Context-Free Graph Grammars and Concatenation of Graphs

Joost Engelfriet * and Jan Joris Vereijken **

Department of Computer Science, Leiden University
P.O.Box 9512, NL-2300 RA Leiden, The Netherlands
e-mail: engelfri@wi.leidenuniv.nl

Abstract. An operation of concatenation is defined for graphs. This allows strings to be viewed as expressions denoting graphs, and string languages to be interpreted as graph languages. For a class K of string languages, Int(K) is the class of all graph languages that are interpretations of languages from K. For the classes REG and LIN of regular and linear context-free languages, respectively, Int(REG) = Int(LIN). Int(REG) is the smallest class of graph languages containing all singletons and closed under union, concatenation and star (of graph languages). Int(REG) equals the class of graph languages generated by linear HR (= Hyperedge Replacement) grammars, and Int(K) is generated by the corresponding K-controlled grammars. Two characterizations are given of the largest class K' such that Int(K') = Int(K). For the class CF of context-free languages, Int(CF) lies properly inbetween Int(REG) and the class of graph languages generated by HR grammars. The concatenation operation on graphs combines nicely with the sum operation on graphs. The class of context-free (or equational) graph languages, with respect to these two operations, is the class of graph languages generated by HR grammars.

1 Introduction

Context-free graph languages are generated by context-free graph grammars, which are usually graph replacement systems. One of the most popular types of context-free graph grammar is the Hyperedge Replacement System, or HR grammar (see, e.g., [Hab, HabKre, HabKV]). A completely different way of generating graphs is to select a number of graph operations, to generate a set of expressions (built from these operations), and to interpret the expressions as graphs. The set of expressions is generated by a classical context-free grammar generating strings (or more precisely, by a regular tree grammar). This way of generating graphs was introduced, for arbitrary objects rather than graphs, in [MezWri], where the generated sets of objects are called equational. For graphs in particular, this

* The first author was supported by ESPRIT BRWG No.7183 COMPUGRAPH II.

** The present address of the second author is Faculty of Mathematics and Computing Science, Eindhoven University of Technology, P.O.Box 513, NL-5600 MB Eindhoven, The Netherlands, e-mail: janjoris@acm.org

generation method was first investigated in [BauCou]. It is shown in [BauCou] that, for a particular collection of graph operations, this new graph generating method is equivalent with the HR grammar. Other work on the generation of graphs through graph expressions is in, e.g., [Cou2, CouER, Dre, Eng].

In this framework we investigate another, natural operation on graphs that was introduced (for "planar nets") in [Hot1] (and which is a simple variation of the graph operations in [BauCou]). Due to its similarity to the concatenation of strings, we call it *concatenation of graphs*. Together with the sum operation of graphs (introduced for planar nets in [Hot1] and defined for graphs in [BauCou]) and all constant graphs, a collection of graph operations is obtained that is simpler than the one in [BauCou], but also has the power of the HR grammar (which is our first main result, proved in Section 4). Concatenation and sum satisfy some nice basic properties, discussed in Section 3; in particular, all graphs can be built from a small number of elementary graphs with the operations of concatenation and sum. Thus, it suffices to use these elementary graphs in the context-free grammars that generate graph expressions.

The basic laws that are satisfied by concatenation and sum of planar nets, form the basis of the theory of x-categories developed in [Hot1] (also called strict monoidal categories, see, e.g., [EhrKKK, Ben]). Free x-categories model the sets of derivation graphs of Chomsky type 0 grammars (see [Hot2, Ben]). Finite automata on such graphs are considered, e.g., in [BosDW]. The idea of using concatenation and sum in graph grammars is from [HotKM], where "logic topological nets" are generated by graph grammars (with parallel rewriting). Our first main result (mentioned above) confirms the naturalness of these operations.

