hand side and a replacing structure as right-hand side. Hyperedge replacement grammars mainly consist of a start structure and a finite set of such productions. If a hyperedge labelled with the left-hand side of a production is replaced with the right-hand side this is called a *direct derivation*. If such a grammar, besides the set of productions and the start structure, provides a specification of terminal structures, it can generate a language: the set of terminal structures derivable from the start structure.

The aim of this survey is to present the theory of hyperedge replacement graph grammars. To keep the technicalities as simple as possible, we deal with hypergraphs that consist of sets of nodes and sets of hyperedges as described above, rather than with hybrid objects where some underlying structures are equipped with hyperedges in addition. Hypergraphs are general and flexible enough to cover many interesting cases. In particular, the ordinary directed graphs are special hypergraphs because hyperedges of type 2 are just directed edges. Clearly, undirected graphs can also be handled as a special case.

If a hyperedge is replaced its context is not affected. Therefore, hyperedge replacement provides a context-free type of rewriting (as long as no additional application conditions are employed). This is the main reason for the fact that several structural results for hyperedge replacement grammars and languages are quite similar to the properties of context-free string grammars and languages and that many interesting problems turn out to be decidable. In this survey we try to cover some typical parts of the theory of hyperedge replacement. In particular, we consider structural properties and decision problems. It turns out that the generative power is increasing with the type of hyperedges involved in the derivation process even if one wants to generate only graphs or even string graphs. The generative power of hyperedge replacement grammars generating string graphs can be characterized in terms of tree-walking transducers.

Concerning decision problems we discuss two types of questions. First, one can ask whether the hypergraphs generated by a given hyperedge replacement grammar satisfy a property π of interest. This question—and related ones—are of course interesting for any kind of language generating device since the languages are usually infinite sets. In the case of hyperedge replacement grammars one can find a large class of such properties π for which the mentioned question is decidable (closely related results are discussed in Section 6 of Chapter [13]). The second decision problem we are going to address is the classical membership problem: Does a given hypergraph belong to the considered language? As the reader probably expects, the problem is decidable for hyperedge replacement languages. However, there is an important difference to context-free string grammars: The membership problem turns out to be NP-complete. Only restricted subclasses lead to polynomial membership algorithms.

Two further, equally important parts of the theory are left out here. They can be found in other chapters of this volume. The relation of hyperedge replacement and node replacement is discussed in Sections 3.3 and 3.4 of Chapter [38] and in Section 5 of Chapter [13]. Furthermore, monadic second-order logic on graphs, its relation with hyperedge replacement, and its use in connection with hyperedge replacement is presented in Chapter [13] (see in particular Sections 5–7 of that chapter).

The introduction above should have made clear that the chapter presents an overview of the *theory* of hyperedge replacement graph grammars rather than of their practical use. However, it must be pointed out that hyperedge replacement is not only a concept of mathematical beauty, but is useful also from a practical point of view. This is no surprise at all. Graphs are used successfully in all branches of computer science, and context-free generation mechanisms of formal languages are not less important. This opens a wide area of potential applications. Just to mention some examples, hyperedge replacement can be used to model and support the design of VLSI circuits (see, for example, [72,84] by Lengauer and Wanke), and it has been used in a generator for diagram editors to describe the context-free part of the syntax of diagrams (see [74,75,82] by Minas and Viehstaedt). An early application to the generation of semi-structured control-flow diagrams was given by Farrow, Kennedy, and Zucconi [42] (see also Example 2.2). In general, wherever classes of graphs or hypergraphs are to be specified, generated, manipulated, etc. it seems a good idea to distinguish between context-free aspects and non-context-free ones, and to describe the former by a context-free rewriting mechanism like hyperedge or node replacement (see also Chapter [38]).

The survey is organized in the following way. Section 2 provides the basic notions and notations of hyperedge replacement grammars. In Section 3, the context-freeness lemma is presented stating that derivations in hyperedge replacement grammars do not interfere with each other as long as they concern different hyperedges. This is the key result to the major part of the theory of hyperedge replacement as discussed in this chapter. Some structural properties (including a fixed-point theorem, a pumping lemma and a Parikh theorem) are presented in Section 4, while the generative power is considered in Section 5. Sections 6 and 7 are devoted to decision problems. While in Section 7 the membership problem is discussed, Section 6 concerns the decidability of properties of hypergraphs and hypergraph languages. Each of the main sections ends with bibliographic notes and hints to further results.

2 Hyperedge replacement grammars

In this section, we introduce hyperedge replacement grammars as hypergraph manipulating and hypergraph-language generating devices. A simple example may perhaps be useful to illustrate the idea. Let us consider directed graphs with multiple edges having two distinguished nodes *begin* and *end*. We can replace an edge e of a graph G with another graph G' by removing e from G, adding G' disjointly, and fusing the *begin*-node of G' with the source of e and the *end*-node of G' with the target of e. The resulting graph is denoted by $G[e/G']$. An example is given in Figure 1. Note that $G[e/G']$ keeps the *begin*-

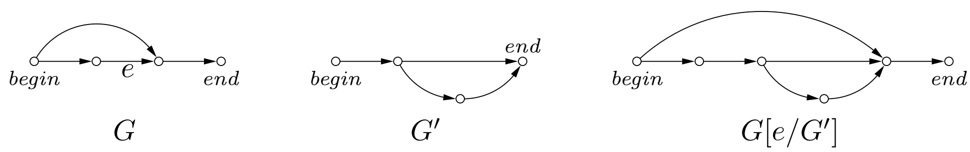

Figure 1: The replacement of an edge.

and *end*-nodes of G.

Now, let us allow in addition to label edges with arbitrary symbols. Then we can build productions whose left-hand sides are labels and whose right-hand sides are graphs. A production $S ::= G'$ is applied to a graph G by choosing an edge e labelled with S and replacing it with G', which yields a *direct derivation* $G \Longrightarrow G[e/G']$. The productions shown in Figure 2, for instance, generate the set of *series-parallel graphs* (which describe a simple type of concurrent processes) if we start with a single, S-labelled edge and apply productions until no S-labelled edge is left. A sample derivation beginning with a slightly larger

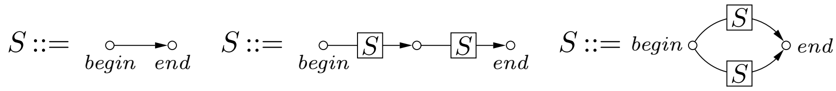

Figure 2: Productions to generate series-parallel graphs.

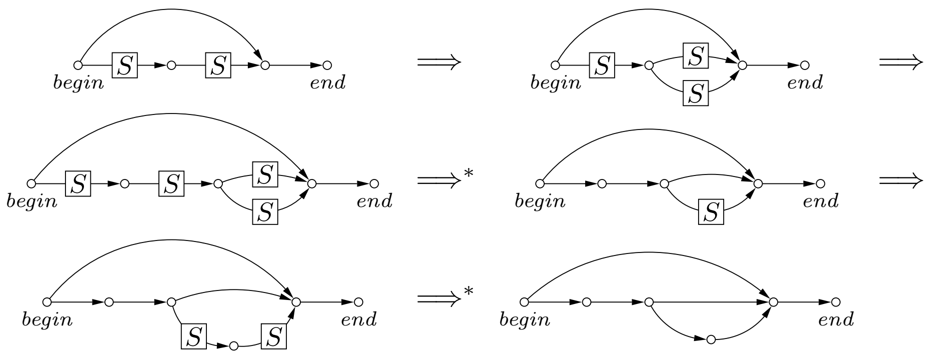

Figure 3: A derivation using the productions of Figure 2.

graph is shown in Figure 3. Here, $\Longrightarrow^*$ denotes the transitive and reflexive closure of $\Longrightarrow$, as usual.

The notion of replacement discussed so far is commonly called *edge replacement*, for obvious reasons. It is a special case of the notion of hyperedge replacement presented in this chapter. In the remainder of this section the approach will be defined formally. The first paragraph provides the basic notions concerning hyperedges and hypergraphs. In the introduced approach, a hyperedge is an atomic item with a label and an ordered set of tentacles. A set of nodes together with a collection of such hyperedges (usually with varying numbers of tentacles) forms a hypergraph if each tentacle is attached to a node. Directed and labelled graphs, undirected and unlabelled graphs as well as strings can be considered as special cases of hypergraphs. For technical reasons, we assume that the labels are typed by non-negative integers in such a way that, for each hyperedge, the type of its label and the number of its tentacles coincide.

Moreover, a hypergraph is equipped with a sequence of external nodes, which is used to construct the replacement of hyperedges by hypergraphs. The external nodes correspond to the nodes *begin* and *end* in the example above. To make the construction simpler, we assume in addition that the attachment nodes of each hyperedge on the one hand and the external nodes on the other hand are pairwise distinct. It must be pointed out, however, that this restriction is no vital one. If it is dropped one can prove normal-form results yielding systems of the type considered here.

Similar to the example presented above, the replacement of some hyperedges of a hypergraph by other hypergraphs yields an expanded hypergraph by removing the chosen hyperedges, adding the replacing hypergraphs and fusing their external nodes with the corresponding attachment nodes of the replaced hyperedges. This notion of hyperedge replacement yields the basic steps in the derivation process of a hyperedge replacement grammar, where the pair of the label of each replaced hyperedge and the replacing hypergraph form a production of the grammar.

2.1 Hypergraphs

By $\mathbb{N}$ we denote the set of all natural numbers, including 0. For a set A, A^* denotes the set of all strings over A, including the empty string λ; $A^+ = A^* - \{\lambda\}$ denotes the set of all strings over A, except the empty string λ. For $w \in A^*$, $|w|$ denotes the *length* of w, $[w]$ denotes the set of all symbols occurring in w, and $w(i)$ denotes the i-th symbol in w, for $1 \leq i \leq |w|$. The free symbolwise extension $f^*\colon A^* \to B^*$ of a mapping $f\colon A \to B$ is defined by $f^*(a_1 \cdots a_k) = f(a_1) \cdots f(a_k)$ for all $k \in \mathbb{N}$ and $a_i \in A$ $(i = 1, \ldots, k)$.

In the following, let C be an arbitrary, but fixed set of *labels* and let $\mathit{type}\colon C \to \mathbb{N}$ be a *typing* function for C. A (hyperedge-labelled, multi-pointed) *hypergraph* H over C is a tuple $(V, E, \mathit{att}, \mathit{lab}, \mathit{ext})$ where V is a finite set of *nodes*, E is a finite set of *hyperedges*, $\mathit{att}\colon E \to V^*$ is a mapping assigning a sequence of pairwise distinct *attachment nodes* $\mathit{att}(e)$ to each $e \in E$, $\mathit{lab}\colon E \to C$ is a mapping that *labels* each hyperedge such that $\mathit{type}(\mathit{lab}(e)) = |\mathit{att}(e)|$, and $\mathit{ext} \in V^*$ is a sequence of pairwise distinct *external nodes*.

The components of a hypergraph H may be denoted by V_H, E_H, att_H, lab_H, ext_H, respectively. Furthermore, given a set $X \subseteq C$ of labels we denote by E_H^X the set $\{e \in E_H \mid \mathit{lab}_H(e) \in X\}$ of hyperedges of H with labels in X. The number of nodes plus the number of hyperedges of H is called the *size* of H, denoted by $|H|$. The class of all hypergraphs over C is denoted by $\mathcal{H}_C$.

If the hypergraph in question is understood, we say that $e \in E$ is an *m-edge* for some $m \in \mathbb{N}$ and m is its *type*, denoted by $\mathit{type}(e)$, if $\mathit{type}(\mathit{lab}(e)) = m$. In order to avoid confusion we may also write $\mathit{type}_H(e)$ if H is the hypergraph referred to. $H \in \mathcal{H}_C$ is an *n-hypergraph* for some $n \in \mathbb{N}$ and n its *type*, denoted by $\mathit{type}(H)$, if $|\mathit{ext}_H| = n$.

The sequence of external nodes may be empty so that ordinary hypergraphs (without external nodes) may be seen as 0-hypergraphs and in this way as special cases of hypergraphs with external nodes. An n-hypergraph H over C is considered as a (directed) *n-graph* if all hyperedges of H are 2-edges.

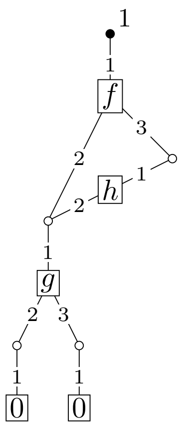

Figure 4: A hypergraph denoting the expression $f(g(0,0), h(g(0,0)))$.

The first node of the attachment nodes of a 2-edge corresponds to the source and the second one to the target. The two external nodes of a 2-graph G are denoted by $begin_G$ and end_G in this order.

As a convention, an *unlabelled* hyperedge is a hyperedge labelled with a special label $\sqcup$ which we do not draw in figures. Subject to this convention, unlabelled graphs and hypergraphs turn out to be special cases of the sort of hypergraphs defined above.

A graph $G = (\{v_0, v_1, \ldots, v_n\}, \{e_1, \ldots, e_n\}, att, lab, v_0\, v_n)$ over C is called a *string graph* if $v_0, v_1, \ldots, v_n$ are pairwise distinct and $att(e_i) = v_{i-1}\, v_i$ for $i = 1, \ldots, n$. If $w = lab(e_1) \cdots lab(e_n)$, then G is called the *string graph induced by* w and is denoted by $w^\bullet$. Such a string graph provides a unique graph representation of the string $w \in C^+$.

In drawings of hypergraphs, a dot ($\bullet$) represents an external node, internal nodes are drawn as circles, and a box depicts a hyperedge with attachment nodes, where the label is inscribed in the box, and the i-th tentacle is attached to the i-th attachment node ($i = 1, \ldots, m$). In other words, the graphical representation makes use of the one-to-one correspondence between hypergraphs and bipartite graphs. As an example, see the 1-hypergraph in Figure 4, which represents a functional expression with sharing. A 2-edge may also be drawn as an arrow pointing from its first attached node to the second.

An m-hypergraph H with m nodes (that is, all nodes are external) and a single hyperedge e is said to be a *handle*[a] if $att_H(e) = ext_H$. If $lab_H(e) = A$, then H is said to be the *handle* induced by A and is denoted by $A^\bullet$.

[a]Note that this notion of *handles* differs from the one used in Section 4.3 of Chapter [38].

Let $H, H' \in \mathcal{H}_\mathcal{C}$. Then H is a *sub-hypergraph* of H', denoted by $H \subseteq H'$, if $V_H \subseteq V_{H'}$, $E_H \subseteq E_{H'}$, $att_H(e) = att_{H'}(e)$, and $lab_H(e) = lab_{H'}(e)$ for all $e \in E_H$. Note that nothing is assumed about the relation of the external nodes. H and H' are *isomorphic* if there is a pair $h = (h_V, h_E)$ of bijective mappings $h_V \colon V_H \to V_{H'}$ and $h_E \colon E_H \to E_{H'}$ with $h_V^*(att_H(e)) = att_{H'}(h_E(e))$ and $lab_H(e) = lab_{H'}(h_E(e))$ for all $e \in E_H$ as well as $h_V^*(ext_H) = ext_{H'}$.

2.2 Hyperedge replacement

Let $H \in \mathcal{H}_\mathcal{C}$ be a hypergraph, $B \subseteq E_H$ be a set of hyperedges to be replaced. Let $repl \colon B \to \mathcal{H}_\mathcal{C}$ be a mapping with $type(repl(e)) = type(e)$ for all $e \in B$. Then the *replacement* of B in H by $repl$ yields the hypergraph $H[repl]$ obtained by removing B from E_H, adding the nodes and hyperedges of $repl(e)$ for each $e \in B$ disjointly and fusing the i-th external node of $repl(e)$ with the i-th attachment node of e for each $e \in B$ and $i = 1, \ldots, type(e)$. All hyperedges keep their labels and attachment nodes; the external nodes of $H[repl]$ are those of H. If $B = \{e_1, \ldots, e_n\}$ and $repl(e_i) = R_i$ for $i = 1, \ldots, n$, then we also write $H[e_1/R_1, \ldots, e_n/R_n]$ instead of $H[repl]$.

If, for each $e \in B$, the replacing hypergraph $repl(e)$ consists of the attachment nodes as external nodes and nothing else, the replacement of B in H removes B and adds nothing. Hence, the result may be denoted by $H - B$ in this case. The replacement of some hyperedges is illustrated in Figure 5.

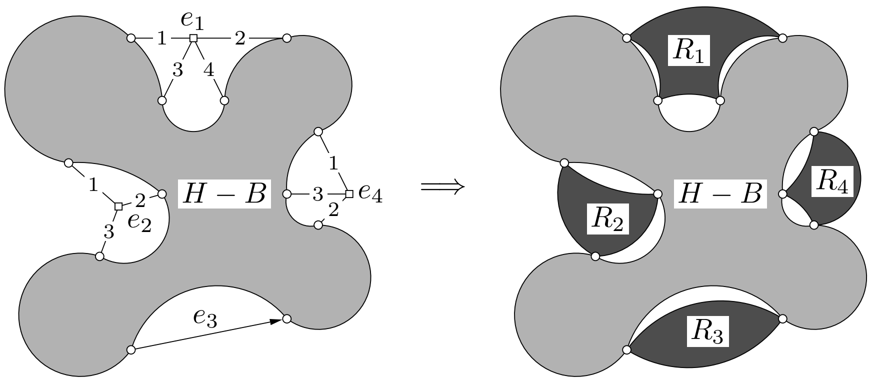

Figure 5: The replacement of (unlabelled) hyperedges $e_1, \ldots, e_4$ with $R_1, \ldots, R_4$.

Note that the result of a hyperedge replacement is defined only up to isomorphism. Therefore, to keep the technicalities as simple as possible, we usually do not distinguish between isomorphic copies of a hypergraph.

Hyperedge replacement enjoys some nice properties well-known from other rule-based formalisms. First of all, we have a sequentialization and parallelization property. It does not matter whether we replace some hyperedges of a hypergraph one after another, or simultaneously. The second property is confluence. Hyperedges of a hypergraph can be replaced in any order, without affecting the result. (In fact, this follows already from the sequentialization and parallelization property. If we replace hyperedges simultaneously there is no order among them at all.) The last and maybe most important property is associativity. If a hyperedge is replaced and afterwards a hyperedge of the new part is replaced with a third hypergraph, the same is obtained by first replacing the latter hyperedge and then replacing the first one with the result. These properties are stated formally below.

Sequentialization and parallelization. Let H be a hypergraph with pairwise distinct $e_1, \dots, e_n \in E_H$ and let H_i be a hypergraph with $\mathit{type}(H_i) = \mathit{type}_H(e_i)$ for $i = 1, \dots, n$. Then

$$H[e_1/H_1, \dots, e_n/H_n] = H[e_1/H_1] \cdots [e_n/H_n].$$

Confluence. Let H be a hypergraph with distinct $e_1, e_2 \in E_H$ and let H_i be a hypergraph with $\mathit{type}(H_i) = \mathit{type}_H(e_i)$ for $i \in \{1, 2\}$. Then

$$H[e_1/H_1][e_2/H_2] = H[e_2/H_2][e_1/H_1].$$

Associativity. Let H, H_1, H_2 be hypergraphs with $e_1 \in E_H$ and $e_2 \in E_{H_1}$, such that $\mathit{type}_H(e_1) = \mathit{type}(H_1)$ and $\mathit{type}_{H_1}(e_2) = \mathit{type}(H_2)$. Then

$$H[e_1/H_1][e_2/H_2] = H[e_1/H_1[e_2/H_2]].$$

2.3 Hyperedge replacement derivations, grammars, and languages

Let $N \subseteq C$ be a set of *nonterminals*. A *production* over N is an ordered pair $p = (A, R)$ with $A \in N$, $R \in \mathcal{H}_C$ and $\mathit{type}(A) = \mathit{type}(R)$. A is called the *left-hand side* of p and is denoted by $\mathit{lhs}(p)$; R is called the *right-hand side* and is denoted by $\mathit{rhs}(p)$.

Let $H \in \mathcal{H}_C$ and let P be a set of productions. Let $e \in E_H$ and $(\mathit{lab}_H(e), R) \in P$. Then H *directly derives* $H' = H[e/R]$. In this case, we write $H \Longrightarrow_P H'$, or just $H \Longrightarrow H'$ if P is clear from the context, and call this a *direct derivation*. Examples of direct derivations have already been discussed in the beginning of this section (see Figure 3).

A sequence of direct derivations $H_0 \Longrightarrow \cdots \Longrightarrow H_k$ is called a *derivation of length k* from H_0 to H_k and is denoted by $H \Longrightarrow_P^* H'$ or $H_0 \Longrightarrow^* H_k$. If the length of the derivation matters, we write $H_0 \Longrightarrow^k H_k$. Additionally, if H'_0 is isomorphic to H_0, we speak of a derivation from H_0 to H'_0 of length 0.