Our main interest in this paper is in the generation of graphs through graph expressions that use concatenation only. Since graph concatenation is associative, an expression that is built from constant graphs by concatenation, is essentially the same as a string. This shows that we can use arbitrary context-free grammars as graph grammars, by just interpreting the generated strings as graphs. More generally, every class K of string languages determines a class $\mathrm{Int}(K)$ of graph languages: $\mathrm{Int}(K)$ is the set of all graph languages $h(L)$ where h is an "interpretation" and L is a string language from K. An *interpretation* of an alphabet A is a mapping h that associates a graph $h(a)$ with every symbol a; it is extended to strings over A by $h(a_1 \cdots a_n) = h(a_1) \circ \cdots \circ h(a_n)$, where $\circ$ denotes concatenation of graphs. Thus, symbols are interpreted as graphs, strings are interpreted as graphs (by interpreting string concatenation as graph concatenation), and string languages are interpreted as graph languages. Note that an interpretation looks like a semi-group homomorphism; however, it is not exactly one, because concatenation on graphs is, in fact, a partial operation. More precisely, graphs are typed, and concatenation is defined only if the types "fit". In fact, as in [Hab], our graphs are equipped with a designated sequence of "begin nodes" and a designated sequence of "end nodes" (generalizing the idea that strings have a beginning and an end). A graph g_1 can be concatenated with a graph g_2 only if the sequence of end nodes of g_1 has the same length as the sequence of begin nodes of g_2. Their concatenation $g_1 \circ g_2$ is obtained by identifying

each end node of g_1 with the corresponding begin node of g_2 (just as strings are concatenated by identifying the end of the first string with the beginning of the second).

We investigate Int(K) for specific K (such as the class REG of regular languages, the class CF of context-free languages, and the class LIN of linear context-free languages), but also for arbitrary K (satisfying some mild closure properties). In Section 5, after defining the notion of interpretation, we show that the graph languages in Int(REG) are exactly those that can be denoted by regular expressions, built from singleton graph languages with the operations of union, concatenation, and star (on graph languages). We also show that Int(REG) = Int(LIN) and that it equals the class LIN-HR of graph languages generated by *linear* HR grammars. This suggests that regularity and linearity are the same for graph languages. The class Int(CF) contains, as expected, exactly those graph languages that can be generated by expression generating context-free grammars that do not use the sum operation. Thus, by our first main result, it is included in the class of graph languages generated by HR grammars. The inclusion is proper, due to the close connection between graph concatenation and the pathwidth of graphs: every graph language in Int(K) is of bounded pathwidth (and graph languages of unbounded pathwidth, such as the set of trees, can be generated by HR grammars).

Generalizing the result that Int(REG) = LIN-HR, we show in Section 6 that (under the rather weak assumption that K is closed under sequential machine mappings) Int(K) is equal to LIN-HR(K), the class of graph languages that are generated by linear HR grammars with a control language from K (with the usual notion of control).

As observed above, Int(REG) = Int(LIN). In Section 7 we investigate the question, for given K and K', whether or not Int(K') = Int(K) (where we assume that K and K' are closed under sequential machine mappings). Trivially, for every K there is a largest class $\overline{K}$ such that Int($\overline{K}$) = Int(K). We call this class the *extension* of K, denoted Ext(K). Clearly, the question Int(K') = Int(K) is now reduced to the question Ext(K') = Ext(K), which concerns classes of string languages rather than graph languages. The main result of this section is that Ext(K) consists exactly of all string languages that are in Int(K), coding strings as graphs in the obvious way (viz., as edge-labeled chain graphs). Using the characterization in Section 6, and generalizing a result concerning the string generating power of linear HR grammars from [EngHey], we show that Ext(K) = 2DGSM(K), the class of all languages that are images of languages from K under 2-way deterministic gsm mappings. Thus, Int(K') = Int(K) iff 2DGSM(K') = 2DGSM(K), a purely formal language-theoretic question. By the well-known result that 2DGSM(REG) is properly included in 2DGSM(CF), we conclude that Int(REG) is properly included in Int(CF).

A preliminary version of this paper was presented at the 5th International Workshop on Graph Grammars and their Application to Computer Science [EngVer]. The work is based on the Master's Thesis of the second author [Ver].

2 Preliminaries

2.1 Strings

We assume the reader to be familiar with formal language theory (see, e.g., [Ber, HopUll, Sal]). Here we just recall some of the concepts to be used.