Using the concepts of productions and derivations, hyperedge replacement grammars and languages can be introduced in a straightforward way. This is done below.

Definition 2.1 (*hyperedge replacement grammar*) *A* hyperedge replacement grammar *is a system* $HRG = (N, T, P, S)$ *where* $N \subseteq C$ *is a set of* nonterminals*,* $T \subseteq C$ *with* $T \cap N = \emptyset$ *is a set of* terminals*,* P *is a finite set of* productions *over* N *and* $S \in N$ *is the* start symbol.

The hypergraph language $L(HRG)$ generated by HRG *is* $L_S(HRG)$*, where for all* $A \in N$*,* $L_A(HRG)$ *consists of all hypergraphs in* $\mathcal{H}_\mathcal{T}$ *derivable from* $A^\bullet$ *by applying productions of* P*:*

$$L_A(HRG) = \{H \in \mathcal{H}_\mathcal{T} \mid A^\bullet \overset{*}{\underset{P}{\Longrightarrow}} H\}.$$

We denote the class of all hyperedge replacement grammars by $\mathcal{HRG}$ and the class of all *hyperedge replacement languages* by $\mathcal{HRL}$. A hyperedge replacement grammar $HRG = (N, T, P, S)$ is said to be of *order* r (for some $r \in \mathbb{N}$) if for all $(A, R) \in P$, $type(R) \leq r$. A hyperedge replacement language L is of *order* r (for some $r \in \mathbb{N}$) if there is a hyperedge replacement grammar HRG of order r with $L(HRG) = L$. The classes of all hyperedge replacement grammars and languages of order r are denoted by $\mathcal{HRG}_\triangledown$ and $\mathcal{HRL}_\triangledown$, respectively. A hyperedge replacement grammar $ERG = (N, T, P, S)$ such that all right-hand sides of productions in P are graphs is also called an *edge replacement grammar*. Note that, if given such an edge replacement grammar, one may always assume without loss of generality that all nonterminal labels except perhaps the start symbol have type 2. The class of all edge replacement grammars is denoted by $\mathcal{ERG}$.

Even if one wants to generate graph languages (or string-graph languages) rather than hypergraph languages, one may use nonterminal hyperedges because the generative power of hyperedge replacement grammars increases with their order (see Section 5). By definition of derivations the set $L(HRG)$ is closed under isomorphisms. Moreover, $L(HRG)$ is homogeneous, that is, all its hypergraphs are of the same type. Therefore, non-homogeneous languages and languages not closed under isomorphism cannot be generated by the grammars introduced above.

Example 2.2 (generation of semi-structured control-flow graphs) Control-flow graphs of so-called semi-structured programs are useful for data flow analysis, as shown by Farrow, Kennedy, and Zucconi [42]. These control-flow graphs (seen as hypergraphs) are generated by the hyperedge replacement

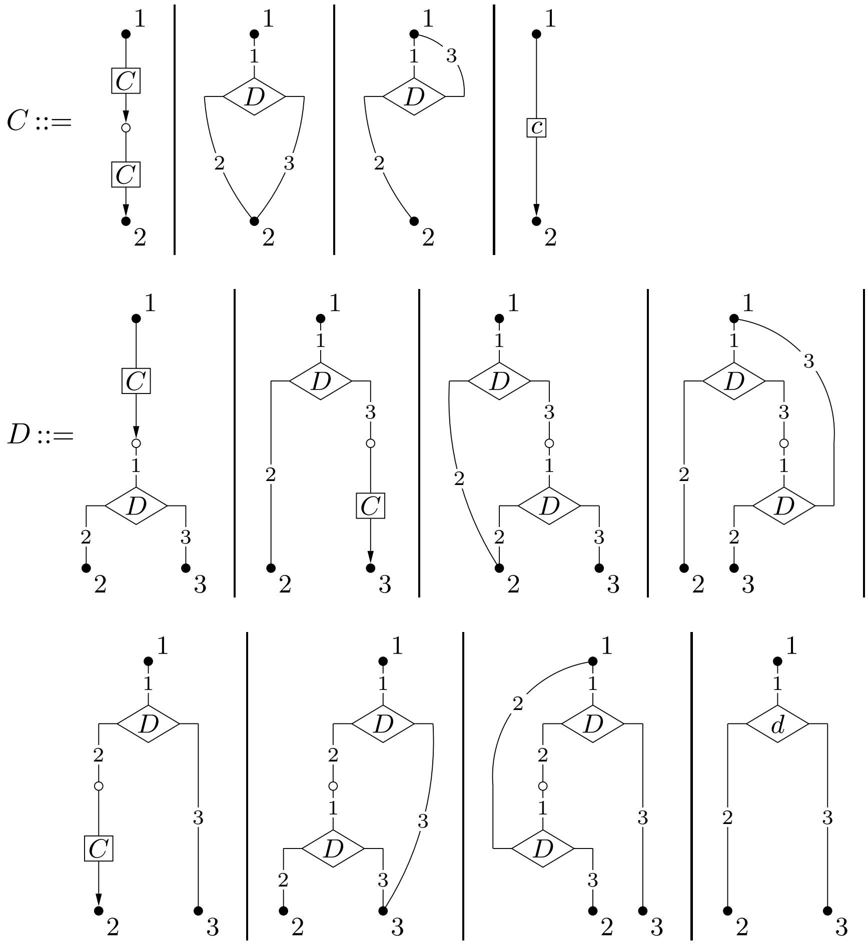

Figure 6: Productions of the hyperedge replacement grammar *FLOW-GRAPHS*.

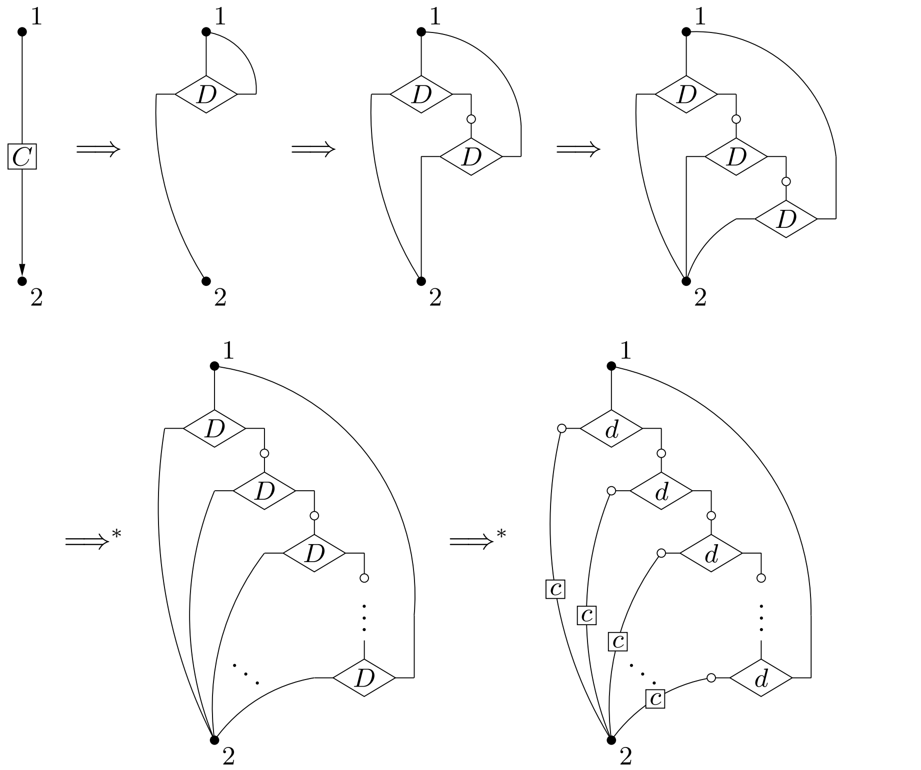

Figure 7: A derivation in the hyperedge replacement grammar *FLOW-GRAPHS*.

grammar *FLOW-GRAPHS* $= (\{C, D\}, \{c, d\}, P, C)$, where P contains the productions given in Figure 6 in a kind of Backus-Naur-Form. (The types of nonterminal labels are those of the corresponding right-hand sides.) A derivation deriving a multi-exit loop is given in Figure 7, where tentacle numbers are omitted.

Example 2.3 (generation of a string-graph language) Let us consider the hyperedge replacement grammar $A^nB^nC^n = (\{S, A\}, \{a, b, c\}, P, S)$ where P consists of the productions depicted in Figure 8. Beginning with $S^\bullet$, the application of the second production yields the string graph $(abc)^\bullet$. By applying the first production, then applying the third production $n - 1$ times, followed by an application of the fourth production, we obtain the derivation in Figure 9. Furthermore, the only hypergraphs in $L(A^nB^nC^n)$ are string graphs of the form $(a^nb^nc^n)^\bullet$ for $n \geq 1$. Thus, $L(A^nB^nC^n) = \{(a^n b^n c^n)^\bullet \mid n \geq 1\}$.

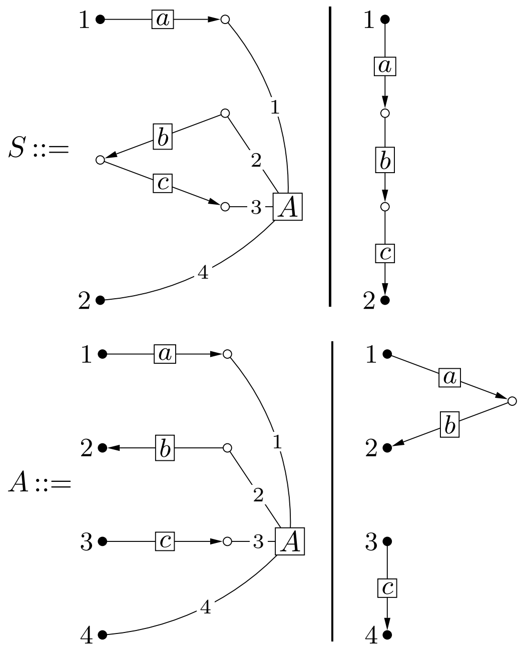

Figure 8: Productions of the hyperedge replacement grammar $A^nB^nC^n$.

2.4 Bibliographic notes

The kind of introduction chosen here is similar to the one in [48]. It must be noted, however, that the notion of direct derivations chosen here is a purely sequential one. A direct derivation replaces only one hyperedge. Instead, one can as well use a notion of parallel direct derivations, where arbitrarily many hyperedges can be replaced in one step. More precisely, we could say $H \Longrightarrow_P H[repl]$ if $repl\colon B \to \mathcal{H}_\mathcal{C}$ is a mapping with $B \subseteq E_H$ and $(lab_H(e), repl(e)) \in P$ for all $e \in B$. Using the sequentialization property of hyperedge replacement quoted above, it is clear that every derivation consisting of this type of parallel direct derivations can be transformed into a derivation using only direct derivations as defined here. In particular, both notions of direct derivations lead to hyperedge replacement grammars of equal generative power.

The hyperedge replacement approach presented here is described on a set-theoretical level. Since hyperedge replacement may be seen as a special case of hypergraph replacement in the double-pushout approach (see [49]) and the double-pushout approach has an algebraic description (see Chap-

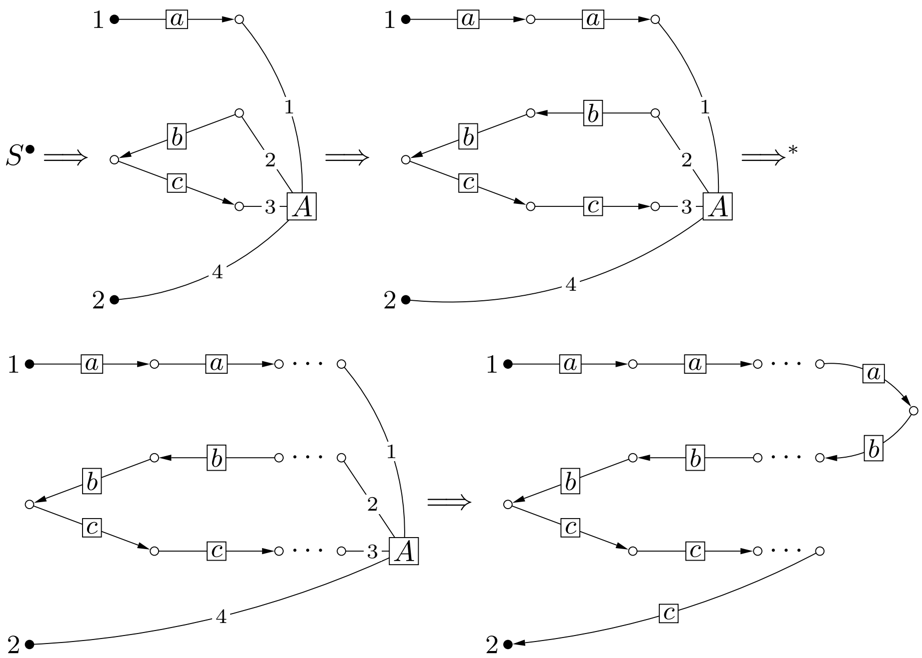

Figure 9: A derivation in $A^nB^nC^n$.

ter [8]), hyperedge replacement has an algebraic description, too. The relation between hyperedge replacement languages, logical definability of hypergraph languages, and recognizability has been intensively studied by Courcelle (see Chapter [13]).

Courcelle, Engelfriet, and Rozenberg [15] studied the notion of separated handle-rewriting hypergraph grammars (see also Section 4.3 of Chapter [38]). These grammars combine the rewriting mechanisms of vertex- and hyperedge replacement graph grammars. If we restrict our attention to the generation of graphs, handle rewriting grammars are as powerful as the C-edNCE graph grammars discussed in Chapter [38] and are thus more powerful than hyperedge replacement grammars. Looking at the more general situation where hypergraphs are generated, hyperedge replacement and handle-rewriting graph grammars turn out to be incomparable, though.

In [32] Engelfriet shows how regular tree grammars can be used to generate graphs, by generating expressions that denote graphs. Top-down and bottom-up tree transducers can then be used as a tool for proving properties of such graph generating tree grammars.

As mentioned in the introduction, the idea of hyperedge replacement can also be applied to structures other than hypergraphs. Considering pictures (that is, subsets of $\mathbb{R}$ for some $d \in \mathbb{N}$) as underlying structures Habel and Kreowski [54] introduced the so-called *collage grammars*. These allow to generate and to study (approximations of) fractal images and other sorts of pictures (see the work by Dassow, Drewes, Habel, Kreowski, and Taubenberger [26,17,27,23]).

3 A context-freeness lemma

In this section, we present a context-freeness lemma for hyperedge replacement grammars and languages. Furthermore, we employ the lemma in order to introduce derivation trees as a helpful and intuitive representation of derivations.

3.1 Context freeness

Most of the results presented in this survey (as, for example, the fixed-point theorem and the pumping lemma) are mainly based on the context-free nature of hyperedge replacement. Suppose we are given any notion of replacement allowing to replace primitive components of certain objects with other such objects. In the string case the objects would be strings and the primitive components would be the individual letters. In the case of hyperedge replacement we have hypergraphs as objects and hyperedges as primitive components. Intuitively, context-freeness means that in a derivation of an object O' from and object O we can consider the primitive components of O separately to derive from them the corresponding parts of O'. More precisely, O derives O' if and only if each nonterminal component x derives an object $repl(x)$ such that O' is the object obtained by replacing each nonterminal x of O with $repl(x)$.

In the case of type-2 string grammars we can derive a string w from $A_1 \cdots A_n$ if and only if $w = repl(A_1) \cdots repl(A_n)$, where each $repl(A_i)$ is a string derivable from A_i. In the case of hyperedge replacement grammars the situation is similar, though not as completely obvious. Intuitively, during a derivation that derives a hypergraph H from a hypergraph R the nonterminal hyperedges of R are turned step by step into larger hypergraphs. Thus, each nonterminal hyperedge of R gives rise to a particular sub-hypergraph of H. The important point is that the applicability of a production depends only on the existence of a hyperedge with the required label, and nothing but this hyperedge is affected by the application of the production. Therefore, if we are given a nonterminal hyperedge $e \in E_R^N$ we may simply forget about the context of e in R, keeping only the handle $lab_R(e)^\bullet$, and restrict the original derivation accordingly. In this way we get for every $e \in E_R^N$ a sub-derivation $lab_R(e)^\bullet \Longrightarrow^* repl(e)$ such that $R[repl] = H$. Clearly, the converse is also true: If we are given the hypergraphs $repl(e)$ with $lab_R(e)^\bullet \Longrightarrow^* repl(e)$ $(e \in E_R^N)$ then the hypergraph $R[repl]$ is derivable from R. This property justifies to call hyperedge replacement grammars context-free.

Below, we state the result in a recursive version especially suitable for inductive proofs. If a derivation starts in a handle $A^\bullet$ it must necessarily have the form $A^\bullet \Longrightarrow R \Longrightarrow^* H$, where (A, R) is a production. The remainder $R \Longrightarrow^* H$ can now be decomposed as described above.

For the rest of this chapter let us employ the following assumption.

General assumption. Let N, T be two disjoint subsets of the alphabet C.

Theorem 3.1 (context-freeness lemma) *Let P be a set of productions over N. Let $H \in \mathcal{H}_\mathcal{C}$, $A \in N$, and $k \in \mathbb{N}$. Then there is a derivation $A^\bullet \Longrightarrow^{k+1} H$ if and only if there is a production $(A, R) \in P$ and a mapping $repl\colon E_R^N \to \mathcal{H}_\mathcal{C}$ with $H = R[repl]$, such that $lab_R(e)^\bullet \Longrightarrow^{k(e)} repl(e)$ for all $e \in E_R^N$ and $\sum_{e \in E_R^N} k(e) = k$.*

Proof. We shall prove this result in detail because of central character. By definition of direct derivations, if a derivation $A^\bullet \Longrightarrow^{k+1} H$ exists it must have the form $A^\bullet \Longrightarrow R \Longrightarrow^k H$ for some $(A, R) \in P$. Hence, the proof is finished if we can show the following.

> Let G, G' be hypergraphs. Then we have $G \Longrightarrow^k G'$ if and only if there is some $repl\colon E_G^N \to \mathcal{H}_\mathcal{C}$ with $G' = G[repl]$, such that $lab_G(e)^\bullet \Longrightarrow^{k(e)} repl(e)$ for all $e \in E_G^N$ and $\sum_{e \in E_G^N} k(e) = k$.

We proceed by induction on k. For $k = 0$ both directions are trivial, so let us assume $k > 0$. Proving first the *only-if* direction, we must have $G \Longrightarrow G_0 \Longrightarrow^{k-1} G'$ where $G_0 = G[e_0/R_0]$ for some $e_0 \in E_G^N$ and some $(lab_G(e_0), R_0) \in P$. The induction hypothesis yields a mapping $repl_0\colon E_{G_0}^N \to \mathcal{H}_\mathcal{C}$ with $G' = G_0[repl_0]$, such that $lab_{G_0}(e)^\bullet \Longrightarrow^{k_0(e)} repl_0(e)$ for all $e \in E_{G_0}^N$, where the sum of the $k_0(e)$ is $k-1$. Now, let $repl(e) = repl_0(e)$ for $e \in E_G^N \setminus \{e_0\}$ and $repl(e_0) = R_0[repl'_0]$, where $repl'_0$ is the restriction of $repl_0$ to $E_{R_0}^N$. Then, if $repl''_0$ is the restriction of $repl_0$ to $E_G \setminus \{e_0\}$ we get $G' = G_0[repl_0] = G[e_0/R_0][repl_0] = G[repl''_0][e_0/R_0][repl'_0] = G[repl''_0][e_0/repl(e_0)] = G[repl]$, as required.