$\mathbf{N} = \{0, 1, 2, \ldots\}$ denotes the set of natural numbers. For a set V, V^* denotes the set of all finite sequences (or strings) of elements of V. A sequence $\langle v_1, v_2, \ldots, v_n\rangle \in V^*$, with $v_i \in V$, is also written as $v_1 v_2 \cdots v_n$ (as λ if $n = 0$). The length of a string $w \in V^*$ is denoted $|w|$, and, for $1 \leq i \leq |w|$, its ith element is denoted $w(i)$. Thus, if $w = v_1 \cdots v_n$, then $|w| = n$ and $w(i) = v_i$. Concatenation of strings is defined in the usual way.

A context-free grammar is a tuple $G = (N, T, P, S)$ where N is the nonterminal alphabet, T is the terminal alphabet (disjoint with N), P is the set of productions (of the form $X \to \alpha$, with $\alpha \in (N \cup T)^*$), and S is the initial nonterminal. The language $L(G) \subseteq T^*$ generated by G is defined in the usual way. A context-free grammar is linear if there is at most one nonterminal occurrence in each right-hand side of a production, and it is right-linear if each production is of the form $X \to aY$ or $X \to a$, with $X, Y \in N$ and $a \in T$. The class of languages generated by all (all linear) context-free grammars is denoted CF (LIN, respectively). By REG we denote the class of regular languages. Note that the right-linear context-free grammars generate the class of λ-free regular languages (i.e., those regular languages that do not contain λ).

2.2 Graphs and graph replacement

We consider the multi-pointed, directed, edge-labeled hypergraphs of [Hab]. Such a hypergraph consists of a set of nodes and a set of (hyper)edges, just as an ordinary graph, except that an edge may have any number of sources and any number of targets, rather than just one source and one target. Each edge is labeled with a symbol from a "doubly-ranked" alphabet, in such a way that the first (second) rank of its label equals the number of its sources (targets, respectively). Finally, every hypergraph is multi-pointed in the sense that it has a designated sequence of "begin nodes", and a designated sequence of "end nodes"; these can be used conveniently for gluing hypergraphs to each other.

Formally, a *typed* (or doubly ranked) *alphabet* is an alphabet Σ together with a mapping $\mathrm{type} : \Sigma \to \mathbf{N} \times \mathbf{N}$. A *multi-pointed hypergraph* over Σ is a tuple $g = (V, E, s, t, l, \mathrm{begin}, \mathrm{end})$, where V is the finite set of nodes, E is the finite set of (hyper)edges, $s : E \to V^*$ is the source function, $t : E \to V^*$ is the target function, $l : E \to \Sigma$ is the labeling function such that $\mathrm{type}(l(e)) = (|s(e)|, |t(e)|)$ for every $e \in E$, $\mathrm{begin} \in V^*$ is the sequence of begin nodes, and $\mathrm{end} \in V^*$ is the sequence of end nodes.

For a given multi-pointed hypergraph g, its components will also be denoted by V_g, E_g, s_g, t_g, l_g, $\mathrm{begin}(g)$, and $\mathrm{end}(g)$. If $|\mathrm{begin}(g)| = m$ and $|\mathrm{end}(g)| = n$, then g is said to be of *type* (m, n) and we write $\mathrm{type}(g) = (m, n)$. Similarly, for an edge e of g, we write $\mathrm{type}(e)$ to denote $\mathrm{type}(l(e))$; thus, by the above

requirement, if type$(e) = (m, n)$, then e has m sources and n targets. If a multi-pointed hypergraph is of type $(0, 0)$ and all its edges are of type $(1, 1)$, then it is an ordinary directed graph (with labeled edges).

For a typed symbol σ, with type$(\sigma) = (m, n)$, we denote by atom(σ) the multi-pointed hypergraph g of type (m, n) such that $V_g = \{x_1, \ldots, x_m, y_1, \ldots, y_n\}$, $E_g = \{e\}$ with $l(e) = \sigma$, and begin$(g) = s(e) = \langle x_1, \ldots, x_m \rangle$, and end$(g) = t(e) = \langle y_1, \ldots, y_n \rangle$. A multi-pointed hypergraph of the form atom(σ) will be called an *atom* (it is called a handle in [Hab]).

Two multi-pointed hypergraphs g and h are *disjoint* if $V_g \cap V_h = \emptyset$ and $E_g \cap E_h = \emptyset$.

From now on we will just say graph *instead of multi-pointed hypergraph.* As usual we consider both concrete and abstract graphs, where an abstract graph is an equivalence class of isomorphic concrete graphs. The isomorphisms between graphs g and h are the usual ones, which, additionally, should map begin(g) to begin(h), and end(g) to end(h). In particular, isomorphic graphs have the same type. We are only interested in abstract graphs; concrete graphs are just used as representatives of abstract graphs. The set of abstract graphs over a typed alphabet Σ will be denoted GR(Σ), and GR denotes the union of all GR(Σ) (where Σ is taken from some fixed, infinite set of symbols). A (typed) *graph language* is a subset L of GR(Σ), for some Σ, such that all graphs in L have the same type (m, n), also called the type of L, and denoted by type$(L) = (m, n)$.

A basic operation on graphs is the substitution of a graph for an edge (see [Hab, BauCou]). To define it formally, it is convenient to use an operation of *node identification* (or "gluing"), as follows.

Let g be a graph, and let $R \subseteq V_g \times V_g$. Intuitively, we wish to identify nodes x and y, for every pair $(x, y) \in R$. For $x \in V_g$, let $[x]_R$ denote the equivalence class of x with respect to the smallest equivalence relation on V_g containing R. For $V \subseteq V_g$, let $V/R = \{[x]_R \mid x \in V\}$. For a sequence $x = \langle x_1, \ldots, x_n \rangle \in V_g^*$ with $x_i \in V_g$, let $[x]_R = \langle [x_1]_R, \ldots, [x_n]_R \rangle$. Then we define the graph g/R by $g/R = (V_g/R, E_g, s, t, l_g, [\text{begin}]_R, [\text{end}]_R)$ such that $s(e) = [s_g(e)]_R$ and $t(e) = [t_g(e)]_R$ for every $e \in E_g$.

Substitution of a graph for an edge is now defined as follows. Let g be a graph, let e be an edge of g, and let h be a graph such that type$(h) =$ type$(e) = (m, n)$. We assume that g and h are disjoint (otherwise an isomorphic copy of h should be taken). Let g' be the graph that is obtained from g by removing e and adding h (disjointly), i.e., $g' = (V_g \cup V_h, (E_g - \{e\}) \cup E_h, s, t, l, \text{begin}(g), \text{end}(g))$, where $s(e) = s_g(e)$ for $e \in E_g - \{e\}$ and $s(e) = s_h(e)$ for $e \in E_h$, and similarly for t and l. Note that g' has the begin and end nodes of g. Then the *substitution* of h for e in g, denoted by $g[e/h]$, is the graph g'/R where $R = \{(s_g(e)(i), \text{begin}(h)(i)) \mid 1 \leq i \leq m\} \cup \{(t_g(e)(i), \text{end}(h)(i)) \mid 1 \leq i \leq n\}$. Thus, intuitively, after removing e and adding h, the ith source of e is identified with the ith begin node of h, and the ith target of e is identified with the ith end node of h. The notion of substitution defined here is not precisely the one in [Hab], but it is (the appropriate extension to the doubly ranked case of) the one in [BauCou]; however, they are equivalent from the point of view of graph generation by hyperedge replacement grammars.

In a substitution $g[e/h]$, h can be taken as an abstract graph (in the sense that if h and h' are isomorphic, then so are $g[e/h]$ and $g[e/h']$); but g is necessarily concrete, because its concrete edge e is involved. To turn substitution into an operation on abstract graphs, we substitute graphs for all edges of g and let the graph h to be substituted for edge e be determined by the label of e. This leads us to a notion of substitution that generalizes the notion of homomorphism of strings (in formal language theory), and that we will call "replacement" (of edges by graphs).