For the *if* direction, let $G' = G[repl]$ with $lab_G(e)^\bullet \Longrightarrow^{k(e)} repl(e)$ for all $e \in E_G^N$ and $\sum_{e \in E_G^N} k(e) = k$. We may assume without loss of generality that $k(e) > 0$ for all $e \in E_G^N$ (otherwise, consider the restriction of $repl$ to those hyperedges). Suppose $E_G^N = \{e_1, \ldots, e_n\}$. Then there are hypergraphs G_i

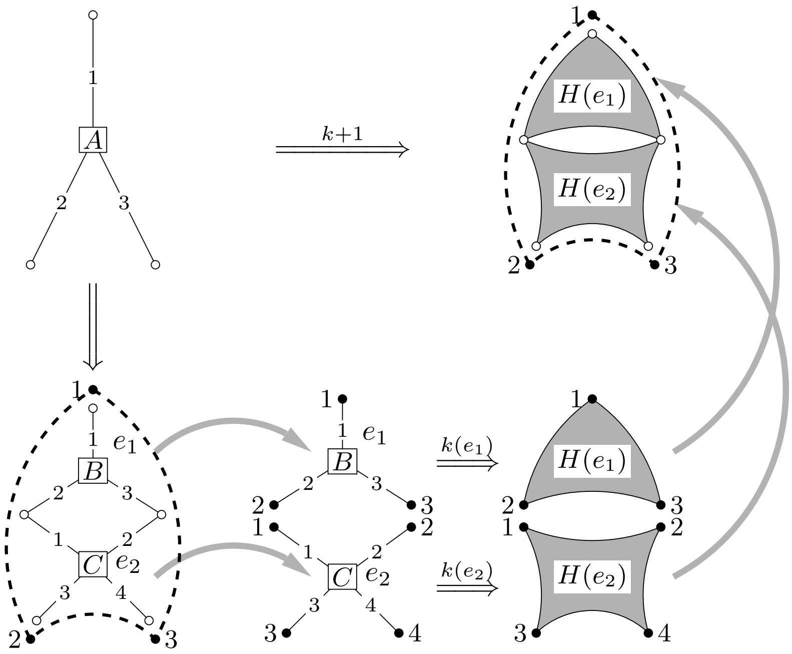

Figure 10: Decomposition of a derivation according to the context-freeness lemma

$(i = 1, \ldots, n)$ such that $lab_G(e_i)^\bullet \Longrightarrow G_i \Longrightarrow^{k(e_i)-1} repl(e_i)$ for $i = 1, \ldots, n$. Making use of the (already proved) *only-if* direction there are $repl_i \colon E^N_{G_i} \to \mathcal{H}_\mathcal{C}$ for $i = 1, \ldots, n$ such that $G_i[repl_i] = repl(e_i)$, $lab_{G_i}(e)^\bullet \Longrightarrow^{k_i(e)} repl_i(e)$ for all $e \in E^N_{G_i}$, and $\sum_{e \in E^N_{G_i}} k_i(e) = k(e_i) - 1$.

Let $G_0 = G[e_1/G_1, \ldots, e_n/G_n]$ and $repl_0(e) = repl_i(e)$ for all $e \in E^N_{G_i}$ $(i = 1, \ldots, n)$. Then we get $G' = G[repl] = G[e_1/G_1[repl_1], \ldots, e_n/G_n[repl_n]] = G[e_1/G_1, \ldots, e_n/G_n][repl_0] = G_0[repl_0]$. By the induction hypothesis this yields a derivation $G_0 \Longrightarrow^{k-n} G'$ and by construction of G_0 we have $G \Longrightarrow^n G_0$, which completes the proof. $\square$

The possibility of decomposing a derivation as stated in the context-freeness lemma may be illustrated as in Figure 10. The context-freeness lemma can also be formulated as a characterization of the languages generated from handles, which yields an alternative method of deriving hypergraphs.

Corollary 3.2 *Let $HRG = (N, T, P, S) \in \mathcal{HRG}$. For all $A \in N$ let $rhs(A) = \{R \in \mathcal{H}_\mathcal{T} \mid (A, R) \in P\}$. Then*

$$L_A(HRG) \\ = \bigcup_{R \in rhs(A)} \big\{ R[repl \colon E^N_R \to \mathcal{H}_\mathcal{T}] \;\big|\; repl(e) \in L_{lab_R(e)}(HRG) \text{ for } e \in E^N_R \big\}.$$

Proof. Apply the context-freeness lemma to terminal graphs which are the elements of the languages $L_A(HRG)$ for $A \in N$. $\square$

3.2 Derivation trees

The context-freeness lemma allows us to introduce the concept of derivation trees as a convenient representation of derivations. Consider some derivation $A^\bullet \Longrightarrow^* H$. If $k = 0$ then $H = A^\bullet$ and we may represent the derivation by a one-node tree whose only node is the label A. Otherwise, the context-freeness lemma states that the derivation has the form $A^\bullet \Longrightarrow R \Longrightarrow^* R[\mathit{repl}]$, where (A, R) is a production and $\mathit{lab}_R(e)^\bullet \Longrightarrow^* \mathit{repl}(e)$ for all $e \in E_R^N$. Thus, the derivation gives rise to a tree whose root is the production (A, R) and whose subtrees are the trees obtained recursively from the derivations $\mathit{lab}_R(e)^\bullet \Longrightarrow^* \mathit{repl}(e)$ for $e \in E_R^N$. Thus, derivation trees represent derivations up to some reordering of direct derivation steps (which does not affect the result of a derivation, as we know). The resulting hypergraph of a derivation tree is then given by $R[\mathit{repl}]$, where R is the right-hand side of the production in its root and $\mathit{repl}(e)$ is recursively obtained as the result of the subtree corresponding to $e \in E_R^N$.

Definition 3.3 (*derivation tree) The set $\mathit{TREE}(P)$ of* derivation trees over *P is recursively defined as follows.*

- *$N \subseteq \mathit{TREE}(P)$ with $\mathit{root}(A) = A$ and $\mathit{result}(A) = A^\bullet$ for $A \in N$.*
- *For every production $(A, R) \in P$ and every mapping $\mathit{branch}\colon E_R^N \to \mathit{TREE}(P)$ such that we have $\mathit{type}(e) = \mathit{type}(\mathit{root}(\mathit{branch}(e)))$ for all $e \in E_R^N$, the triple $t = (A, R, \mathit{branch})$ is in $\mathit{TREE}(P)$.*

  Furthermore, we let $\mathit{root}(t) = A$ and $\mathit{result}(t) = R[\mathit{repl}]$ where, for all $e \in E_R^N$, $\mathit{repl}(e) = \mathit{result}(\mathit{branch}(e))$.

One should notice that a derivation tree contains nonterminal labels $A \in N$ as subtrees if and only if its result is not terminal. The theorem below states the expected correspondence between derivations and derivation trees.

Theorem 3.4 *Let P be a set of productions over N, let $A \in N$ and $H \in \mathcal{H}_\mathcal{T}$. Then there is a derivation $A^\bullet \Longrightarrow^* H$ if and only if there is a derivation tree t over P with $\mathit{root}(t) = A$ and $\mathit{result}(t) = H$.*

Proof. For the first direction, suppose t is a derivation tree over P with $\mathit{root}(t) = A$ and $\mathit{result}(t) = H$. The proof is by induction. If $t = A$ then $\mathit{root}(t)^\bullet = A^\bullet \Longrightarrow^0 A^\bullet = \mathit{result}(t)$. If $t = (A, R, \mathit{branch})$, using the induction hypothesis we get a derivation $\mathit{root}(\mathit{branch}(e))^\bullet \Longrightarrow^* \mathit{result}(\mathit{branch}(e))$ for all $e \in E_R^N$. By the context-freeness lemma, this yields a derivation

$root(t)^\bullet = A^\bullet \Longrightarrow^* R[repl] = result(t)$ with $repl(e) = result(branch(e))$ for all $e \in E_R^N$.

The proof for the other direction can be done in a similar way by induction on the length of derivations using the other direction of the context-freeness lemma. $\square$

An interesting consequence of Theorem 3.4 is obtained by considering the root and the result of a derivation tree over P as a new production. Formally, let $P^* = \{(root(t), result(t)) \mid t \in TREE(P)\}$. Then, $A^\bullet \Longrightarrow_{P^*} H$ if and only if $A^\bullet \Longrightarrow_P^* H$, by Theorem 3.4. By induction on the length of derivations this yields, for every label A and all hypergraphs H,

$$A \Longrightarrow_P^* H \text{ if and only if } A \Longrightarrow_{P^*}^* H.$$

This property will be used in Section 7 in order to obtain a cubic membership algorithm for a certain class of edge replacement languages.

3.3 Bibliographic notes

To emphasize their context-freeness, hyperedge replacement grammars are sometimes called context-free hypergraph grammars (see, for example, [52,37] and Chapter [38] of this handbook). The context-freeness lemma was first formulated and proved for edge replacement grammars (see [50]); a formulation for hyperedge replacement grammars can be found in [52,48]. In [9] Courcelle presents an axiomatic definition of context-free rewriting. The definition requires that a context-free notion of replacement be associative and confluent. As mentioned in Section 2 hyperedge replacement indeed satisfies these requirements. Thus, hyperedge replacement is a context-free rewriting mechanism in Courcelle's sense. In fact, though in a somewhat different way, Theorem 3.1 also expresses nothing else than associativity and confluence of hyperedge replacement. It says that the order in which productions are applied is not important (on one side of the equivalence there is no order) and that we can first build the results of the sub-derivations and afterwards replace the hyperedges of the initial hypergraph with these results instead of doing it step by step (which is associativity).

In the literature, some suggestions concerning non-context-free extensions of hyperedge replacement are encountered. The parallel mode of rewriting as known from L-systems with tables may be employed. This was studied by Ehrig and Tischer [28] for edge replacement and by Kreowski [64] for hyperedge replacement. David, Drewes, and Kreowski [18] combined this with a rendezvous mechanism. Here, nodes of different right-hand sides added in a parallel step can be fused in a certain way whenever the replaced hyperedges are neighbours. This allows to overcome most of the restrictions. Besides the parallel case and other application conditions, Kreowski [65] discusses hyperedge replacement as a means to specify infinite (hyper)graphs by infinite derivations, which were first investigated by Bauderon [5,6].

Derivation trees as they are defined here are closely related to rule trees as introduced and investigated by Kreowski [63].

4 Structural properties

In this section, we discuss some structural properties of hyperedge replacement languages. They demonstrate that several results well-known for context-free string languages (like the fixed-point theorem, the pumping lemma, and Parikh's theorem) can be generalized to hyperedge replacement languages.

4.1 A fixed-point theorem

We start with a fixed-point theorem for hyperedge replacement languages generalizing Ginsburg's and Rice's fixed-point theorem for context-free string languages [47]. It is shown that hyperedge replacement languages are the least fixed points of their generating productions (considered as a system of language equations).

In the following, for every set C of labels let us denote by $\mathcal{HL}_\mathcal{C}$ the set of all languages of hypergraphs over C, that is, $\mathcal{HL}_\mathcal{C}$ is the powerset of $\mathcal{H}_\mathcal{C}$.

Definition 4.1 (*system of equations* **and** *fixed-point*) *Let $N, T \subseteq C$ be sets of labels with $N \cap T = \emptyset$. A* system of equations *over N is a mapping $EQ\colon N \to \mathcal{HL}_{\mathcal{N}\cup\mathcal{T}}$. A mapping $L\colon N \to \mathcal{HL}_\mathcal{T}$ is a* fixed-point *of EQ if, for all $A \in N$,*

$$L(A) = \bigcup_{R \in EQ(A)} \left\{ R[repl\colon E_R^N \to \mathcal{H}_\mathcal{T}] \;\middle|\; repl(e) \in L(lab_R(e)) \text{ for } e \in E_R^N \right\}.$$

$L\colon N \to \mathcal{HL}_\mathcal{T}$ is a least fixed-point *of EQ if L is a fixed-point of EQ and $L(A) \subseteq L'(A)$ for all $A \in N$ and all fixed-points L' of EQ.*

Remarks.

1. A system of equations $EQ\colon N \to \mathcal{HL}_{\mathcal{N}\cup\mathcal{T}}$ over N may be denoted by $A = EQ(A)$ for $A \in N$ to emphasize the name. Actually, it is nothing else than a representation of a set P of productions over N. For P, there is an *associated system of equations* $EQ_P\colon N \to \mathcal{HL}_{\mathcal{N}\cup\mathcal{T}}$ defined as $EQ_P(A) = \{R \mid (A, R) \in P\}$. Conversely, a system of equations $EQ\colon N \to \mathcal{HL}_{\mathcal{N}\cup\mathcal{T}}$ over N yields a set of productions $P(EQ) = \{(A, R) \mid R \in EQ(A)\}$.

2. The *language family* $L\colon N \to \mathcal{HL}_{\mathcal{T}}$ *generated by* a hyperedge replacement grammar *HRG* is given by $L(A) = L_A(HRG)$ for $A \in N$.

According to Corollary 3.2, the language family generated by a hyperedge replacement grammar is a fixed-point of the system of equations associated with the set of productions of the grammar. Even better, the following holds.

Theorem 4.2 (fixed-point theorem) *Let $HRG = (N, T, P, S)$ be a hyperedge replacement grammar, $EQ_P\colon N \to \mathcal{HL}_{\mathcal{N}\cup\mathcal{T}}$ the system of equations associated with P, and $L\colon N \to \mathcal{HL}_{\mathcal{T}}$ the language family generated by HRG. Then L is the least fixed-point of EQ_P.*

Proof. Let $L'\colon N \to \mathcal{HL}_{\mathcal{T}}$ be a fixed-point of EQ. We have to show $L(A) \subseteq L'(A)$ for all $A \in N$, that is, $H \in L'(A)$ for all $H \in \mathcal{H}_{\mathcal{T}}$ with $A^\bullet \Longrightarrow_P^* H$. This can be shown by induction on the length of derivations. $A^\bullet \Longrightarrow_P^1 H$ implies $(A, H) \in P$ with $H \in \mathcal{H}_{\mathcal{T}}$, so that $H[\mathit{empty}] = H$ is defined for the empty mapping $\mathit{empty}\colon \emptyset \to \mathcal{H}_{\mathcal{T}}$. Because L' is a fixed-point of EQ, $H = H[\mathit{empty}] \in L'(A)$.

Consider now $A^\bullet \Longrightarrow_P^{k+1} H$. Due to the context-freeness lemma, there are $(A, R) \in P$, and derivations $\mathit{lab}_R(e)^\bullet \Longrightarrow^{k(e)} \mathit{repl}(e)$ for $e \in E_R^N$ with $k(e) \leq k$ and $H = R[\mathit{repl}]$. By the induction hypothesis, $\mathit{repl}(e) \in L'(\mathit{lab}_R(e))$. Therefore, $H = R[\mathit{repl}] \in L'(A)$ because L' is a fixed-point of EQ. □

4.2 A pumping lemma

We now present a pumping lemma for hyperedge replacement languages. It says that each sufficiently large hypergraph belonging to a hyperedge replacement language can be decomposed into three hypergraphs *FIRST*, *LINK*, and *LAST*, so that a suitable composition of *FIRST*, k samples of *LINK* for each natural number k, and *LAST* yields also a member of the language. This theorem generalizes the well-known pumping lemma for context-free string languages (see, for instance, [4,59]). As in the string case, the pumping lemma can be used to show that certain languages are no hyperedge replacement languages.

A hypergraph H is *substantial* if $V_H \neq [\mathit{ext}_H]$ or $|E_H| > 1$. A hyperedge replacement grammar each of whose productions has a substantial right-hand side is *growing*. A growing hyperedge replacement grammar generates only substantial hypergraphs. Note that all except a finite number of hypergraphs in a hyperedge replacement language L are substantial, since $\{|\mathit{ext}_H| \mid H \in L\}$ is finite. Using a straightforward extension of the proof for the well-known result saying that one can eliminate empty and chain productions from context-free

Chomsky grammars it is not hard to prove the following normal-form result, which is used later on.

Lemma 4.3 *For every hyperedge replacement grammar HRG we can effectively construct a growing hyperedge replacement grammar HRG' of the same order satisfying $L(HRG') = \{H \in L(HRG) \mid H \text{ substantial}\}$.*

Definition 4.4 (*composition* **and** *iterated composition*)

1. *Let $X \in C$. Then $H \in \mathcal{H}_C$ is said to be* X-handled *if there is a unique type(X)-edge $e \in E_H$ with label X. In this case, e is called the* X-handle *of H; the hypergraph without the X-handle, $H - \{e\}$, is denoted by $H^{\ominus}$.*

2. *Let $H \in \mathcal{H}_C$ be a hypergraph with X-handle e and let $H' \in \mathcal{H}_C$ be a type(X)-hypergraph. Then the hypergraph $H[e/H']$ is called the* composition *of H and H' with respect to e and is abbreviated by $H \otimes H'$.*

3. *Let $H \in \mathcal{H}_C$ be an X-handled type(X)-hypergraph. Then for $k \in \mathbb{N}$, $H^k \in \mathcal{H}_C$ is recursively defined by $H^0 = X^{\bullet}$ and $H^{i+1} = H \otimes H^i$ for $i \geq 0$.*

Note that, in the definition above, H^i is X-handled, thus H^{i+1} is well-defined.

Theorem 4.5 (pumping lemma) *Let L be some hyperedge replacement language of order r (for some $r \in \mathbb{N}$). Then there exist constants p and q such that the following is true: For every hypergraph H in L with $|H| > p$ there are an X-handled hypergraph $FIRST$, a substantial X-handled type(X)-hypergraph $LINK$, and a type(X)-hypergraph $LAST$ for some $X \in C$ with $H = FIRST \otimes LINK \otimes LAST$, $|LINK \otimes LAST| \leq q$, and type$(LINK) \leq r$, such that $FIRST \otimes LINK^k \otimes LAST \in L$ for all $k \in \mathbb{N}$. Furthermore, for every hypergraph H in L with $|V_H| > p$ we can choose $LINK$ in such a way that $V_{LINK} \setminus [ext_{LINK}] \neq \emptyset$.*

*Proof.*Let $HRG = (N, T, P, S) \in \mathcal{HRG}_{\nabla}$ with $L(HRG) = L$ and n the number of nonterminals. Since there are only finitely many non-substantial hypergraphs in $L(HRG)$ we may assume HRG is growing. Let max be the size of the largest right-hand side of HRG. Let $t \in TREE(P)$ with $root(t) = S$ and $H = result(t) \in \mathcal{H}_T$. If $|H| > max^n$, then t contains a path from the root to a leaf longer than n such that one of the nonterminals, say X, occurs twice. In other words, t has a subtree t' with root X which has a proper subtree t'' with root X. Choose $LAST = result(t'')$, $LINK = result(t' - t'')$ and $FIRST = result(t - t')$ where $t' - t''$ is obtained from t' by removing the subtree t'' and $t - t'$ by removing t' from t. Then, $FIRST$ and $LINK$ are X-handled, $LINK$ and $LAST$

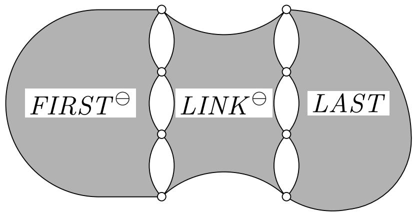

Figure 11: The result of composing *FIRST*, *LINK*, and *LAST*.

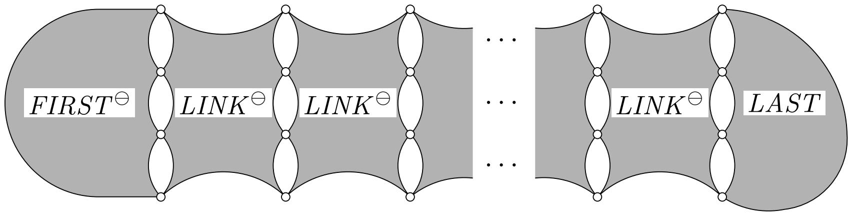

Figure 12: The hypergraph obtained by composing $FIRST^{\ominus}$, k samples of $LINK^{\ominus}$, and $LAST$.

are $type(X)$-hypergraphs, and $H = FIRST \otimes LINK \otimes LAST$. Since *HRG* is growing, *LINK* must be substantial. As in the case of the pumping lemma for context-free string languages one can now show $FIRST \otimes LINK^k \otimes LAST \in L$ for all $k \in \mathbb{N}$.

If we even have $|V_H| > p$ we may choose $t' - t''$ in such a way that $result(t' - t'')$ has at least one internal node. This is because, in this case it suffices to consider the nodes of the derivation tree of which the corresponding right-hand sides have internal nodes. This yields the second statement of the theorem and thus finishes the proof. □

The composition of H using *FIRST*, *LINK*, and *LAST* can be depicted as shown in Figure 11. The pumped hypergraphs have the shape depicted in Figure 12. Note that $FIRST^{\ominus}$, $LINK^{\ominus}$, and *LAST* are not necessarily connected hypergraphs (see Example 4.6 below).

As in the case of context-free string grammars, the pumping lemma can be used to show that the *finiteness problem* is decidable for hyperedge replacement languages. Another helpful consequence of the pumping lemma is the linear growth within hyperedge replacement languages. More explicitly, let L be an infinite hyperedge replacement language. Then the pumping of a large enough member of L yields an infinite sequence of hypergraphs in L, say $H_0, H_1, H_2, \ldots$, and constants $c, d \in \mathbb{N}$ with $c + d > 0$ such that $\left|V_{H_{i+1}}\right| = |V_{H_i}| + c$ and $\left|E_{H_{i+1}}\right| = |E_{H_i}| + d$, for all $i \geq 0$.