Let Σ be a typed alphabet. A *replacement* is a mapping $\phi : \Sigma \to \mathrm{GR}$ such that $\mathrm{type}(\phi(\sigma)) = \mathrm{type}(\sigma)$ for every $\sigma \in \Sigma$; it is extended to a mapping from $\mathrm{GR}(\Sigma)$ to GR by defining, for $g \in \mathrm{GR}(\Sigma)$, $\phi(g) = g[e_1/\phi(l(e_1))] \cdots [e_k/\phi(l(e_k))]$, where $E_g = \{e_1, \ldots, e_k\}$. Thus, every edge e of g with label $l(e) = \sigma$ is replaced by the graph $\phi(\sigma)$. It is well known that this definition does not depend on the order $e_1, \ldots, e_k$ in which the edges are replaced (because substitution is *confluent*, cf. [Cou1]). It should also be clear that every replacement is an operation on abstract graphs: if g and g' are isomorphic, then so are $\phi(g)$ and $\phi(g')$.

We denote the class of all replacements by Repl, and, for a class K of graph languages, we let $\mathrm{Repl}(K) = \{\phi(L) \mid \phi \in \mathrm{Repl}, L \in K\}$.

Another basic property of substitution is its *associativity* (see [Cou1]). In our present formulation it means that the composition of two replacements is again a replacement (as one would expect from a generalization of string homomorphism).

Proposition 1. Repl *is closed under composition.*

Proof. It can be shown, based on the associativity of substitution, that, for a replacement ϕ, $\phi(g[e_1/h_1] \cdots [e_k/h_k]) = g[e_1/\phi(h_1)] \cdots [e_k/\phi(h_k)]$, where $E_g = \{e_1, \ldots, e_k\}$. Now let Σ_1 and Σ_2 be two typed alphabets. Let $\phi_1 : \Sigma_1 \to \mathrm{GR}(\Sigma_2)$ and $\phi_2 : \Sigma_2 \to \mathrm{GR}$ be two replacements. Define the replacement $\phi : \Sigma_1 \to \mathrm{GR}$ by: $\phi(\sigma) = \phi_2(\phi_1(\sigma))$ for every $\sigma \in \Sigma_1$. Then, for a graph g with $E_g = \{e_1, \ldots, e_k\}$, $\phi_2(\phi_1(g)) = \phi_2(g[e_1/\phi_1(l(e_1))] \cdots [e_k/\phi_1(l(e_k))]) = g[e_1/\phi_2(\phi_1(l(e_1)))] \cdots [e_k/\phi_2(\phi_1(l(e_k)))] = g[e_1/\phi(l(e_1))] \cdots [e_k/\phi(l(e_k))] = \phi(g)$. This shows that $\phi = \phi_2 \circ \phi_1$. □

A useful elementary property of replacements is that, for every replacement $\phi : \Sigma \to \mathrm{GR}$ and every $\sigma \in \Sigma$, $\phi(\mathrm{atom}(\sigma)) = \phi(\sigma)$. Also, if $\phi(\sigma) = \mathrm{atom}(\sigma)$ for every $\sigma \in \Sigma$, then $\phi(g) = g$ for every $g \in \mathrm{GR}(\Sigma)$.

Besides replacement operations, there are two other, simpler operations on (abstract) graphs that will be useful. They only change the begin and end nodes of a graph. Let g be a graph. The *fold* of g, denoted $\mathrm{fold}(g)$, is the same as g, except that $\mathrm{begin}(\mathrm{fold}(g)) = \lambda$ and $\mathrm{end}(\mathrm{fold}(g)) = \mathrm{begin}(g) \cdot \mathrm{end}(g)$, where $\cdot$ denotes concatenation of strings, as usual. The *backfold* of g, denoted $\mathrm{backfold}(g)$, is the same as g, except that $\mathrm{begin}(\mathrm{backfold}(g)) = \mathrm{begin}(g) \cdot \mathrm{end}(g)$ and $\mathrm{end}(\mathrm{backfold}(g)) = \lambda$.

2.3 Hyperedge replacement grammars

Hyperedge replacement grammars (or HR grammars) are context-free graph grammars that substitute graphs for edges. An *HR grammar* is a tuple $G = (N, T, P, S)$ where N is a typed alphabet of nonterminals, T is a typed alphabet of terminals (disjoint with N), P is a finite set of productions, and $S \in N$ is the initial nonterminal. Every production in P is of the form $X \to h$ with $X \in N$, $h \in \mathrm{GR}(N \cup T)$, and $\mathrm{type}(X) = \mathrm{type}(h)$; moreover, we assume (without loss of generality) that *no two edges of h are labeled by the same nonterminal*.