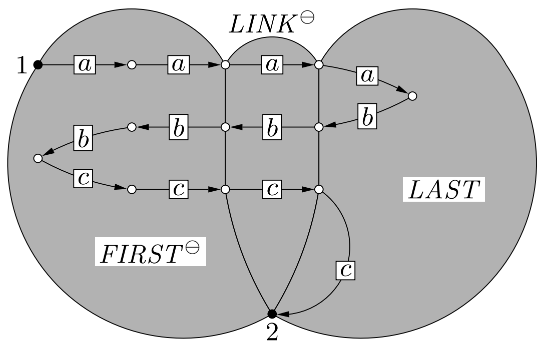

Figure 13: Decomposition of $(a^4b^4c^4)^\bullet$.

Example 4.6 First, we want to illustrate the pumping property for the string-graph language $L = \{(a^nb^nc^n)^\bullet \mid n \geq 1\}$ that can be generated by a hyperedge replacement grammar of order 4, as shown in Example 2.3. For $n \geq 3$, the string graph $(a^nb^nc^n)^\bullet$ can be decomposed as indicated in Figure 13 for the case $n = 4$.

FIRST is a 2-hypergraph containing the two external nodes of the string graph as external nodes. It consists of three chains of edges of length $n-2$, an a-chain, a b-chain, and a c-chain, where the a-chain is attached to the first external node and the c-chain is attached to the end of the b-chain. Moreover, it may be seen as 4-handled, where the first pair of tentacles is attached to the end of the a-chain and the beginning of the b-chain and the second pair of tentacles is attached to the end of the c-chain and the second external node. It is composed with the *LINK*-component with respect to this 4-handle. The *LINK*-component possesses a 4-handle which is attached to the target of the a-edge, the source of the b-edge, the target of the c-edge and the second external node. Note that neither $FIRST^\ominus$ (*FIRST* without the 4-handle) nor $LINK^\ominus$ (*LINK* without the 4-handle) nor *LAST* is connected. Moreover, *LINK* is non-trivial.

The pumped 2-hypergraphs have the shape shown in Figure 14. Obviously, each resulting 2-hypergraph is a string graph of the form $(a^nb^nc^n)^\bullet$ for some $n \geq 1$.

One of the main uses of results like the pumping lemma is to prove that specific hypergraph languages are no hyperedge replacement languages. There is, for example, no hyperedge replacement language of unbounded connectivity. To see this, assume such a language is generated by a hyperedge replacement grammar of order r. Then, a sufficiently large graph of connectivity greater than r could be written as $FIRST \otimes LINK \otimes LAST$, where $|LINK \otimes LAST| \leq q$

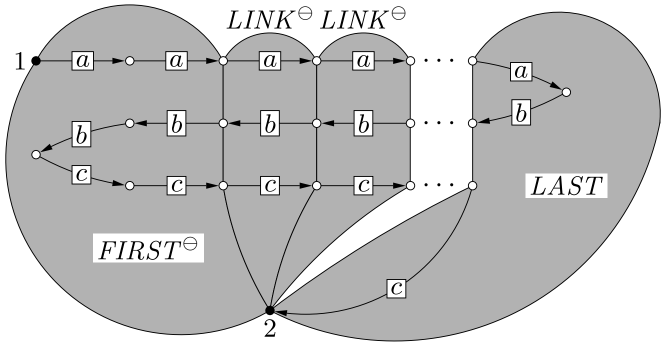

Figure 14: Pumping of $(a^4b^4c^4)^\bullet$.

for some fixed q, $type(LINK) \leq r$, and $LINK$ contains at least one internal node. But since q is fixed we may assume that $FIRST$ contains more than r nodes, so $FIRST \otimes LINK \otimes LAST$ is of connectivity at most r because removing the external nodes of $LINK$ from the graph yields at least two components.

As another example, the string-graph language $\{(a^{n^2})^\bullet \mid n \geq 1\}$ cannot be generated by a hyperedge replacement grammar because the growth of the number of edges is not linear. The pumping lemma can also be used to show that the string-graph language $\{(a_1^n \cdots a_{2k}^n)^\bullet \mid n \in \mathbb{N}\}$ cannot be generated by any hyperedge replacement grammar of order less than $2k$, for $k \geq 1$. This will be done in Section 5 (see Example 5.3).

4.3 Parikh's theorem

As a third structural result, we present the hyperedge replacement version of Parikh's theorem [77] which relates hyperedge replacement languages to semilinear sets. The proof makes use of Parikh's theorem for context-free string languages by mapping hyperedge replacement grammars to context-free string grammars. In the following, for $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ in $\mathbb{N}^\ltimes$ and $c \in \mathbb{N}$, let us define $x + y = (x_1 + y_1, \ldots, x_n + y_n)$, $x - y = (x_1 - y_1, \ldots, x_n - y_n)$, and $cx = (cx_1, \ldots, cx_n)$.

Definition 4.7 (*Parikh mapping* **and** *semilinear set***)**

1. *Let $T = \{a_1, \ldots, a_n\}$ be an alphabet and $\psi\colon \mathcal{H}_T \to \mathbb{N}^\ltimes$ be the mapping given by $\psi(H) = (\#_{a_1}(H), \ldots, \#_{a_n}(H))$, where $\#_{a_i}(H)$ denotes the number of a_i-labelled hyperedges in $H \in \mathcal{H}_T$. Then ψ is called a* Parikh mapping*. For every language $L \subseteq \mathcal{H}_T$, $\psi(L)$ denotes the set $\psi(L) = \{\psi(H) \mid H \in L\}$.*

2. A set $S \subseteq \mathbb{N}^{\mathcal{K}}$ is linear *if S is of the form*

$$S = \{x_0 + \sum_{i=1}^{k} c_i x_i \mid c_1, \dots, c_k \in \mathbb{N}\},$$

where $k \geq 1$ and $x_1, \dots, x_k \in \mathbb{N}^{\mathcal{K}}$.

3. $S \subseteq \mathbb{N}^{\mathcal{K}}$ is semilinear *if it is the union of a finite number of linear sets.*

Theorem 4.8 (Parikh's theorem) *For all hyperedge replacement languages L and every Parikh mapping ψ, the set $\psi(L)$ is semilinear.*

Proof. Let L be a hyperedge replacement language and $HRG = (N, T, P, S)$ be a hyperedge replacement grammar with $L = L(HRG)$. We construct a context-free string grammar G as follows: Let $str\colon \mathcal{H}_{\mathcal{N}\cup\mathcal{T}} \to (\mathcal{N} \cup \mathcal{T})^*$ be the mapping assigning the string $str(H) = lab_H(e_1) \cdots lab_H(e_n)$ to a hypergraph H with $E_H = \{e_1, \dots, e_n\}$ (ordered in some arbitrary way) and let $G = (N, T, str(P), S)$ with $str(P) = \{(A, str(R)) \mid (A, R) \in P\}$. Then $\psi(L(HRG)) = \psi_{str}(L(G))$ where ψ_{str} denotes the usual Parikh mapping for string languages. By Parikh's theorem for context-free string languages, the set $\psi_{str}(L(G))$ is semilinear. Consequently, the set $\psi(L(HRG)) = \psi(L)$ is semilinear. □

It is perhaps interesting to notice that one may attach a 1-edge with a special label to each internal node of a right-hand side of a production. Then counting the hyperedges with the special label means counting the number of (internal) nodes. Consequently, counting the number of nodes in hypergraphs of a hyperedge replacement language yields a semilinear set, too. As a second remark, it is not hard to see that there are several languages L (such as the set of all graphs) which have the semilinearity property but are no hyperedge replacement languages.

4.4 Bibliographic notes

The fixed-point theorem for hyperedge replacement languages was formulated for edge replacement in [50] by Habel and Kreowski and for hyperedge replacement by Bauderon and Courcelle in [7] (in a slightly different form). The reader should compare the fixed-point theorem with the way in which Courcelle defines *HR sets of hypergraphs* in Section 5 of Chapter [13]. HR sets of hypergraphs are just the least fixed-points of systems of equations. However, the allowed equations do not make use of hyperedge replacement. They are built upon a more primitive set of operations, which nevertheless give rise to the same least fixed-points. Thus, HR sets of hypergraphs are hyperedge replacement languages and vice versa.

The pumping lemma is based on the pumping lemma for edge replacement languages given in Kreowski [62]. A hyperedge replacement version was first given in [52] and proved in [48]. Moreover, a number of consequences of the pumping lemma for hyperedge replacement languages can be found in [48]. In particular, it is shown that for hyperedge replacement languages of simple graphs the clique size and the minimum degree is bounded.

The extension of Parikh's theorem to hyperedge replacement languages was first published in [48]. Extensions of Parikh's theorem are also discussed in Section 6.3 of Chapter [13].

5 Generative power

In this section, we discuss the generative power of hyperedge replacement grammars, dependent on their order. By definition, $\mathcal{HRL}_{/} \subsetneqq \mathcal{HRL}_{\infty} \subsetneqq \mathcal{HRL}_{\in} \subsetneqq \cdots$ since there cannot occur any hypergraph of type k in a language $L \in \mathcal{HRL}_{\|'}$, for $k > k'$. In the following the generative power of graph and of string-graph generating hyperedge replacement grammars is studied.

5.1 Graph-generating hyperedge replacement grammars

We first show that the generative power of hyperedge replacement grammars depends on their order, that is, on the maximum number of tentacles involved in the replacement of hyperedges, even if only graph-generating grammars are considered. For example the set of all (partial) k-trees can easily be shown to be in $\mathcal{HRL}_{\|}$. Using the pumping lemma it turns out that this language is not in $\mathcal{HRL}_{\|-\infty}$. In other words, the family $(\mathcal{HRL}_{\|})_{\|\in\mathbb{N}}$ forms an infinite hierarchy of classes of hypergraph languages, that remains proper if restricted to graph languages because k-trees are graphs.

Example 5.1 (k-tree and partial k-tree) As usual, let us call two nodes of a graph *adjacent* if they are connected by an edge. A *k-clique* of a graph is a subset of k pairwise adjacent nodes of this graph. For $k \geq 1$, the set $kTREE$ of all *k-trees* (see Rose [79]) is recursively defined as follows:

- Every complete graph with k nodes is a k-tree on k-nodes.
- Given a k-tree H on n nodes and a k-clique $G \subseteq H$, a k-tree on $n+1$ nodes is obtained when a new $(n+1)$-th node is made adjacent to each node of G.

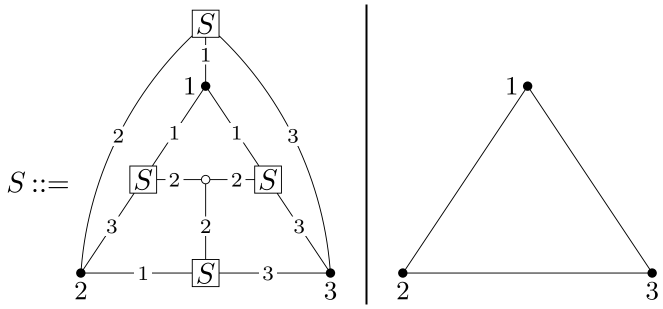

Figure 15: Productions to generate 3-trees.

A *partial* k-tree is an undirected graph that is obtained from a k-tree by removing an arbitrary number of edges. The set of all partial k-trees is denoted by $kTREE^{\mathrm{p}}$. For arbitrary, but fixed positive k, one can easily find a hyperedge replacement grammar HRG_{kTREE} of order k generating the set of all k-trees. The grammar simulates the recursive construction of a k-tree: There is one production that produces a new node *new* and $k+1$ hyperedges indicating the old k-clique and the k newly created k-cliques. The second production allows to replace a hyperedge by a k-clique. For $k = 3$ this yields productions as shown in Figure 15. Thus, for $k \geq 1$, the set $kTREE$ can be generated by a grammar of order k. Suppose $kTREE$ is a language of order $k-1$. Then, except perhaps a finite number, all elements in $kTREE$ are of connectivity $\leq k-1$ due to the pumping lemma. On the other hand, each k-tree with at least $k+1$ nodes is of connectivity k, a contradiction.

Furthermore, for $k \geq 2$, the addition of a production removing an ordinary edge yields a hyperedge replacement grammar $HRG_{kTREE^{\mathrm{p}}}$ of order k generating the set of all partial k-trees. Hence, by the same arguments as above, for $k \geq 2$ the set $pkTREE$ of all partial k-trees can be generated by a grammar of order k, but not by a grammar of order $k-1$.

Let $\mathcal{L}_{GRAPH}$ be the class of all graph languages. As an immediate consequence of Example 5.1, we get the following result, saying that the classes $\mathcal{GL}_{\|} = \mathcal{HRL}_{\|} \cap \mathcal{L}_{GRAPH}$ ($k \in \mathbb{N}$) of hyperedge replacement graph languages of order k form a proper hierarchy.

Theorem 5.2 (hierarchy theorem 1) $\mathcal{GL}_{\|} \subsetneq \mathcal{GL}_{\|+\infty}$ *for all* $k \in \mathbb{N}$.

5.2 String-generating hyperedge replacement grammars

Let us now turn to the classes of string-graph languages generated by hyperedge replacement grammars of order k. We denote by $\mathcal{SL}_{\|}$ the set of all hyperedge replacement languages of order k which solely consist of string graphs. Compared with the situation encountered above, these classes turn out to behave slightly different. We show that the family $(\mathcal{SL}_{\in\|})_{\|\geq\infty}$ forms an infinite hierarchy that starts with the class of context-free string languages (represented as string-graph languages), but $\mathcal{SL}_{\in\|+\infty} = \mathcal{SL}_{\in\|}$ for all $k \in \mathbb{N}$.

It is quite obvious that, for a given string language L, $L^\bullet$ is in $\mathcal{SL}_{\in}$ if and only if L is context-free. If L is context-free, just choose the set of all productions $(A, w^\bullet)$ for which (A, w) is a production in the given context-free grammar to build an edge replacement grammar yielding $L^\bullet$. For the other direction, using standard techniques one can remove productions that just delete an edge. Afterwards, it is not hard to see that every right-hand side of a production applied in a derivation yielding a string graph must be a string graph, and the productions of this kind obviously induce the needed context-free productions for the string case.

In the following, we present an example of a string-graph language that can be generated by a grammar of order $2k$, but not by a grammar of smaller order. As a consequence, we get a proper language hierarchy.

Example 5.3 The string-graph language $L_k = \{(a_1^n \cdots a_{2k}^n)^\bullet \mid n \in \mathbb{N}\}$ can be generated by the hyperedge replacement grammar of order $2k$, for every $k \geq 1$. The construction is a straightforward generalization of the one presented in Example 2.3. Thus, L_k is a string-graph language of order $2k$. Making use of the pumping lemma, it can be shown that L_k is no string-graph language of order $2k-1$. To see this, assume L_k is a language of order $2k-1$. Let *FIRST*, *LINK*, and *LAST* be as in the pumping lemma, for some sufficiently large member of L_k. Then *LINK* has type less than $2k$ and is substantial. The latter implies $E_{LINK^\ominus} \neq \emptyset$ because $|V_H| = |E_H| + 1$ for all $H \in L_k$.

Since $FIRST \otimes LINK \otimes LAST$ is a string graph the connected components of $LINK^\ominus$ must be paths whose first and last nodes are in $[ext_{LINK}] \cup [att_{LINK}(e)]$, where e is the X-handle of *LINK*. Hence, the number of connected components of $LINK^\ominus$ which contain at least one edge is at most $k-1$. Moreover, none of these connected components can contain edges with different labels, since otherwise pumping obviously yields graphs not in L_k. But then $LINK^\ominus$ lacks at least one of the labels, so $FIRST \otimes LINK \otimes LAST \in L$ implies $FIRST \otimes LAST \notin L$ because $E_{LINK^\ominus} \neq \emptyset$.

A similar reasoning as in the previous example shows that the string-graph language $W^k = \{(w^k)^\bullet \mid w \in T^+\}$ is of the same type. It can be generated by a hyperedge replacement grammar of order $2k$ but not by any one of order $2k-1$. The example proves the second hierarchy theorem.

Theorem 5.4 (hierarchy theorem 2) *For all* $k \in \mathbb{N}$, $\mathcal{SL}_{\in\|} \subsetneqq \mathcal{SL}_{\in(\|+\infty)}$.

The obvious question is, of course, whether one can also prove $\mathcal{SL}_{\|} \subsetneqq \mathcal{SL}_{\|+\infty}$. More general, which string-graph languages are generated by hyperedge replacement grammars of order k? In the following, we want to give a characterization by Engelfriet and Heyker [33] of these languages by means of *deterministic tree-walking transducers*, a class of finite automata operating on trees that was introduced by Aho and Ullman. These transducers act on regular sets of terms (that we may also consider as sets of ordered trees over some ranked alphabet). A regular set of terms is generated by a *regular tree grammar* (see [46]) $R = (\Sigma, \Gamma, \Psi, \gamma_0)$, where Σ is a ranked alphabet, Γ is a set of nonterminals with $\gamma_0 \in \Gamma$ (the *start symbol*), and Ψ is a finite set of productions of the form $\gamma \to f(\gamma_1, \ldots, \gamma_n)$, for $\gamma, \gamma_1, \ldots, \gamma_n \in \Gamma$ and $f \in \Sigma$ of rank n. The language $L(R)$ of terms generated by R is defined in the obvious way. It contains all terms over Σ that can be derived from γ_0 by considering the productions as term rewrite rules. The set $occ(t)$ of *occurrences* of a term t over Σ is defined as usual: If $t = f(t_1, \ldots, t_n)$ then $occ(t) = \{\lambda\} \cup \{io \mid 1 \leq i \leq n \text{ and } o \in occ(t_i)\}$. We denote by $o(t)$ the symbol at occurrence $o \in occ(t)$ and by t/o the subterm of t whose root is the occurrence o in t. The parent occurrence of oi ($i \in \mathbb{N}$) is o and the *parent occurrence* of λ is $\perp$.

Definition 5.5 (*deterministic tree walking transducer***)** *A* deterministic tree walking transducer *(dtwt) is a tuple* $M = (Q, R, \Delta, \delta, q_0, F)$*, where* Q *is a finite set of states with* $q_0 \in Q$ *(the initial state) and* $F \subseteq Q$ *(the set of final states),* $R = (\Sigma, \Gamma, \Psi, \gamma_0)$ *a regular tree grammar,* Δ *the output alphabet, and* $\delta\colon Q \times \Sigma \to Q \times D \times \Delta^*$ *the transition function with* $D = \{stay, up\} \cup \{down(i) \mid i \geq 1\}$.

A configuration *of* M *is a tuple* (q, t, o, w) *with* $q \in Q$, $t \in L(R)$, $o \in occ(t) \cup \{\perp\}$*, and* $w \in \Delta^*$*. A step of* M *turns* (q, t, o, w) *into* (q', t, o', ww') *if we have* $\delta(q, o(t)) = (q', d, w')$ *and either*

- $d = up$ *and* o' *is the parent occurrence of* o*, or*
- $d = stay$ *and* $o' = o$*, or*
- $o(t)$ *is of rank* $\geq i$, $d = down(i)$*, and* $o' = oi$.

If M*, starting with configuration* $(q_0, t, \lambda, \lambda)$*, finally reaches a configuration* $(q, t, \perp, w)$ *with* $q \in F$ *then the computation is* successful *and yields the output* w*.* $L(M)$ *denotes the language of all outputs of* M*, and* $\mathcal{DTWT}^+$ *denotes the class of all dtwt's* M *with* $\lambda \notin L(M)$.

A property of dtwt's that will turn out to be related to the order of hyperedge replacement grammars is the so-called crossing number. Consider a successful run of M on t. Every move from an occurrence o to an occurrence o' or from o' to o, where o' is the parent occurrence of o, is a *crossing* of the edge between o and o'. If $k \in \mathbb{N}$ is such that for no successful run on any input tree, an edge is crossed more than $2k$ times then M is said to be k-crossing. Note that every dtwt is k-crossing for some $k \in \mathbb{N}$, for if a computation crosses an edge more than $|Q|$ times, a node is visited twice with the same state, so the computation cannot end. Let us denote by $\mathcal{DTWT}^{+}_{\|}$ the set of all k-crossing $M \in \mathcal{DTWT}^{+}$. Then, the following can be shown.

Lemma 5.6 *For all* $k \in \mathbb{N}$, $\mathcal{SL}_{\in\|+\infty} \subseteq \mathcal{L}(\mathcal{DTWT}^{+}_{\|})$.

Proof. We roughly sketch the basic idea underlying the construction. One first proves that a given hyperedge replacement grammar $HRG = (N, T, P, S)$ generating a string-graph language can be modified (without changing the order), such that for every $A \in N$ there is some $out_A \in \{0, 1\}^{type(A)}$ such that for all $H \in L_A(HRG)$ the out-degree of $ext_H(i)$ is $out_A(i)$, for $i = 1, \ldots, type(A)$. Furthermore, one can ensure by a straightforward construction that for every right-hand side R in HRG and all distinct $e, e' \in E^N_R$ we have $lab_R(e) \neq lab_R(e')$. Then one can construct a dtwt $M = (Q, R, T, \delta, q_0, F) \in \mathcal{DTWT}^{+}_{\|}$ with $L(M) = L(HRG)$, as follows.