Application of a production $p = X \to h$ to a graph is defined as follows. Let $g \in \mathrm{GR}(N \cup T)$, and let $e \in E_g$. Then p is applicable to e if $l_g(e) = X$, and the result of the application is the graph $g[e/h]$. We write $g \Rightarrow_p g'$, or just $g \Rightarrow g'$, if g' is the result of applying p to e of g, i.e., if g' is (isomorphic to) $g[e/h]$. As usual, $\Rightarrow^*$ denotes the transitive reflexive closure of $\Rightarrow$. The *graph language generated by G* is $L(G) = \{g \in \mathrm{GR}(T) \mid \mathrm{atom}(S) \Rightarrow^* g\}$. Note that $\mathrm{type}(L(G)) = \mathrm{type}(S)$.

We denote by HR the class of graph languages generated by HR grammars. An HR grammar is *linear* if there is at most one nonterminal edge in each right-hand side of a production. We denote by LIN-HR the class of graph languages generated by linear HR grammars.

A fundamental property of HR grammars is formulated in the following "context-freeness lemma" (cf. Section II.2 of [Hab]). As shown in Lemma 2.14 of [Cou1], it is based on the associativity of substitution. Due to the above assumption that a nonterminal occurs at most once in the right-hand side of a production, it can be stated in terms of replacements, as follows.

Proposition 2. *Let $G = (N, T, P, S)$ be an HR grammar. Let $X \to h$ be in P, and let $g \in \mathrm{GR}(T)$. Let $\mathrm{lab}(h) = \{l_h(e) \mid e \in E_h\}$. Then $h \Rightarrow^* g$ if and only if there exists a replacement $\phi : \mathrm{lab}(h) \to \mathrm{GR}(T)$ such that $\phi(h) = g$ and $\mathrm{atom}(\sigma) \Rightarrow^* \phi(\sigma)$ for every $\sigma \in \mathrm{lab}(h)$. Moreover, the length of the derivation $h \Rightarrow^* g$ equals the sum of the lengths of all derivations $\mathrm{atom}(\sigma) \Rightarrow^* \phi(\sigma)$.*

3 Concatenation and Sum

In this section we define the graph operation of concatenation, and investigate some of its basic properties. In particular we show that it combines well with the sum operation on graphs. These operations work on abstract graphs. Intuitively, concatenation is sequential composition of graphs, and sum is parallel composition of graphs.

If g and h are graphs with $\mathrm{type}(g) = (k, m)$ and $\mathrm{type}(h) = (m, n)$, then their *concatenation* $g \circ h$ is the graph obtained by first taking the disjoint union of g and h, and then identifying the ith end node of g with the ith begin node of h, for every $i \in \{1, \ldots, m\}$; moreover, $\mathrm{begin}(g \circ h) = \mathrm{begin}(g)$ and $\mathrm{end}(g \circ h) = \mathrm{end}(h)$, and so $\mathrm{type}(g \circ h) = (k, n)$. Note that the concatenation of g and h is defined only when $|\mathrm{end}(g)| = |\mathrm{begin}(h)|$. Formally, the definition is as follows (where we use node identification as defined in Section 2.2).

Definition 3. Let g and h be graphs such that $|\mathrm{end}(g)| = |\mathrm{begin}(h)|$. We assume that g and h are disjoint (otherwise an isomorphic copy of g or h should be taken). The *concatenation* $g \circ h$ of g and h is the graph $(g\&h)/R$ where $g\&h = (V_g \cup V_h, E_g \cup E_h, s_g \cup s_h, t_g \cup t_h, l_g \cup l_h, \mathrm{begin}(g), \mathrm{end}(h))$ and $R = \{(\mathrm{end}(g)(i), \mathrm{begin}(h)(i)) \mid 1 \leq i \leq |\mathrm{end}(g)|\}$. □

The *sum* $g \oplus h$ of arbitrary graphs g and h is their disjoint union, with their sequences of begin nodes concatenated, and similarly for their end nodes. More formally, assuming that g and h are disjoint, $g \oplus h = (V_g \cup V_h, E_g \cup E_h, s_g \cup s_h, t_g \cup t_h, l_g \cup l_h, \mathrm{begin}(g) \cdot \mathrm{begin}(h), \mathrm{end}(g) \cdot \mathrm{end}(h))$.

The sum operation is taken from [BauCou] (where only graphs without end nodes are considered). All other operations in [BauCou] (viz. source redefinitions and source fusions) are unary operations, each of which is left-concatenation with a specific fixed graph.

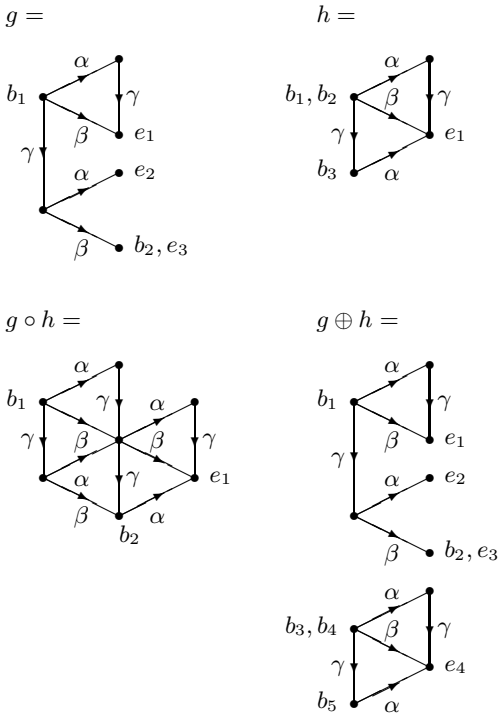

Fig. 1. Two graphs, their concatenation, and their sum.

Figure 1 shows two (ordinary) abstract graphs, g of type $(2, 3)$ and h of type $(3, 1)$, with their concatenation $g \circ h$ of type $(2, 1)$ and their sum $g \oplus h$ of type

$(5, 4)$. The graphs are drawn in the usual way; the ith begin node is indicated by b_i, and the ith end node by e_i.

These two graph operations have a number of simple properties. We stress again that the following lemmas are all about abstract graphs; in particular, the equality sign refers to the equality of abstract graphs (which is isomorphism of concrete graphs). First of all we show the basic fact that replacements are homomorphisms with respect to concatenation (just as string homomorphisms, of which they are a generalization) and with respect to sum.

Lemma 4. *Let $\phi : \Sigma \to \mathrm{GR}$ be a replacement, and let $g, h \in \mathrm{GR}(\Sigma)$.*
(1) if $|\mathrm{end}(g)| = |\mathrm{begin}(h)|$, then $\phi(g \circ h) = \phi(g) \circ \phi(h)$, and
(2) $\phi(g \oplus h) = \phi(g) \oplus \phi(h)$.

Proof. (1) It is easy to verify this equality in the case that both g and h are atoms (for the definition of an atom, see Section 2.2). The general case is then proved as follows. Let σ and τ be two symbols with the same type as g and h, respectively. Let $\psi : \{\sigma, \tau\} \to \mathrm{GR}(\Sigma)$ be the replacement with $\psi(\sigma) = g$ and $\psi(\tau) = h$. Then, by the above special case, $\psi(\mathrm{atom}(\sigma) \circ \mathrm{atom}(\tau)) = \psi(\mathrm{atom}(\sigma)) \circ \psi(\mathrm{atom}(\tau)) = \psi(\sigma) \circ \psi(\tau) = g \circ h$. Hence $\phi(g \circ h) = \phi(\psi(\mathrm{atom}(\sigma) \circ \mathrm{atom}(\tau)))$. By Proposition 1, $\phi \circ \psi$ is a replacement. Hence, again by the above special case, $\phi(g \circ h) = \phi(\psi(\mathrm{atom}(\sigma)) \circ \phi(\psi(\mathrm{atom}(\tau)) = \phi(\psi(\sigma)) \circ \phi(\psi(\tau)) = \phi(g) \circ \phi(h)$.