R is designed so that, roughly speaking, $L(R)$ is the set of derivation trees of HRG. The nonterminals of R are those of HRG, the start symbol is S, and the alphabet is P, where every $(A, H) \in P$ has rank $\left|E^N_H\right|$. For every production $p = (A, H) \in P$, choose an arbitrary (but fixed) order $e_1, \ldots, e_n$ on E^N_H, and let R contain the production $A \to p(lab_H(e_1), \ldots, lab_H(e_n))$. Now, M is constructed to walk on the trees $t \in L(R)$. Every node of a right-hand side of a production in P is a state of M. If M is at occurrence o of $t \in L(R)$ and $o(t) = (A, H)$, then the current state is a node v of H. M works by searching for the occurrence in t generating the terminal edge e whose first attached node is the image of v in the generated string graph. If there is some $e \in E^T_H$ with $att_H(e, 1) = v$, the occurrence has been found. M follows e by changing its state to $att_H(e, 2)$, outputs $lab_H(e)$, and stays at the same occurrence of t. If there is no such edge, but there is $e_i \in E^N_H$ and $j \in \{1, \ldots, type(lab_H(e_i))\}$ with $att_H(e_i, j) = v$ and $out_{lab_H(e_i)}(j) = 1$, then M moves to the i-th subtree by a $down(i)$-move and proceeds in state $ext_{H'}(j)$ (where H' is the right-hand side of the new occurrence). If neither one of these two possibilities holds, the edge sought is not generated within the current subtree, so v must be an external node $ext_H(i)$ of H (since a string graph is generated). Then M performs an up-move and—if H' is the right-hand side of the occurrence reached thereby—assumes state $att_{H'}(e,i)$, where e is the (unique) A-labelled hyperedge of H'.

In order to implement the *down*- and *up*-moves correctly we have to use some auxiliary states that remember i and (A,i), respectively, since we do not know H' in advance. Since M enters and leaves the occurrences of an input tree in states which are external nodes of the right-hand sides, and none of these external nodes is used twice it is not hard to see that M is k-crossing. □

Lemma 5.7 *For all $k \in \mathbb{N}$, $L(\mathcal{DTWT}^{+}_{\|}) \subseteq \mathcal{SL}_{\in\|}$.*

Proof. The construction is based on the following ideas. First, one shows that it suffices to consider transducers M without *stay*-moves that produce exactly one output symbol in each step. (The latter is due to the fact that for $L_1^\bullet \in \mathcal{SL}_{\|}$ the image L_2 of L_1 under a string homomorphism satisfies $L_2^\bullet \in \mathcal{SL}_{\|}$, provided $\lambda \notin L_2$.) Now, if M satisfies these requirements and is k-crossing, every subtree t/o of an input tree t is visited at most k times, where a visit to t/o is considered to start at occurrence o at the time it is entered from its parent occurrence o' and ends at o' at the time o' is reached the next time. During each visit, a (nonempty) substring of the output string is produced. The visit—and thus the string produced —- is composed of the visits to the immediate subtrees, which are connected by the steps where M is at occurrence o, leaving it down- or upwards.

To exploit these observations, one designs a hyperedge replacement grammar *HRG* whose derivation trees are more or less the trees in $L(R)$, such that the result of a derivation tree corresponding to t/o is the disjoint union of the $l \leq k$ substrings generated by the l visits of M to t/o. The $2l$ end nodes of the strings are the external nodes, and the substrings resulting from the different visits are generated in parallel. Consistency can be ensured by guessing (using the nonterminal labels) the states M is in when the i-th visit begins and ends, respectively. The productions of *HRG* corresponding to a rule $\gamma_0 \to f(\gamma_1,\dots,\gamma_n)$ of R are of the form (X_0,H), where H has nonterminal hyperedges $e_1,\dots,e_n$ labelled with $X_1,\dots,X_n$. Every label X_i is of the form $\langle\gamma_i, \mathit{begin}_i(1)\mathit{end}_i(1)\cdots\mathit{begin}_i(l_i)\mathit{end}_i(l_i)\rangle$, where $l_i \leq k$ and $\mathit{begin}_i(j), \mathit{end}_i(j)$ are the guessed states for the start and the end of the i-th visit to the tree generated by γ_i. The type of X_i is $2l_i$ and $V_H = \sum_{i=0}^{n}\{\mathit{begin}_i(1), \mathit{end}_i(1),\dots,\mathit{begin}_i(l_i), \mathit{end}_i(l_i)\}$, where we define $att_H(e_i) = \mathit{begin}_i(1)\mathit{end}_i(1)\cdots\mathit{begin}_i(l_i)\mathit{end}_i(l_i)$ for $i = 1,\dots,n$. In addition, H contains edges labelled with the output symbols M generates when leaving the node f, that is, the root of the subtree generated by γ_0, down- or upwards. As an example, see the hypergraph in Figure 16. For this to be a consistent right-hand side obtained from $\gamma_0 \to f(\gamma_1,\gamma_2)$ we must have, for instance, $\delta(\bar{q}_1, f) = (q_3, \mathit{down}(2), b)$. □

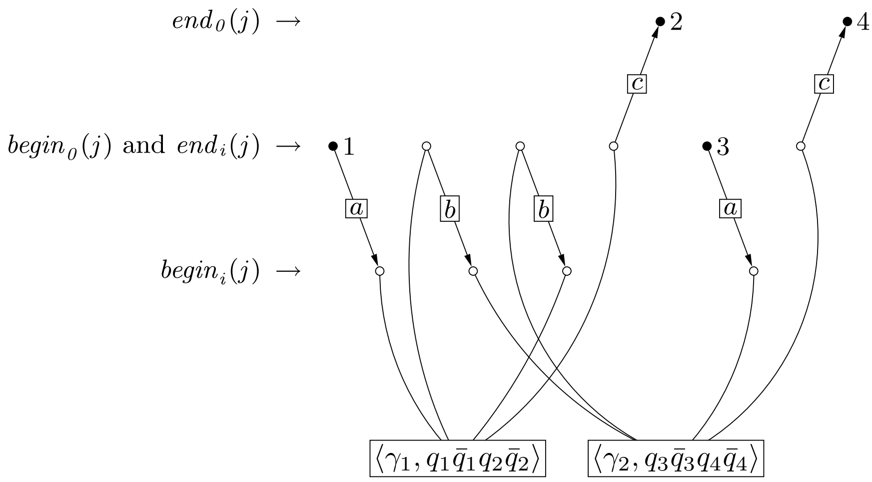

Figure 16: A Right-hand side in the grammar constructed to simulate a dtwt. The tentacles of hyperedges are to be numbered from left to right.

Lemmas 5.6 and 5.7 have a number of interesting consequences, of which we mention only two. First of all, it follows that, indeed, the string generating power of hyperedge replacement grammars increases only every second step.

Theorem 5.8 *For all* $k \in \mathbb{N}$, $\mathcal{SL}_{\in\|} = \mathcal{L}(\mathcal{DTWT}^{+}_{\|}) = \mathcal{SL}_{\in\|+\infty}$.

As mentioned above, every dtwt is k-crossing for some $k \in \mathbb{N}$, so we get the following characterization of the class of all hyperedge replacement string-graph languages.

Theorem 5.9 $\mathcal{SL} = \mathcal{L}(\mathcal{DTWT}^{+})$.

5.3 Further results and bibliographic notes

Most of the results concerning the generative power are formulated in [52] and proved in [48].

The notion of tree-walking transducers was invented by Aho and Ullman ([2]; see also [39]). Lemmas 5.6 and 5.7 and their consequences are taken from a paper by Engelfriet and Heyker [33], where the results are proved in detail. (Figure 16 is also from [33], although slightly modified.) Engelfriet and Heyker mention a number of additional consequences, including results for linear hyperedge replacement grammars that generate string-graph languages. Roughly speaking, one obtains the same results as presented here, by looking at two-way deterministic generalized sequential machines instead of dtwt's.

(The former can be seen as dtwt's whose second component is a linear regular tree grammar.) From results known for dtwt's Engelfriet and Heyker also obtain the corollary that the class $\mathcal{SL}$, its linear counterpart, and $\mathcal{SL}_{\in\|}$ are substitution-closed full AFLs (see [60]).

Asking similar questions for term-generating rather than string-generating hyperedge replacement grammars Engelfriet and Heyker show in [34] that hyperedge replacement grammars have the same term-generating power as attribute grammars. In this context one should also mention the work by Engelfriet and Vogler [41]. They define the so-called tree-to-graph-to-tree transducers, a sort of tree transducers making use of hyperedge replacement in order to construct the computed output trees. The main result of [41] states that these transducers—in the deterministic total case—have the same power as macro tree transducers (for the later see [40].

Quite a lot is also known about the generative power of hyperedge replacement grammars compared with other graph generating types of grammars. A comparison of hyperedge replacement grammars and boundary edNCE graph grammars is established by Engelfriet and Rozenberg in [37]. It is shown that hyperedge replacement grammars and boundary graph grammars of bounded nonterminal degree have the same power, both for generating sets of graphs and for generating sets of hypergraphs. Arbitrary boundary graph grammars have more generating power than hyperedge replacement grammars, but they have the same hypergraph generating power (subject to a representation of hypergraphs as bipartite graphs).

In [35], Engelfriet and Heyker compare the power of hyperedge replacement grammars and separated handle hypergraph grammars (or S-HH grammars) to generate hypergraph languages of bounded degree. It is shown that every S-HH language of bounded degree can be generated by a (separated) hyperedge replacement grammar. This implies that those two types of grammars generate the same class of graph languages of bounded degree. In the general case, incomparable classes of hypergraph languages are generated.

In [36], Engelfriet, Heyker, and Leih show that hyperedge replacement graph languages of bounded degree are generated by apex graph grammars.

In [12], Courcelle separates vertex replacement from hyperedge replacement: A set of graphs generated by vertex replacement is a hyperedge replacement language if and only if its graphs do not contain arbitrarily large complete bipartite graphs $K_{n,n}$ as subgraphs if and only if its graphs have a number of edges that is linearly bounded in terms of the number of vertices. These properties can be shown to be decidable.

In [14], Courcelle and Engelfriet give a grammar independent characterization of hyperedge replacement languages (see also Section 4.2 of Chapter [38]).

These are exactly the images of the recognizable sets of finite trees under certain graph transformations definable in monadic second-order logic. Several results follow saying that sets of graphs defined by vertex-replacement satisfying additional conditions (like bounded degree, planarity, bounded tree-width, or closure under subgraphs or minors) are hyperedge replacement languages.

6 Decision problems

A hyperedge replacement grammar specifies a hypergraph language. Unfortunately, the derivation process never produces more than a finite subset of the language explicitly (and even this may consume much time before significant members of the language occur). Hence, one may wonder what the hyperedge replacement grammar can tell about the generated language. As a matter of fact, the context-freeness lemma leads the way. Given a hyperedge replacement grammar and an arbitrary terminal hypergraph H with derivation $A^\bullet \Longrightarrow R \Longrightarrow^* H$, we get a decomposition of H into smaller components which are derivable from the handles of the hyperedges in R. This can be employed if one wants to reason about certain graph-theoretic properties or numerical functions on (hyper)graphs. The first leads to the notion of *compatible properties*, the second to *compatible functions*.

6.1 Compatible properties

If a graph-theoretic property can be tested for each H generated by a hyperedge replacement grammar by testing the property (or related properties) for the components and composing the results to a result for H, it is called *compatible*. It can be shown that compatibility implies decidability of the following questions:

- Is there a hypergraph in the generated language having the property?
- Do all hypergraphs in the generated language (except perhaps for a finite number) have the property?
- Are there only a finite number of hypergraphs in the generated language having the property?

These decision problems are reduced to the emptiness problem and to the finiteness problem by a filter theorem that shows that the intersection of a hyperedge replacement language with the set of hypergraphs that satisfy a compatible property is a hyperedge replacement language, too.

As an illustrating example, let us consider the property that two external nodes of a hypergraph are connected by a path. Let $H \in \mathcal{H}_\mathcal{C}$ and $v, v' \in V_H$. A *vv'-path* (in H) is an alternating sequence $v_0 e_1 v_1 \cdots e_n v_n$ with $v_0, \ldots, v_n \in V_H$, $e_1, \ldots, e_n \in E_H$, $v = v_0$, and $v' = v_n$ such that, for $i = 1, \ldots, n$, v_{i-1} and v_i are attachment nodes of e_i. If there is such a path, H is called *vv'-connected*.

Let $HRG = (N, T, P, S) \in \mathcal{HRG}$, and consider $A^\bullet \Longrightarrow R \Longrightarrow^* H$ with $(A, R) \in P$ and $H \in \mathcal{H}_\mathcal{T}$ as well as $lab_R(e)^\bullet \Longrightarrow^* H(e)$ for $e \in E_R^N$ given by the context-freeness lemma. Hence we can assume $H = R[repl]$.

If H is vv'-connected, there is a path $p = v_0 e_1 v_1 \cdots e_n v_n$ connecting v and v'. Then either $e_i \in E_R^T$ or $e_i \in E_{repl(e)}$ for some $e \in E_R^N$ $(i = 1, \ldots, n)$. Replace now, for all $e \in E_R^N$, each longest subpath $v_{i(e)} e_{i(e)+1} \cdots e_{j(e)} v_{j(e)}$ of p in $repl(e)$ where $i(e) < j(e)$ with $v_{i(e)}\, e\, v_{j(e)}$. This yields a vv'-path in R such that $repl(e)$ is $v_{i(e)} v_{j(e)}$-connected for each $e \in E_R^N$ on this path. Conversely, if one has such paths in R and in the components $repl(e)$, one gets obviously a vv'-path in H.

This means that the vv'-connectedness of H can be checked by looking for a vv'-path in R and checking corresponding connectedness properties for some components of H. If v and v' are external nodes of H, one obtains a terminating recursive procedure because the components have shorter derivations than H. Moreover, connectedness needs to be considered for only a finite number of pairs of natural numbers because the external nodes of a hypergraph are ordered and their number is bounded by the order of the given grammar if the involved hypergraph is derived from a handle.

Definition 6.1 (*compatible predicate*) *Let $\mathcal{C} \subseteq \mathcal{HRG}$ and let I be a (possibly infinite) index set, such that there is an effective procedure constructing for every $HRG = (N, T, P, S) \in \mathcal{C}$ a finite subset I_{HRG} of I, together with some distinguished index $i(type(S)) \in I_{HRG}$. Let $PROP'$ be a decidable predicate defined on triples (R, ass, i), with $R \in \mathcal{H}_\mathcal{C}$, $ass\colon E_R^N \to I$ a mapping, and $i \in I$, and let $PROP(H, i) = PROP'(H, empty, i)$ for $H \in \mathcal{H}_\mathcal{T}$ and $i \in I$.*

*Then $PROP$ is $(\mathcal{C}, PROP')$-*compatible *if for every $HRG = (N, T, P, S) \in \mathcal{C}$, all derivations $A^\bullet \Longrightarrow R \Longrightarrow^* H$ with $H \in \mathcal{H}_\mathcal{T}$, and all $i \in I_{HRG}$ we have that $PROP(H, i)$ holds if and only if there is a mapping $ass\colon E_R^N \to I_{HRG}$ such that $PROP'(R, ass, i)$ holds and $PROP(H(e), ass(e))$ holds for all $e \in E_R^N$.*

*The unary predicate $PROP_0$ that holds on $H \in \mathcal{H}_\mathcal{T}$ with $type(H) = k$ if and only if $PROP(H, i(k))$ holds, is called $\mathcal{C}$-*compatible.

It is not difficult to see that compatible predicates are closed under boolean operations. In addition to the properties of hyperedge replacement languages discussed in Section 4, one can now show one more structural property in connection with compatible properties: The set of all hypergraphs from a hyperedge replacement language satisfying a compatible property is again a hyperedge replacement language.

Theorem 6.2 (filter theorem) *Let $PROP_0$ be a $\mathcal{C}$-compatible predicate for some $\mathcal{C} \subseteq \mathcal{HRG}$. For every $HRG \in \mathcal{C}$ there is a hyperedge replacement grammar HRG_{PROP_0} such that $L(HRG_{PROP_0}) = \{H \in L(HRG) \mid PROP_0(H)\}$.*

Proof. Let $HRG = (N, T, P, S)$ and let I_{HRG}, $PROP'$, and $i(type(S))$ be as in Definition 6.1. Then we construct a hyperedge replacement grammar $HRG' = (N', T, P', S')$ as follows.

- $N' = N \times I_{HRG}$;
- P' is the set of all pairs $((A, i), (R, ass))$ such that $(A, R) \in P$, $ass(e) \in I_{HRG}$ for $e \in E_R^N$, $i \in I_{HRG}$, and $PROP'(R, ass, i)$ holds, where (R, ass) denotes the hypergraph $(V_R, E_R, att_R, lab, ext_R)$ with $lab(e) = lab_R(e)$ if $lab_R(e) \in T$ and $lab(e) = (lab_R(e), ass(e))$ if $lab_R(e) \in N$;
- $S' = (S, type(S))$.

It remains to show $L(HRG') = \{H \in L(HRG) \mid PROP(H, type(S))\}$, which can be done by induction on the length of derivations in HRG and HRG' respectively, in a straightforward way. □

The Filter Theorem can be used to get several decidability results.

Theorem 6.3 (decidability of compatible properties) *Let $PROP_0$ be $\mathcal{C}$-compatible with respect to some class $\mathcal{C}$ of hyperedge replacement grammars. Then for all $HRG \in \mathcal{C}$, it is decidable whether*

1. *$PROP_0$ holds for some $H \in L(HRG)$;*
2. *$PROP_0$ holds for all $H \in L(HRG)$;*
3. *$PROP_0$ holds for no $H \in L(HRG)$ except perhaps a finite number;*
4. *$PROP_0$ holds for all $H \in L(HRG)$ except perhaps a finite number.*

Proof. By the Filter Theorem, for every hyperedge replacement grammar $HRG \in \mathcal{C}$, we can effectively construct hyperedge replacement grammars HRG_{PROP_0} and $HRG_{\neg PROP_0}$ generating the sets

$$L(HRG_{PROP_0}) = \{H \in L(HRG) \mid PROP_0(H)\}$$

and

$$L(HRG_{\neg PROP_0}) = \{H \in L(HRG) \mid \neg PROP_0(H)\},$$

respectively. Now $PROP_0$ holds for some hypergraph $H \in L(HRG)$ if and only if $L(HRG_{PROP_0})$ is not empty. $PROP_0$ holds for a finite number of hypergraphs $H \in L(HRG)$ if and only if $L(HRG_{PROP_0})$ is finite. $PROP_0$ holds for all $H \in L(HRG)$ if and only if $L(HRG_{\neg PROP_0})$ is empty. And $PROP_0$ holds for all $H \in L(HRG)$ except perhaps a finite number if and only if $L(HRG_{\neg PROP_0})$ is finite. As a consequence of the pumping lemma, it was noted above that the finiteness problem for hyperedge replacement languages is decidable. Furthermore, the emptiness problem is decidable using the same proof as in the string case. Altogether, these facts yield the claimed results. □

6.2 Compatible functions

We now turn to the discussion of compatible functions, which generalize compatible predicates. A function on hypergraphs is said to be compatible with the derivation process of hyperedge replacement grammars if it can be computed in the way a compatible predicate can be tested. We restrict the discussion to a certain type of compatible functions that are composed of minima, maxima, sums, and products on natural numbers. They induce compatible predicates of the form: the function value of a graph exceeds a given fixed integer, or the function value does not exceed a fixed integer. Consequently, we get the corresponding decidability results for these predicates as a corollary.

Given a function on hypergraphs (like the size, the maximum degree, the number of components, etc.) and considering the set of values for a hypergraph language, one may wonder whether this set is finite or not. The question is whether the language is bounded with respect to the given function. It can be shown that this boundedness problem is decidable for a class of hyperedge replacement grammars if the function is compatible and composed of sums, products and maxima of natural numbers.

In order to discuss this result, it is appropriate to enrich $\mathbb{N}$ by a special value $\diamond$. This additional element is useful because sometimes the functions one would like to consider have no sensible integer value for some arguments. For example, if we are interested in computing the shortest path between two external nodes of a hypergraph we have to take into account that there is perhaps no path at all between these nodes. Therefore, let $\mathbb{N}^\diamond = \mathbb{N} + \{\diamond\}$ with the following properties for every index set I and $n, n_i \in \mathbb{N}^\diamond$ for $i \in I$, where $I' = \{i \in I \mid n_i \neq \diamond\}$:

- $\diamond \leq n$,
- $\sum_{i \in I} n_i = \diamond$ and $\prod_{i \in I} n_i = \diamond$ if and only if $n_j = \diamond$ for some $j \in I$,
- $\min_{i \in I} n_i = \min_{i \in I'} n_i$ and $\max_{i \in I} n_i = \max_{i \in I'} n_i$, and
- $\min_{i \in I} n_i = \diamond$ and $\max_{i \in I} n_i = \diamond$ for $I = \emptyset$.

Now, the notion of a compatible function is defined as follows.

Definition 6.4 (*compatible function*)

1. *Let $\mathcal{C} \subseteq \mathcal{HRG}$ and let I be a (possibly infinite) index set, such that there is an effective procedure which constructs for every $HRG = (N, T, P, S) \in \mathcal{C}$ a finite subset I_{HRG} of I and a distinguished index $i(type(S)) \in I_{HRG}$. Let VAL be a set of values, f' be a function on triples (R, ass, i) with $R \in \mathcal{H}_{\mathcal{C}}$, $ass\colon E_R^N \times I \to VAL$, and $i \in I$, and let $f(H, i) = f'(H, empty, i)$ for $H \in \mathcal{H}_{\mathcal{T}}$ and $i \in I$.*

   Then f is $(\mathcal{C}, \{'\})$-compatible *if for all $HRG = (N, T, P, S) \in \mathcal{C}$, all derivations $A^\bullet \Longrightarrow R \Longrightarrow^* H$ with $H \in \mathcal{H}_{\mathcal{T}}$, and all $i \in I_{HRG}$, $f(H, i) = f'(R, ass, i)$, where $ass\colon E_R^N \times I \to VAL$ is given by $ass(e, j) = f(H(e), j)$ for $e \in E_R^N$ and $j \in I_{HRG}$.*

   The function $f_0\colon \mathcal{H}_{\mathcal{T}} \to VAL$ given by $f_0(H) = f(H, i(k))$ for all $H \in \mathcal{H}_{\mathcal{T}}$ with $type(H) = k$ is called $\mathcal{C}$-compatible.

2. *A function $f\colon \mathcal{H}_{\mathcal{T}} \times \mathcal{I} \to \mathbb{N}^\diamond$ is said to be* $(\mathcal{C}, \min, \max, +, \cdot)$-compatible *if there exists a function f' such that f is $(\mathcal{C}, \{'\})$-compatible and for each right-hand side R of some production in $\mathcal{C}$ and each $i \in I$, $f'(R, -, i)$ corresponds to an expression formed with variables $ass(e, j)$ ($e \in E_R^N$, $j \in I$) and constants from $\mathbb{N}$ by addition, multiplication, minimum, and maximum. The function f is* $(\mathcal{C}, \max, +, \cdot)$-compatible *if the operation* min *does not occur.*

   The function $f_0\colon \mathcal{H}_{\mathcal{T}} \to \mathbb{N}^\diamond$ given by $f_0(H) = f(H, i(k))$ for all $H \in \mathcal{H}_{\mathcal{T}}$ with $type(H) = k$ is $(\mathcal{C}, \min, \max, +, \cdot)$-compatible, *or* $(\mathcal{C}, \max, +, \cdot)$-compatible, *respectively.*

Remark. Let $f_0\colon \mathcal{H}_{\mathcal{T}} \to \mathbb{N}^\diamond$ be a $(\mathcal{C}, \min, \max, +, \cdot)$-compatible function, let $n \in \mathbb{N}^\diamond$, and let $\bowtie$ be one of the binary predicates $\leq$, $<$, $=$, $\geq$, and $>$. Then it is not difficult to see that the predicate $PROP(f_0, \bowtie)$ given for $H \in \mathcal{H}_{\mathcal{T}}$ by $PROP(f_0, \bowtie)(H) \iff f_0(H) \bowtie n$ is $\mathcal{C}$-compatible. This together with the decidability results of the previous section yields the solution of some particular decision problems. For all $HRG \in \mathcal{C}$ it is decidable whether

- $f_0(H) \bowtie n$ holds for some $H \in L(HRG)$;
- $f_0(H) \bowtie n$ holds for only a finite number of hypergraphs $H \in L(HRG)$.
- $f_0(H) \bowtie n$ holds for all $H \in L(HRG)$.
- $f_0(H) \bowtie n$ holds for all $H \in L(HRG)$ except a finite number.

As an example, let us have a look at the function that yields the number of simple paths of a 2-graph connecting the two external nodes. Given a graph G, a path is called *simple* if each node appears only once. Let $PATH_G$ denote the set of all simple $begin_G end_G$-paths of G and let $numpath(G) = |PATH_G|$ be its cardinality. Using the Context-freeness Lemma, if $A^\bullet \Longrightarrow R \Longrightarrow^* G$ we get $PATH_G = \bigcup_{p \in PATH_R} \{replace(p, path) \mid path\colon E_R^N \to PATH_{G(e)}\}$, where $replace(p, path)$ denotes the path obtained from p by replacing all edges $e \in E_R^N$ on p by the corresponding paths $path(e)$. This obviously yields

$$numpath(G) = \sum_{p \in PATH_R} \prod_{e \in E_R^N \text{ on } p} numpath(G(e)).$$

These observations can be reformulated in terms of compatible functions. Let $\mathcal{C}$ be the class of all edge replacement grammars, $I = \{np\}$, $VAL = \mathbb{N}^\diamond$, and f' be the function given by $f'(R, ass, np) = \sum_{p \in PATH_R} \prod_{e \in E_R^N \text{ on } p} ass(e, np)$. Then we have $f(G, np) = f'(G, empty, np) = \sum_{p \in PATH_G} 1 = numpath(G)$ and $f(G, np) = f'(R, ass, np)$ with $ass(e, np) = f(G(e), np)$ $(e \in E_R)$, so f turns out to be $(\mathcal{ERG}, \{'\})$-compatible. Consequently, the function $numpath = f(-, np)$ is $(\mathcal{ERG}, \max, +, \cdot)$-compatible.

Other compatible functions are the number of nodes, the size, the density of a hypergraph, that is, the ratio of the number of edges and the number of nodes, the minimum-path length (of paths connecting external nodes), the maximum-simple-path length (of paths connecting external nodes), the number of simple cycles, the minimum-cycle length, the maximum-simple-cycle length, the minimum degree, the maximum degree and the number of components.

Theorem 6.5 (meta-theorem for boundedness problems) *Let f_0 be a $(\mathcal{C}, \max, +, \cdot)$-compatible function for a class $\mathcal{C}$ of hyperedge replacement grammars. Then, for all $HRG \in \mathcal{C}$, it is decidable whether there is a natural number $n \in \mathbb{N}$ such that $f_0(H) \leq n$ for all $H \in L(HRG)$.*

Proof. Let f, f' be the required functions, so that f is $(\mathcal{C}, \{'\})$-compatible and $f_0(H) = f(H, i(type(H)))$ for $H \in \mathcal{H}_\mathcal{T}$. Let $HRG = (N, T, P, S) \in \mathcal{C}$. Then one can show that f_0 is unbounded on $L(HRG)$ if and only if there are $A \in N$, $j \in I_{HRG}$ and A-handled hypergraphs X and Y with $X_0, Y_0 \in \mathcal{H}_\mathcal{T}$ such that the following hold:

1. $A^\bullet \Longrightarrow^* H$ for some $H \in \mathcal{H}_\mathcal{T}$.
2. $S^\bullet \Longrightarrow^* Y$ such that $f(H, j) \leq f(Y \otimes H, i_0) = f_0(Y \otimes H)$ for all $H \in \mathcal{H}_\mathcal{T}$ with $A^\bullet \Longrightarrow^* H$.
3. $A^\bullet \Longrightarrow^* X$ such that $f(H, j) < f(X \otimes H, j)$ for all $H \in \mathcal{H}_\mathcal{T}$ that satisfy $A^\bullet \Longrightarrow^* H$.

To check for such A, j, X and Y, one has to inspect the finite number of derivations that start in handles and are of length up to the number of nonterminals. The somewhat tedious technical details are omitted. $\square$

6.3 Further results and bibliographic notes

Results which are closely related to those discussed in this section are surveyed in Section 6 of Chapter [13] (see also the remarks on inductive predicates and inductively computable functions below). The results about compatibility are taken from [56,57] by Habel, Kreowski, and Vogler. It must be mentioned, however, that a finite index set was used in these papers. For the case of compatible predicates an infinite index set was first used by Habel, Kreowski, and Lautemann in [55].

The Filter Theorem is published in [53,48]. A corresponding result can be found in [11]. The statements (1) and (2) of the decidability result are published in [56]. In that paper, a direct proof is given which is based on the idea of constructing the set of handles from which a hypergraph with the desired property can be derived and checking whether the start handle belongs to the constructed set. The statements (3) and (4) of the decidability result are presented in [53,48]. In [69] Lautemann shows how these and several related results in connection with tree decompositions and bounded tree-width can by systematized by the use of finite tree automata and well-known characterizations of recognizability. Some of the methods for finite tree automata are carried over to deal with certain tree automata with infinite state sets. In this way it is shown that graph properties defined by formulae of monadic second order logic with arithmetic can be decided efficiently for graphs of bounded tree-width, a result which was first shown (in somewhat more general form) by Arnborg, Lagergren, and Seese [3].

In [72], Lengauer and Wanke consider efficient ways of analyzing graph languages generated by so-called cellular graph grammars. Cellular graph grammars are in fact hyperedge replacement graph grammars, defined in a slightly different way. In particular, they generate the same class of graph languages. A characteristic of graph properties called finiteness is defined, and combinatorial algorithms are presented for deciding whether a graph language generated by a given cellular graph grammar contains a graph with a given finite graph property. Structural parameters are introduced that bound the complexity of the decision procedure and special cases for which the decision can be made in polynomial time are discussed. The results provide explicit and efficient combinatorial algorithms.

In [11], Courcelle introduces the notion of a recognizable set of graphs.

Every set of graphs definable in monadic second-order logic is recognizable, but not vice versa. It is shown that the monadic second-order theory of hyperedge replacement languages is decidable. The notion of F-inductive predicates studied in [11] is closely related to the concept of compatible properties (see also Section 6.2 of Chapter [13]).

In [55], Habel, Kreowski, and Lautemann compare compatible, finite, and inductive graph properties and show that the three notions are essentially equivalent. Consequently, three lines of investigation in the theory of hyperedge replacement—including the one discussed here—merge into one.

In [85], Wanke and Wiegers investigate the decidability of the bandwidth problem on linear hyperedge replacement languages. In particular, they show the following. Let $\mathcal{HRG}_{\mathrm{lin}}$ denote the class of all linear hyperedge replacement grammars generating graphs. Then it is undecidable whether an instance grammar $HRG \in \mathcal{HRG}_{\mathrm{lin}}$ generates a graph G having bandwidth k, for any fixed integer $k \geq 3$. The result implies that the bandwidth-k property for $k \geq 3$ is not $\mathcal{HRG}_{\mathrm{lin}}$-compatible.

Theorem 6.5 is published in [57]. Related work is done in [16] by Courcelle and Mosbah, who investigate monadic second-order evaluations on tree-decomposable graphs and come up with a general method to translate these evaluations of graph expressions over suitable semirings. Their method allows the derivation of polynomial algorithms for a large number of problems on families of graphs, and especially on graphs definable by hyperedge replacement. The notion of inductively computable functions introduced by Courcelle and Mosbah ([16], see also Section 6.3 of Chapter [13]) is a generalization of the concept of inductive predicates which corresponds closely to the notion of compatible functions.

Boundedness problems for hyperedge replacement grammars correspond closely to boundedness problems for finite tree automata with cost functions over a suitable semiring, investigated by Seidl in [80]. Cost functions for tree automata are mappings from transitions to polynomials over some semiring or, in the so-called k-dimensional case, to k-tuples of polynomials. (The dimensions correspond to the different indices used in the definition of a compatible function.) Four semirings are considered, namely the semiring $\mathbf{N}$ over $\mathbb{N}$ with addition and multiplication, the semiring $\mathbf{A}$ over $\mathbb{N} \cup \{\infty\}$ with maximum and addition, the semiring $\mathbf{T}$ over $\mathbb{N} \cup \{-\infty\}$ with minimum and addition, and the semiring $\mathbf{F}$ over the finite subsets of $\mathbb{N}$ with union and addition. It turns out that for the semirings $\mathbf{N}$ and $\mathbf{A}$, it is decidable in polynomial time (if the dimension k is fixed) whether or not the costs of accepting computations is bounded; for $\mathbf{F}$, it is decidable in polynomial time whether or not the cardinalities of occurring cost sets are bounded. In all three cases, explicit upper bounds are derived. Moreover, for the semiring $\mathbf{T}$, the decidability of boundedness is proved, but a polynomial-time algorithm is obtained only in the case that the degrees of occurring polymonials are at most 1.

In [84], Wanke considers another type of decidability problems on hyperedge replacement grammars, called integer subgraph problems. The main idea is to transform the decidability problem on hyperedge replacement languages concerning function values of graph$\times$subgraph pairs to a decidability problem over semilinear sets.

Finally, one should mention an approach to generalize the idea of compatible functions that was invented by Engelfriet and Drewes [32,24,23,25,22]. The approach is based on the observation that the definition of a compatible function can be considered as a tree transduction. The input trees are derivation trees of the hypergraphs in question and the output trees are expressions using maximum, addition, and multiplication (viewed as trees), that denote the computed numbers. Both the derivation trees (denoting hypergraphs) and the expressions (denoting natural numbers) may be viewed as terms in appropriate algebras (over hypergraphs and natural numbers, respectively). Thus, the idea is to consider such computations by tree transductions in general. A mapping $f\colon A \to B$ between two algebras A and B is said to be computed by a tree transducer if every term denoting an element a of A is transformed into a term denoting $f(a)$ in B.

7 The membership problem

In this section the possibilities for (efficient) algorithmic solutions to the *membership problem* for hyperedge replacement languages are discussed. We consider membership with respect to a given hyperedge replacement grammar HRG, so the membership problem is defined as follows.

Given: A hyperedge replacement grammar HRG.
Instance: A hypergraph H.
Question: Is H in the language generated by HRG?

Algorithms that solve the membership problem are called membership algorithms. In many cases, one would also wish to *parse* hypergraphs with respect to a given grammar, that is, to find a derivation tree for a given hypergraph, if possible. For technical convenience, we concentrate on the membership problem here, but it is easy to see that both the presented algorithms can be extended in a straightforward way to yield solutions to the parsing problem as efficiently as they solve the membership problem.

7.1 NP-completeness

It is not hard to see that the membership problem is in NP for every hyperedge replacement grammar HRG, because for every such grammar there is a linear function f such that, for every hypergraph H of size n, if $H \in L(HRG)$ then H has a derivation of length at most $f(n)$. Thus we can simply use nondeterminism to guess an appropriate derivation and test whether the generated hypergraph is isomorphic to H (again using nondeterminism).

Clearly, one would like to have more efficient algorithms than the one just sketched, and in particular deterministic ones. It turns out, however, that the complexity of the membership problem marks one of the few areas where the results for hyperedge replacement grammars differ significantly from what we know about context-free Chomsky grammars. Whereas, for the latter we have the well-known membership algorithm by Cocke, Kasami, and Younger, which runs in cubic time, hyperedge replacement grammars are able to generate NP-complete graph languages. Thus, one can expect that there is no generally applicable membership algorithm for hyperedge replacement languages. Intuitively, the reason for this is that, in a string, symbols that result from one and the same nonterminal form a substring, and there are only $O(n^2)$ many substrings of a given string of length n. In contrast, from a nonterminal hyperedge a somewhat disconnected subgraph may be generated in a hyperedge replacement grammar, so that, in general, there are exponentially many possible choices for a subgraph to be generated from a given hyperedge. There are simply too many combinations to be tested, which leads to the two NP-completeness results presented below. For this, let us call a hyperedge replacement grammar *linear* if none of its right-hand sides has more than one nonterminal hyperedge. The *node degree* of a graph is the maximal number of edges attached to a node of this graph, and the node degree of a graph languages is the maximum node degree of graphs in that language.

Theorem 7.1 (NP-completeness 1) *There is a linear edge replacement grammar that generates an NP-complete graph language of degree 2.*

Proof. We reduce the NP-complete *Hamiltonian path problem* (see Garey, Johnson [45], problem GT39, p. 199) to the membership problem for a particular edge replacement language generated by a linear grammar. In fact, there is a slightly easier way to prove Theorem 7.1 using the NP-complete problem 3-PARTITION (see [1]). However, we are going to use a modified version of the proof in order to obtain another theorem below. As we do not know how this can be done using 3-PARTITION the Hamiltonian path problem is employed here.

Let us first define the Hamiltonian path problem. We use a slightly more general version on undirected hypergraphs. An undirected, unlabelled hypergraph is a triple $h = (V_h, E_h, att_h)$, where V_h and E_h are the finite sets of vertices and hyperedges, respectively, and att_h is a mapping assigning to every hyperedge $e \in E_h$ the set $att_h(e) \subseteq V_h$ of attached nodes. Now, the Hamiltonian path problem can be formulated as follows.

Instance: An undirected, unlabelled hypergraph h.
Question: Is there a Hamiltonian path in h, that is, is there an alternating sequence $v_0 e_1 v_1 \dots e_n v_n$ of pairwise distinct nodes and hyperedges of h such that $V_h = \{v_0, \dots, v_n\}$ and $v_{i-1}, v_i \in att_h(e_i)$ for all i, $1 \le i < n$?

Clearly, the generalization to hypergraphs does not affect the NP-hardness of the problem. Let us define $\mathcal{S}$ to be the set of all finite sets $S = \{s_1, \dots, s_n\}$ such that the s_i $(1 \le i \le n)$ are strings of equal length of the form u_i-u_i, where $u_i \in \{0,1\}^*$. We say that u_i-u_i is *word adjacent* to u_j-u_j if $u_i(l) = u_j(l)$ for some l, $1 \le l \le |u_i|$.

Every $S \in \mathcal{S}$ defines an undirected, unlabelled hypergraph $h(S)$. If S is as above, with $|s_i| = 2m+1$ for $i = 1, \dots, n$, the set of nodes of $h(S)$ is S and the set of hyperedges is $\{1, \dots, m\}$. Node s_i is in the set of attachments of hyperedge j if $u_i(j) = 1$. By definition, two nodes are adjacent in $h(S)$ if the corresponding strings are word adjacent, so $h(S)$ has a Hamiltonian path if and only if there is a sequence $i_1, \dots, i_n$ with $\{i_1, \dots, i_n\} = \{1, \dots, n\}$ such that, for $i = 1, \dots, n-1$, s_i and s_{i+1} are word adjacent. Clearly, every undirected, unlabelled hypergraph h is represented by some $S \in \mathcal{S}$. Furthermore, given any sensible representation of h we can compute some $S_h \in \mathcal{S}$ with $h = h(S_h)$ in polynomial time.

We are going to construct a linear edge replacement grammar such that each of the generated graphs is a string graph, except that some of the edges are missing. Thus, such a graph represents a multiset of strings. The individual strings of the multisets generated have the form w-w', where $w, w' \in \{0,1\}^*$. The multisets themselves will be of the form

$$\begin{aligned}\{&w_0\text{-}w_{11}1w_{12},\\ &\quad w'_{11}1w'_{12}\text{-}w_{21}1w_{22},\\ &\qquad\quad w'_{21}1w'_{22}\text{-}w_{31}1w_{32},\\ &\qquad\qquad\qquad\ddots\\ &\qquad\qquad\qquad\quad w'_{n1}1w'_{n2}\text{-}w_1\}\end{aligned}$$

where each w_{i1} is generated along with the corresponding w'_{i1}, and each w_{i2} is generated together with the corresponding w'_{i2}. This is used to ensure that w_{ij} and w'_{ij} are of equal length, so that the digit 1 between w_{i1} and w_{i2} appears at the same position as the one between w'_{i1} and w'_{i2}. Thus, if such a multiset S is in $\mathcal{S}$ it defines a hypergraph with a Hamiltonian path passing the nodes

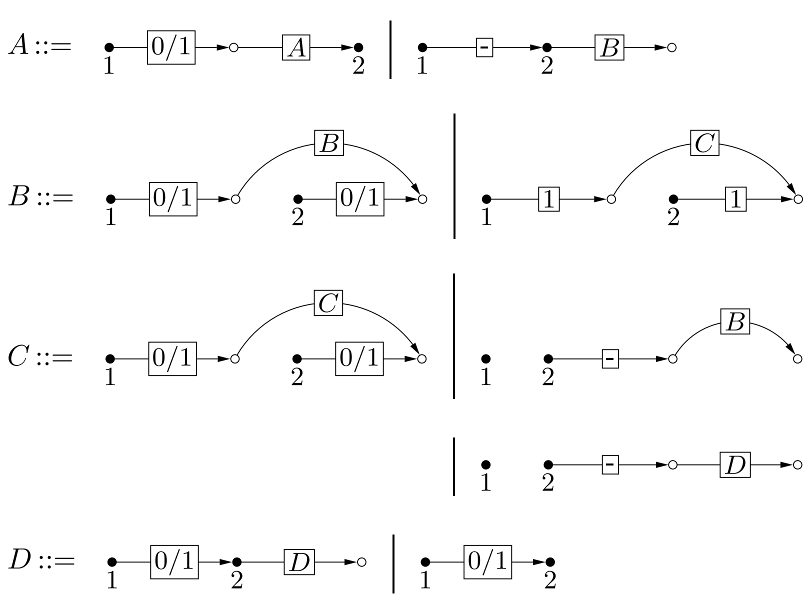

Figure 17: Productions of *ERG* in the proof of Theorem 7.1.

represented by the strings above one after the other.

Consider the grammar $ERG = (\{S, A, B, C, D\}, \{\text{-}, \mathit{0}, \mathit{1}\}, P, S)$, where all terminals and nonterminals are of type 2, except for S, which is of type 0, and where P consists of the productions given in Figure 17. (In Figure 17, a right-hand side some of whose edges are labelled with "0/1" stands for all right-hand sides in which the respective edges are labelled with 0 or 1.) If we interpret the graphs in $L(ERG)$ as multisets of strings in the obvious way, this yields a set $\mathcal{S}(ERG)$ whose elements are these multisets. It is not too hard to see that, as explained above, all these multisets are of the form

$$\{w_0\text{-}w_{11}1w_{12},\ w'_{11}1w'_{12}\text{-}w_{21}1w_{22},\ w'_{21}1w'_{22}\text{-}w_{31}1w_{32},\ \ldots,\ w'_{n1}1w'_{n2}\text{-}w_1\}.$$

The w_{i1} and w'_{i1} are generated using the nonterminal B, and the w_{i2} and w'_{i2} result from a nonterminal C. As a consequence, w_{ij} and w'_{ij} are indeed of equal length, so that the 1 between w_{i1} and w_{i2} appears in the same position as the one between w'_{i1} and w'_{i2}. Therefore, one can show that the following holds:

$\mathcal{S}(ERG) \cap \mathcal{S}$ is the set of all $\{s_1, \ldots, s_n\} \in \mathcal{S}$ for which there is an ordering $i_1, \ldots, i_n$ of $1, \ldots, n$ such that s_{i_j} is word adjacent to $s_{i_{j+1}}$ for all i, $1 \leq i < n$.

Clearly, for all $S \in \mathcal{S}$ some graph $H(S)$ representing S in the same way as above can be constructed in polynomial time. Now, $H(S) \in L(ERG)$ if and only if $h(S)$ has a Hamiltonian path. We have thus reduced the Hamiltonian path problem to the membership problem for $L(ERG)$, as required. $\square$

If we drop the requirement of bounded degree in Theorem 7.1 we can even generate an NP-complete language of connected graphs, as one can see by an easy modification of the grammar used.

Theorem 7.2 (NP-completeness 2) *There is an edge replacement grammar that generates an NP-complete graph language of connected graphs.*

Proof. We construct an edge replacement grammar ERG' that generates for every graph $H \in L(ERG)$, where ERG is the edge replacement grammar in the proof of Theorem 7.1, a graph obtained from H as follows. First, a new node v_0 is added. Then, every edge e labelled - is removed and its two attached nodes are identified, yielding a new node v_e. Finally, for each v_e a new unlabelled edge with attachment $v_0 v_e$ is added. The correspondence between the graphs in $L(ERG)$ and their counterparts in $L(ERG')$ constructed this way is a bijection that can be computed in polynomial time. Therefore, it remains to give an appropriate grammar ERG'. We may choose for this the grammar $ERG' = (\{S, A, B, C, D\}, \{\mathit{0}, \mathit{1}\}, P', S)$, where the types of symbols are as in ERG and the productions are the ones given in Figure 18. In order to understand how the grammar works, the reader should notice that, applying in a first phase only the productions with left-hand sides S, A, and B yields a hypergraph as in Figure 19. Now, it is easy to see that the remaining productions yield just the type of graphs aimed at. $\square$

7.2 Two polynomial algorithms

In view of Theorems 7.1 and 7.2 there is not much hope that one can find an efficient solution to the membership problem for arbitrary hyperedge replacement grammars (and not even for linear edge replacement grammars), since this would require P=NP. It is therefore natural to look for restrictions that can be imposed on the languages or grammars to allow for efficient membership algorithms. The proofs of Theorem 7.1 and 7.2 give a hint. It seems that, in the first case, the high complexity of the generated language is caused by the fact that the graphs are disconnected, whereas in the the second case the potentially unbounded degree of nodes seems to be responsible. In fact, the

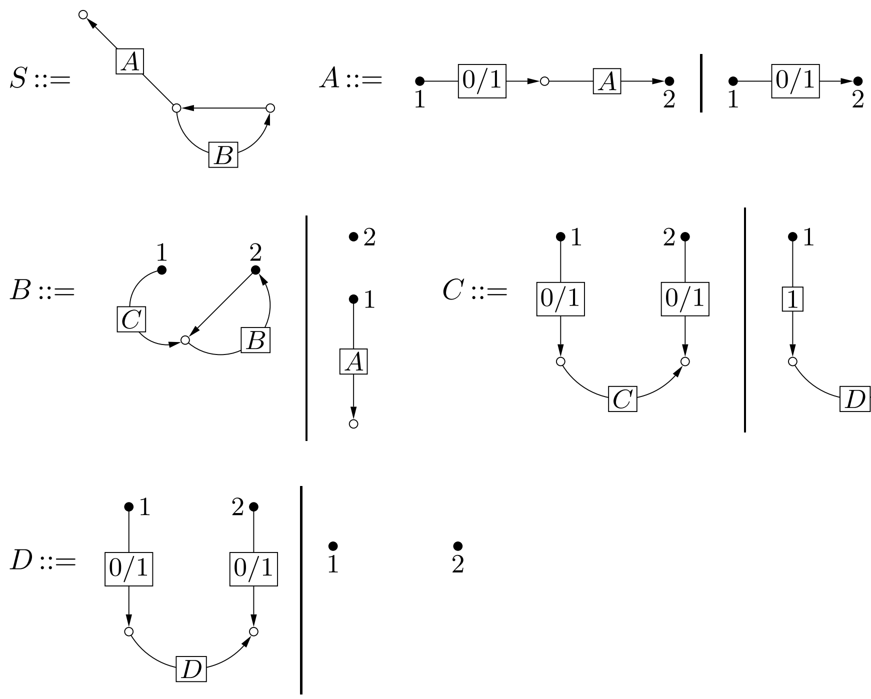

Figure 18: The productions of ERG' in the proof of Theorem 7.2.

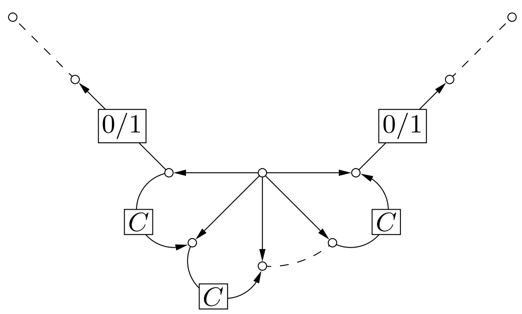

Figure 19: An intermediate graph generated by the productions in Figure 18.

two cases do not differ too much: The unbounded degree in the second case is nothing else than a hidden disconnectedness that reappears if a single node is deleted. Later on, this observation will lead to the notion of k-separability, which is needed to formulate a condition under which the first membership algorithm we will discuss runs in polynomial time. As opposed to the string case both algorithms we are going to present rely on the fact that the grammar considered is supposed to be fixed. It is easy to see that the running time of these algorithms becomes exponential if the grammar is made part of the input.

In the following, we use the notation $H = H_0{<}e_1/H_1, \dots, e_n/H_n{>}$ to express the fact that $H = H_0[e_1/H_1, \dots, e_n/H_n]$, where $V_H = V_{H_0} \cup \dots \cup V_{H_n}$, $E_H = E_{H_0} \cup \dots \cup E_{H_n}$, $ext_H = ext_{H_0}$, and $lab_H(e) = lab_{H_i}(e)$, $att_H(e) = att_{H_i}(e)$ for all $e \in E_H \cap E_{H_i}$, $0 \le i \le n$. In other words, $H_0{<}\cdots{>}$ is hyperedge replacement on *concrete* hypergraphs without the possibility to take isomorphic copies (and is hence not always defined). The notion of X-candidates is central to the first algorithm we are going to present.

Definition 7.3 (X*-candidate) Let H be hypergraph and let $X \in V_H^*$. A hypergraph $H' \subseteq H$ is an X-*candidate *of H if $ext_{H'} = X$ and for every internal node v of H' and every hyperedge $e \in E_H$ with $v \in [att_H(e)]$ we have $e \in E_{H'}$.*

By definition of hyperedge replacement, every hyperedge of a hypergraph $H_0{<}e_1/H_1, \dots, e_n/H_n{>}$ that is attached to an internal node of H_i for some i, $1 \le i \le n$, is a hyperedge of H_i. This can be used to prove that, in $H_0{<}e_1/H_1, \dots, e_n/H_n{>}$, every H_i ($1 \le i \le n$) is an $att_{H_0}(e_i)$-candidate of H for $i = 1, \dots, n$. This fact restricts the number of sub-hypergraphs to be tested recursively in the membership algorithm below. We formulate the algorithm for growing hyperedge replacement grammars. By Lemma 4.3 this means no loss of generality.

Algorithm 7.4
Given: *A growing hyperedge replacement grammar* $HRG = (N, T, P, S)$.
Input: *A hypergraph* $\bar{H}$ *over* T.
Output: $Derive(S, \bar{H})$, *where* $Derive(A, H) =$

(1) *if* $A^\bullet \Longrightarrow R$ *with* $V_R \subseteq V_H$ *and* $E_R^T \subseteq E_H$
(where $\{e_1, \dots, e_n\} = E_R^N$*)*
(2) *such that there are substantial* $att_R(e_i)$*-candidates* H_i *of* $\bar{H}$
$(i = 1, \dots, n)$ *with*
(3) *1.* $H = R{<}e_1/H_1, \dots, e_n/H_n{>}$
(4) *2. for* $i = 1, \dots, n$ *we have* $Derive(lab_R(e_i), H_i)$
(5) *then return* ***true*** *else return* ***false***

Theorem 7.5 *For all hypergraphs $\bar{H} \in \mathcal{H}_{\mathcal{T}}$ Algorithm 7.4 terminates and yields **true** if $\bar{H} \in L(HRG)$, and **false** otherwise.*

Proof. Since HRG is growing all hypergraphs that can be derived from $A^\bullet$ are substantial. Therefore, Algorithm 7.4 is nothing else than an algorithmic formulation of the context-freeness lemma. Since the hypergraph R chosen in step (1) and the $att_R(e_i)$-candidates H_i of step (2) are substantial we have $|H_i| < |H|$ for $i = 1, \ldots, n$, so the algorithm must eventually terminate. $\square$

Since there is only a finite set of productions in HRG there are only polynomially many possibilities to consider in step (1), and the equality in step (3) can also be tested in polynomial time (note that the test is for equality, not isomorphy). Hence, there is only a single point that causes an exponential running time of the algorithm: the number of X-candidates to be tested. Therefore, we aim at a condition to be imposed on $L(HRG)$ that implies a polynomial upper bound on the number of X-candidates. Let us denote by $H|_V$, where H is a hypergraph and $V \subseteq V_H$, the hypergraph $(V, E_H, att, lab_H, \lambda)$, where $att(e)$ is the restriction of $att_H(e)$ to V, for all $e \in E$. A *connected component* of H is a maximal connected sub-hypergraph of H. (Note that a 0-edge is a connected component on its own.)

Definition 7.6 (k-*separability) For $k \in \mathbb{N}$ the k-separability k-$sep(H)$ of a hypergraph H is the maximum number of connected components of $H|_V$, where V ranges over all subsets of V_H of size at most k. For every language L of hypergraphs, k-$sep_L \colon \mathbb{N} \to \mathbb{N}$ is defined by*

$$k\text{-}sep_L(n) = \max\{k\text{-}sep(H) \mid H \in L \text{ and } |H| \leq n\}.$$

The following theorem says that logarithmic k-separability of $L(HRG)$ (where HRG is of order k) results in a polynomial upper bound on the running time of Algorithm 7.4.

Theorem 7.7 *Let $HRG \in \mathcal{HRG}_{\|}$ be growing. If k-$sep_{L(HRG)} \in O(\log n)$ then Algorithm 7.4 can be implemented to run in polynomial time in $|\bar{H}|$.*

Proof. Let $\bar{H} \in \mathcal{H}_{\mathcal{T}}$ and $X \in V_{\bar{H}}^*$ with $|X| \leq k$. By definition of X-candidates, if a node of a connected component G of $\bar{H}|_{[X]}$ occurs in an X-candidate H of $\bar{H}$, then $G \subseteq H$. Since there are only a logarithmic number of connected components in $\bar{H}|_{[X]}$ this yields a polynomial bound on the number of X-candidates in $\bar{H}$ if $\bar{H} \in L(HRG)$. Since the number of sequences $X \in V_{\bar{H}}^*$ with $|X| \leq k$ is bounded by $|V_{\bar{H}}|^k$ this yields a polynomial bound on the number of X-candidates to be considered in the algorithm (that is, if there are more X-candidates in the input graph it can be rejected immediately).

We are now able to make use of the well-known idea underlying the membership algorithm for context-free Chomsky grammars by Cocke, Kasami, and

Figure 20: H is almost 2-connected while H' is not.

Younger. In an initial phase, we compute the set of all X-candidates of $\bar{H}$, where $|X| \leq k$. This can be done in polynomial time since it essentially suffices to determine all combinations of connected components of the hypergraphs $\bar{H}|_V$, where $|V| \leq k$. Now, every call to *Derive* with parameters A and H is performed only once and the result is stored in a list which is looked up when repeated calls occur. As a consequence, the algorithm runs in polynomial time in the number of X-components, which is polynomial in $\left|\bar{H}\right|$, as required. $\square$

The major drawback of Algorithm 7.4 is that, even if $k\text{-}sep_{L(HRG)}$ is a constant, the running time is bounded by a polynomial whose degree may vary with *HRG*. In view of step (1) of the algorithm, for instance, the degree depends on the size of right-hand sides used in the grammar. We now want to discuss a membership algorithm for edge replacement grammars that always runs in cubic time. It applies to all input graphs that are *almost 2-connected*, a notion to be defined below. Again, the algorithm exploits the idea by Cocke, Kasami, and Younger.

From now on, let us consider some arbitrary (but fixed) edge replacement grammar (N, T, P, S) with $type(A) = 2$ for all $A \in N \cup T$. (In particular, every graph is assumed to be of type 2 in the following, without mentioning.)

Definition 7.8 (*almost k-connected hypergraph) A hypergraph H is* almost k-connected *for some $k \in \mathbb{N}$ if*

- $|V_H| \geq k$ *and*
- *for all hypergraphs H_1, H_2 and every hyperedge $e \in E_{H_1}$, if we have $H = H_1[e/H_2]$ and $V_{H_1} \setminus [att_{H_1}(e)] \neq \emptyset \neq V_{H_2} \setminus [ext_{H_2}]$ then $type(H_2) \geq k$.*

Intuitively, almost k-connectedness means that a graph cannot be decomposed into non-trivial parts using hypergraphs of type less than k. The graph H shown in Figure 20, for example, is almost 2-connected, but H' is not.

A substantial graph H (of type 2) is a *bond* if $V_H = ext_H$, that is, if H consists of at least two edges between the external nodes. A substantial string graph is said to be a *chain*. H is called a *block* if it is almost 3-connected,

$|V_H| > 3$, and $[att_H(e)] \neq [ext_H]$ for all $e \in E_H$. Notice that bonds, chains, and blocks are substantial and that these three classes of graphs are mutually disjoint. The smallest bonds and chains are those with two edges. We have the following theorem about almost 2-connected graphs, that we state without proof.

Lemma 7.9

1. *Let $H = H_0{<}e/H_1{>}$ for substantial graphs H_0 and H_1. Then H is almost 2-connected if and only if both H_0 and H_1 are almost 2-connected.*

2. *Every substantial 2-connected graph H can be written as $H_0{<}e/H_1{>}$, for substantial graphs H_0, H_1, unless H is a bond or chain with two edges, or a block.*

In the following, let us reserve a label τ to be used as a special one in some constructions, that is, we assume $\tau \notin N \cup T$. For the algorithm we are going to explain the following notions are of basic importance.

Definition 7.10 (*total and collapsed split tree*) *Let H be a hypergraph over T.*

1. *A* total split tree *of H is a derivation tree for H over P_{t}, where P_{t} is the set of all productions over $\{\tau\}$ whose right-hand sides are bonds and chains with two edges, and blocks.*

2. *Let P_{c} be the set of all productions over $\{\tau\}$ whose right-hand sides are bonds, chains, or blocks. A* collapsed split tree *of H is a derivation tree for H over P_{c} such that no edge in this derivation tree connects two bonds or two chains.*

By definition, the result of a (total or collapsed) split tree t is a substantial, almost 2-connected graph. Since the left-hand side of productions in a split tree is always τ we may consider a split tree as a pair $(H, branch)$ rather than as a triple $(\tau, H, branch)$. Using Lemma 7.9 it is not hard to see that every substantial, almost 2-connected graph has some total split tree. From a total split tree one can obtain a collapsed one as follows: As long as a subtree $(H, branch)$ exists with $branch(e) = (H', branch')$ for some $e \in E^N_H$, where both H and H' are bonds or both are chains, replace the subtree by $(H[e/H'], branch'')$, where

$$branch''(e') = \begin{cases} branch(e') & \text{if } e' \in E^H_N - \{e\} \\ branch'(e') & \text{if } e' \in E^{H'}_N. \end{cases}$$

Then $H[e/H']$ is a bond (chain, respectively), so the procedure eventually leads to a collapsed split tree. As a consequence, every substantial, almost 2-connected graph has a total as well as a collapsed split tree, and for every (total or collapsed) split tree t the graph *result*(t) is substantial and almost 2-connected.

The algorithm we are going to present makes use of a result by MacLane [73] saying that collapsed split trees are unique in a certain sense. It is easy to see that collapsed split trees cannot be really unique. This is because, if H and H' are graphs with $e \in E_H$, and we let G and G' be obtained from them by reversing e in H and interchanging the external nodes of H' we obviously obtain $H[e/H'] = G[e/G']$. Hence, for split trees we must allow to reverse nonterminal edges if at the same time the external nodes of the graphs that replace them are interchanged. This is stated more precisely below.

Definition 7.11 (*similar split trees*) *Let* $t = (H, branch)$ *and* $t' = (H', branch')$ *be split trees with* $E_H^N = \{e_1, \ldots, e_n\}$ *and, for* $i = 1, \ldots, n$, $branch(e_i) = (H_i, branch_i)$. *Then* t *and* t' *are* similar *if there is some* $I \subseteq \{1, \ldots, n\}$ *such that the following hold.*

1. *The graph obtained from* H *by reversing all* e_i *with* $i \in I$ *is isomorphic to* H' *via some isomorphism* h.
2. *For all* $i \in I$, $(H_i', branch_i)$ *and* $branch(h_E(e_i))$ *are similar, where* H_i' *is obtained from* H_i *by reversing* ext_{H_i} *if* $i \in I$ *and* $H_i' = H_i$ *otherwise.*

As one can easily show by induction, similarity of t and t' implies that *result*(t) and *result*(t') are isomorphic. The algorithm we are developing is based on the following result by MacLane [73].

Fact 7.12 (uniqueness of collapsed split trees) *All collapsed split trees of a substantial, almost 2-connected graph are similar.*

Using Theorem 7.12 we want to prove the following.

Theorem 7.13 *Let ERG be an edge replacement grammar. Then there is a cubic algorithm that takes as input a graph* H *and decides whether* H *is an almost 2-connected member of* $L(ERG)$. *In particular, if all graphs in* $L(ERG)$ *are almost 2-connected, the algorithm decides whether* $H \in L(ERG)$.

We are now going to develop the means to prove Theorem 7.13. Let us first assume some edge replacement grammar $ERG = (N, T, P, S)$ as in the theorem is given. By Lemma 4.3 it may be assumed that *ERG* is growing. We modify *ERG* as follows.

(1) Remove all productions whose right-hand side is not almost 2-connected. By Lemma 7.9 we are allowed to do so because this modification does not affect the set of almost 2-connected graphs in the language generated.

(2) As long as there is a production (A, R) such that $H = R[e/H']$ for some substantial H and H', choose a new nonterminal label A' for e in H and replace (A, R) by the two productions (A, H) and (A', H'). Obviously, such a modification does not influence the generated language. By Lemma 7.9 the right-hand sides of the grammar obtained are bonds and chains with two edges, and blocks.

(3) Complete the grammar as follows: For every label A (nonterminal as well as terminal ones), add a new label $\bar{A}$, and add for all productions (A, R) in the grammar obtained in step (2), all productions (X, H), where $X \in \{A, \bar{A}\}$ and H is obtained from R by reversing some edges while changing their label from $lab_R(e)$ to $\overline{lab_R(e)}$, and reversing ext_R if $X = \bar{A}$.

Let the edge replacement grammar resulting from steps (1)–(3) be given by $ERG_0 = (N_0, T_0, P_0, S)$. Then it follows by a straightforward induction that the set of graphs over T generated by ERG_0 coincides with the one of the grammar obtained by steps (1) and (2), that is, $L(ERG_0) = L(ERG)$. For every derivation tree t over P_0^* (see page 20 for the definition of P_0^*) let us denote by $unlabel(t)$ the derivation tree we obtain by replacing all nonterminal labels with τ. We have the following lemma.

Lemma 7.14 *Let $H \in \mathcal{H}_{\mathcal{T}}$, and $A \in N_0$. If $A^\bullet \Longrightarrow^*_{P_0} H$ and t is a collapsed split tree of H, then there is a derivation tree t' over P_0^* with root A such that $unlabel(t') = t$.*

Proof. Let t_0 be a derivation tree for H in ERG_0. As remarked above, by modification (2) $unlabel(t_0)$ is a total split tree for every derivation tree t_0 over P_0. Hence there is some derivation tree t_0' over P_0^* such that $unlabel(t_0')$ is a collapsed split tree. (We can construct t_0' from t_0 in the same way a collapsed split tree can be constructed from a total one; see the paragraph after Definition 7.10.) By Theorem 7.12 the given collapsed split tree t and $unlabel(t_0')$ are similar. By modification (3) this means there is a derivation tree t' over P_0^* with $unlabel(t') = t$. □

We are now able to sketch the construction of the cubic algorithm we aimed at, thereby proving Theorem 7.13.

Proof of Theorem 7.13. To find out whether an input graph H is in $L(ERG_0)$, where ERG_0 is constructed as above, proceed as follows. First, compute a collapsed split tree t of H. By a result proved by Hopcroft and Tarjan [58] this can be done in linear time. By Lemma 7.14, if $H \in L(ERG_0)$ (and, of course, only then) we can exchange every τ in t by an appropriate nonterminal label to obtain a derivation tree over P_0^* for H whose root is S. To find out whether this is possible, for all subtrees t' of t we compute by a bottom-up approach the set $poslab(t')$ of all $A \in N_0$ such that $A^\bullet \Longrightarrow^*_{P_0} result(t')$. This can be done efficiently, as follows. Suppose $t' = (R, branch)$ with $branch(e) = t_e$ for $e \in E^N_R$, and we have already computed $poslab(t_e)$ for all $e \in E^N_R$. Then, we have to decide whether $A^\bullet \Longrightarrow^*_{P_0} R_1$ for some R_1 obtained from R by labelling every edge labelled with τ in R with one of the labels in $poslab(t_e)$. Since R is either a bond, a chain, or a block there are three possible cases to consider.

1. R is a bond.
   Construct R' from R by giving a hyperedge $e \in E_R$ the label $\{lab_R(e)\}$ if $lab_R(e) \in T_0$ and the label $poslab(t_e)$ if $lab_R(e) = \tau$. Let R'' be the graph obtained from R' by reversing all edges e with $att_{R'}(e) = end_{R'}\, begin_{R'}$, while exchanging $lab_{R'}(e)$ with $\{\bar{A} \mid A \in lab_{R'}(e)\}$. By the completion performed in modification (3) and Theorem 4.8 it is not hard to see that $A \in poslab(t')$ if and only if $\psi(R'')$ is in some fixed semilinear set associated with A. (To be able to apply Theorem 4.8, add productions $(A, X^\bullet)$, where $X \subseteq N_0$ and $A \in X$, or $A \in T_0$ and $X = \{A\}$. Then we have to decide whether $A^\bullet \Longrightarrow^*_{P_0} R''$.) Due to results by Fischer, Meyer, and Rosenberg [44] membership in a semilinear set can be decided in linear time.

2. R is a chain.
   Then, since in a derivation $A^\bullet \Longrightarrow^*_{P_0} R_1$ we cannot use any production with a substantial right-hand side which is not a chain, we can use the well-known algorithm by Cocke, Kasami, and Younger to find out $poslab(t')$. (Note that we do not have to consider all the possible R_1 separately, because the CKY-algorithm works with sets of nonterminals like our $poslab(t_e)$ provide.) Since the CKY-algorithm runs in cubic time this step takes cubic time in the size of R.

3. R is a block.
   Here, we only have to test a finite number of possibilities since R must be isomorphic to a right-hand side in ERG_0, up to the labelling. Hence, this case can be handled in constant time.

Altogether, we need at most a cubic number of steps (in $|H|$) in order to find $poslab(t)$, and accept H if $S \in poslab(t)$. □

7.3 Further results and bibliographic notes

The proof of NP-completeness given in the beginning of this section was found by Lange and Welzl [66], who presented it for *string grammars with disconnecting* rather than for edge replacement, as we did here.

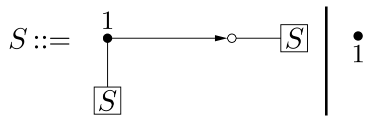

Figure 21: A hyperedge replacement grammar with componentwise derivations.

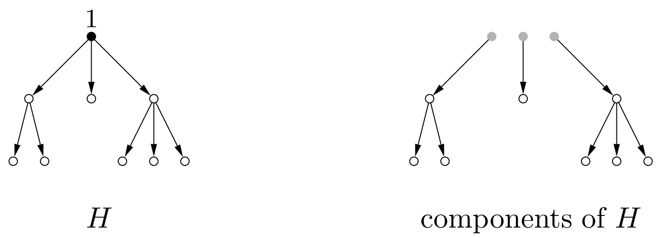

Figure 22: A graph generated by *HRG* and its components.

Algorithm 7.4 is due to Lautemann [68]. The paper contains a second variant of the algorithm that does not rely on logarithmic k-separability. Instead, it is required that the considered hyperedge replacement grammar has *componentwise derivations*. Roughly speaking, this means that an X-candidate is derivable from $A^\bullet$ if and only if each of the connected components this X-candidate consists of is derivable from $A^\bullet$. As an example, one may consider the grammar $HRG = (\{S\}, \{\llcorner\!\lrcorner\}, P, S)$ (where "␣" means "unlabelled") whose productions are the two given in Figure 21. $L(HRG)$ is the set of all (rooted, directed) trees, and a graph as in Figure 22 on the left is derivable from $S^\bullet$ if and only if each of its components (shown on the right) can be derived from $S^\bullet$. Under the condition that the grammar satisfies this requirement, we may look at the (linear number of) individual connected components rather than at all X-candidates, which yields a polynomial algorithm. As Lautemann remarks in his paper, more sophisticated notions of componentwise derivations, that still lead to polynomial algorithms, may more or less obviously be thought of.

The cubic membership algorithm was presented by Vogler in [83]. As shown by Drewes [21], Theorem 7.13 and the results used to prove it can be generalized to hyperedge replacement grammars of order k generating almost k-connected k-hypergraphs, where a k-hypergraph is a hypergraph all of whose hyperedges are of type k. This yields the case treated here by choosing $k = 2$. It may be interesting to notice that the generalized version of the algorithm is still cubic; the exponent does not depend on the order of the grammar.

As mentioned, the generalization works only for k-hypergraphs. It is therefore natural to wonder whether this restriction can be avoided. Is there still a polynomial algorithm if a grammar of order k generates, for instance, a language of almost k-connected graphs? Unfortunately, it is likely that this question has to be answered negatively, since it was shown by Drewes [20] that this case is again NP-complete for all $k > 2$. (For $k = 2$, we have Theorem 7.13.) Intuitively, this is caused by the fact that, if we replace every hyperedge in a k-connected k-hypergraph by, say, a clique on the k attached nodes, we retain k-connectedness, but loose the information which of the resulting edges belong together, so that a unique reconstruction of the hypergraph is not possible.

8 Conclusion

In this survey, we have outlined the theory of hyperedge replacement as a grammatical device for the generation of hypergraph, graph, and string languages with an emphasis on the sequential mode of rewriting. Hyperedge replacement (without application conditions) has a context-free nature which is the key for an attractive mathematical theory including structural properties and decidability results.

Although several interesting (hyper)graph languages occurring in computer science and graph theory can be generated by hyperedge replacement grammars, their generative power is bound to be restricted. As we saw in Section 4 there is, for instance, no way to generate a language of unbounded connectivity. As mentioned in the bibliographic notes of Section 3, extensions can be found in the literature, which make it possible to overcome one or the other deficiency. However, one cannot expect to get additional power for free. Normally, extensions lack some of the useful properties of hyperedge replacement, or their rewriting mechanisms are more complicated and therefore harder to reason about. In many cases hyperedge replacement seems to be a good compromise between the wish to have a nicely developed mathematical theory and the demand for reasonable generative power.

Acknowledgement

We thank Grzegorz Rozenberg and the anonymous referee for helpful comments on a draft of this chapter. The work presented here has been supported by ESPRIT Basic Research Working Group 7183 (COMPUGRAPH II).

References

1. I. J. Aalbersberg, A. Ehrenfeucht, and G. Rozenberg. On the membership problem for regular DNLC grammars. *Discrete Applied Mathematics*, 13:79–85, 1986.
2. A. V. Aho and J. D. Ullman. Translations on a context free grammar. *Information and Control*, 19:439–475, 1971.
3. S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
4. Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase-structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–177, 1961.
5. M. Bauderon. Infinite hypergraphs I. Basic properties. *Theoretical Computer Science*, 82:177–214, 1991.
6. M. Bauderon. Infinite hypergraphs II. Systems of recursive equations. *Theoretical Computer Science*, 103:165–190, 1992.
7. M. Bauderon and B. Courcelle. Graph expressions and graph rewriting. *Mathematical Systems Theory*, 20:83–127, 1987.
8. A. Corradini, H. Ehrig, R. Heckel, M. Löwe, U. Montanari, and F. Rossi. Algebraic approach to graph transformations. part I: Basic concepts and double pushout approach, 1996. This volume.
9. B. Courcelle. An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theoretical Computer Science*, 55:141–181, 1987.
10. B. Courcelle. Graph rewriting: An algebraic and logical approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume Vol. B., pages 193–242. Elsevier, Amsterdam, 1990.
11. B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
12. B. Courcelle. Context-free graph grammars: Separating vertex replacement from hyperedge replacement. In Z. Ésik, editor, *Fundamentals of Computation Theory*, volume 710 of *Lecture Notes in Computer Science*, pages 181–193, 1993.
13. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic, 1996. This volume.
14. B. Courcelle and J. Engelfriet. A logical characterization of the sets of hypergraphs defined by hyperedge replacement systems. *Mathematical Systems Theory*, 28:515–552, 1995.
15. B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46:218–270,

1993.
16. B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109:49–82, 1993.
17. J. Dassow, A. Habel, and S. Taubenberger. Chain-code pictures and collages generated by hyperedge replacement. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Fifth Intl. Workshop on Graph Grammars and Their Application to Comp. Sci.*, Lecture Notes in Computer Science. Springer, 1996. To appear.
18. G. David, F. Drewes, and H.-J. Kreowski. Hyperedge replacement with rendezvous. In P. Jouannaud, editor, *Proc. Theory and Practice of Software Development*, volume 668 of *Lecture Notes in Computer Science*, pages 167–181, 1993.
19. P. Della Vigna and C. Ghezzi. Context-free graph grammars. *Information and Control*, 37:207–233, 1978.
20. F. Drewes. NP-completeness of k-connected hyperedge-replacement languages of order k. *Information Processing Letters*, 45:89–94, 1993.
21. F. Drewes. Recognising k-connected hypergraphs in cubic time. *Theoretical Computer Science*, 109:83–122, 1993.
22. F. Drewes. Computation by tree transductions. Doctoral Dissertation, University of Bremen, Germany, 1996.
23. F. Drewes. Language theoretic and algorithmic properties of d-dimensional collages and patterns in a grid. *Journal of Computer and System Sciences*, 1996. To appear.
24. F. Drewes. A lower bound on the growth of functions computed by tree transductions. *Fundamenta Informaticae*, 1996. To appear; short version in Lecture Notes in Computer Science 787 (CAAP 94).
25. F. Drewes. The use of tree transducers to compute translations between graph algebras. In *Proc. Fifth Intl. Workshop on Graph Grammars and Their Application to Comp. Sci.*, Lecture Notes in Computer Science. Springer, 1996. To appear.
26. F. Drewes, A. Habel, H.-J. Kreowski, and S. Taubenberger. Generating self-affine fractals by collage grammars. *Theoretical Computer Science*, 145:159–187, 1995.
27. F. Drewes and H.-J. Kreowski. (Un)decidability of properties of pictures generated by collage grammars. *Fundamenta Informaticae*, 1996. To appear.
28. H. Ehrig and K. W. Tischer. Graph grammars and applications to specialization and evolution in biology. *Journal of Computer and System Sciences*, 11:212–236, 1975.
29. J. Engelfriet. Context-free NCE graph grammars. In J. Csirik,

J. Demetrovics, and F. Gécseg, editors, *Fundamentals of Computation Theory*, volume 380 of *Lecture Notes in Computer Science*, pages 148–161, 1989.

30. J. Engelfriet. A characterization of context-free NCE graph languages by monadic second-order logic on trees. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 311–327, 1991.
31. J. Engelfriet. A Greibach normal form for context-free graph grammars. In W. Kuich, editor, *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 138–149, 1992.
32. J. Engelfriet. Graph grammars and tree transducers. In *CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pages 15–36, 1994.
33. J. Engelfriet and L. Heyker. The string generating power of context-free hypergraph grammars. *Journal of Computer and System Sciences*, 43:328–360, 1991.
34. J. Engelfriet and L. Heyker. Context-free hypergraph grammars have the same term-generating power as attribute grammars. *Acta Informatica*, 29:161–210, 1992.
35. J. Engelfriet and L. Heyker. Hypergraph languages of bounded degree. *Journal of Computer and System Sciences*, 48:58–89, 1994.
36. J. Engelfriet, L. Heyker, and G. Leih. Context-free graph languages of bounded degree are generated by apex graph grammars. *Acta Informatica*, 31:341–378, 1994.
37. J. Engelfriet and G. Rozenberg. A comparison of boundary graph grammars and context-free hypergraph grammars. *Information and Computation*, 84:163–206, 1990.
38. J. Engelfriet and G. Rozenberg. Node replacement graph grammars, 1996. This volume.
39. J. Engelfriet, G. Rozenberg, and G. Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20:150–202, 1980.
40. J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31:71–146, 1985.
41. J. Engelfriet and H. Vogler. The translation power of top-down tree-to-graph transducers. *Journal of Computer and System Sciences*, 49:258–305, 1994.
42. R. Farrow, K. Kennedy, and L. Zucconi. Graph grammars and global program data flow analysis. In *Proc. 17th Ann. IEEE Symp. on Found. of Comp. Sci.*, pages 42–56, Houston, 1976.

43. J. Feder. Plex languages. *Information Sciences*, 3:225–241, 1971.
44. P. C. Fischer, A. R. Meyer, and A. L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2:265–283, 1968.
45. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
46. F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.
47. S. Ginsburg and G. Rice. Two families of languages related to ALGOL. *Journal of the ACM*, 9:350–371, 1962.
48. A. Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992.
49. A. Habel. Hypergraph grammars: Transformational and algorithmic aspects. *Journal of Information Processing and Cybernetics EIK*, 28:241–277, 1992.
50. A. Habel and H.-J. Kreowski. Characteristics of graph languages generated by edge replacement. *Theoretical Computer Science*, 51:81–115, 1987.
51. A. Habel and H.-J. Kreowski. May we introduce to you: Hyperedge replacement. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 15–26, 1987.
52. A. Habel and H.-J. Kreowski. Some structural aspects of hypergraph languages generated by hyperedge replacement. In F. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *STACS 87*, volume 247 of *Lecture Notes in Computer Science*, pages 207–219, 1987.
53. A. Habel and H.-J. Kreowski. Filtering hyperedge-replacement languages through compatible properties. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science*, pages 107–120, 1990.
54. A. Habel and H.-J. Kreowski. Collage grammars. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 411–429, 1991.
55. A. Habel, H.-J. Kreowski, and C. Lautemann. A comparison of compatible, finite, and inductive graph properties. *Theoretical Computer Science*, 110:145–168, 1993.
56. A. Habel, H.-J. Kreowski, and W. Vogler. Metatheorems for decision problems on hyperedge replacement graph languages. *Acta Informatica*, 26:657–677, 1989.

57. A. Habel, H.-J. Kreowski, and W. Vogler. Decidable boundedness problems for sets of graphs generated by hyperedge replacement. *Theoretical Computer Science*, 89:33–62, 1991.
58. J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
59. J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, Mass., 1969.
60. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, Reading, Massachusetts, 1979.
61. D. Janssens and G. Rozenberg. Restrictions, extensions and variations of NLC grammars. *Information Sciences*, 20:217–244, 1980.
62. H.-J. Kreowski. A pumping lemma for context-free graph languages. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 270–283, 1979.
63. H.-J. Kreowski. Rule trees represent derivations in edge replacement systems. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 217–232. Springer-Verlag, Berlin, 1986.
64. H.-J. Kreowski. Parallel hyperedge replacement. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer Systems*, pages 271–282. Springer-Verlag, Berlin Heidelberg New York, 1992.
65. H.-J. Kreowski. Five facets of hyperedge replacement beyond context-freeness. In Z. Ésik, editor, *Fundamentals of Computation Theory*, volume 710 of *Lecture Notes in Computer Science*, pages 69–86, 1993.
66. K.-J. Lange and E. Welzl. String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. *Discrete Applied Mathematics*, 16:17–30, 1987.
67. C. Lautemann. Efficient algorithms on context-free graph languages. In T. Lepistö and A. Salomaa, editors, *Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 362–378, 1988.
68. C. Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27:399–421, 1990.
69. C. Lautemann. Tree automata, tree decomposition, and hyperedge replacement. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 520–537, 1991.
70. T. Lengauer and E. Wanke. Efficient analysis of graph properties on context-free graph languages. In T. Lepistö and A. Salomaa, editors,

Automata, Languages and Programming, volume 317 of *Lecture Notes in Computer Science*, pages 379–393, 1988.
71. T. Lengauer and E. Wanke. Efficient solution of connectivity problems on hierarchically defined graphs. *SIAM Journal on Computing*, 17:1063–1080, 1988.
72. T. Lengauer and E. Wanke. Efficient decision procedures for graph properties on context-free graph languages. *Journal of the ACM*, 40:368–393, 1993.
73. S. MacLane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3:460–472, 1937.
74. M. Minas and G. Viehstaedt. Specification of diagram editors providing layout adjustment with minimal change. In *Proc. IEEE Symp. on Visual Languages (VL'93)*, pages 324–329. IEEE Comp. Society Press, 1993.
75. M. Minas and G. Viehstaedt. *DiaGen*: A generator for diagram editors providing direct manipulation and execution of diagrams adjustment with minimal change. In *Proc. IEEE Symp. on Visual Languages (VL'95).* IEEE Comp. Society Press, 1995.
76. U. Montanari and F. Rossi. An efficient algorithm for the solution of hierarchical networks of constraints. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 440–457, 1987.
77. R. Parikh. On context-free languages. *Journal of the ACM*, 13:570–581, 1966.
78. T. Pavlidis. Linear and context-free graph grammars. *Journal of the ACM*, 19(1):11–23, 1972.
79. D. J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7:317–322, 1974.
80. H. Seidl. Finite tree automata with cost functions. *Theoretical Computer Science*, 126:113–142, 1994.
81. A. Slisenko. Context-free graph grammars as a tool for describing polynomial-time subclasses of hard problems. *Information Processing Letters*, 14:52–56, 1982.
82. G. Viehstaedt. A generator for diagram editors. Doctoral Dissertation, University of Erlangen, Germany, 1995.
83. W. Vogler. Recognizing edge replacement graph languages in cubic time. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 676–687, 1991.
84. E. Wanke. On the decidability of certain integer subgraph problems on

context-free graph languages. *Information and Computation*, 113:26–49, 1994.
85. E. Wanke and M. Wiegers. Undecidability of the bandwidth problem on linear graph languages. *Information Processing Letters*, 33:193–197, 1989.