The proof of (2) is analogous. □

This lemma allows us to prove laws about $\circ$ and $\oplus$ by proving them for atoms only (as, in fact, we already did in the proof of Lemma 4). The next lemma summarizes the main basic properties of $\circ$ and $\oplus$.

Definition 5. For every $n \in \mathbf{N}$ the *identity* id_n of type (n, n) is the discrete graph with nodes $x_1, \ldots, x_n$ and $\mathrm{begin}(\mathrm{id}_n) = \mathrm{end}(\mathrm{id}_n) = x_1 \cdots x_n$. Thus, id_n is the (abstract) graph $(\{x_1, \ldots, x_n\}, \emptyset, \emptyset, \emptyset, \emptyset, \langle x_1, \ldots, x_n \rangle, \langle x_1, \ldots, x_n \rangle)$. In particular, id_0 is the empty graph. □

Lemma 6.
(1) Concatenation is associative, i.e., if $|\mathrm{end}(g_1)| = |\mathrm{begin}(g_2)|$ and $|\mathrm{end}(g_2)| = |\mathrm{begin}(g_3)|$, then $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$.
(2) The id_n are identities with respect to concatenation, i.e., $g \circ \mathrm{id}_n = g$ and $\mathrm{id}_n \circ h = h$ for every g with $|\mathrm{end}(g)| = n$ and h with $|\mathrm{begin}(h)| = n$.
(3) Sum is associative with unity id_0, i.e., $(g_1 \oplus g_2) \oplus g_3 = g_1 \oplus (g_2 \oplus g_3)$ and $g \oplus \mathrm{id}_0 = \mathrm{id}_0 \oplus g = g$.
(4) For every $m, n \in \mathbf{N}$, $\mathrm{id}_{m+n} = \mathrm{id}_m \oplus \mathrm{id}_n$.
(5) Concatenation and sum satisfy the law of strict monoidality*:*
if $|\mathrm{end}(g)| = |\mathrm{begin}(g')|$ and $|\mathrm{end}(h)| = |\mathrm{begin}(h')|$, then

$$(g \oplus h) \circ (g' \oplus h') = (g \circ g') \oplus (h \circ h').$$

Proof. (1) It is easy to verify that concatenation is associative for atoms. Now let σ_i be a symbol with the same type as g_i, and let ϕ be the replacement with $\phi(\sigma_i) = g_i$. Then $\phi((\mathrm{atom}(\sigma_1) \circ \mathrm{atom}(\sigma_2)) \circ \mathrm{atom}(\sigma_3)) = \phi(\mathrm{atom}(\sigma_1) \circ$

$(\mathrm{atom}(\sigma_2) \circ \mathrm{atom}(\sigma_3))$, and, by Lemma 4, $\phi((\mathrm{atom}(\sigma_1) \circ \mathrm{atom}(\sigma_2)) \circ \mathrm{atom}(\sigma_3)) = (\phi(\mathrm{atom}(\sigma_1)) \circ \phi(\mathrm{atom}(\sigma_2))) \circ \phi(\mathrm{atom}(\sigma_3)) = (g_1 \circ g_2) \circ g_3$ and $\phi(\mathrm{atom}(\sigma_1) \circ (\mathrm{atom}(\sigma_2) \circ \mathrm{atom}(\sigma_3)) = \phi(\mathrm{atom}(\sigma_1)) \circ (\phi(\mathrm{atom}(\sigma_2)) \circ \phi(\mathrm{atom}(\sigma_3))) = g_1 \circ (g_2 \circ g_3)$.

Properties (2), (3), and (5) can be shown in exactly the same way: by verifying them for atoms, and applying Lemma 4. Note that $\phi(\mathrm{id}_n) = \mathrm{id}_n$ for every replacement ϕ. Property (4) is obvious. □

Lemma 6 means that GR is a strict monoidal category (or x-category), see, e.g., [EhrKKK, Hot1, Ben]. The objects of this category are the natural numbers in $\mathbf{N}$, and each (abstract) graph of type (m, n) is a morphism from m to n in this category. Concatenation is the composition of morphisms (but is usually written $h \circ g$ rather than $g \circ h$), and the id_n are the identity morphisms. The set of objects and the set of morphisms form a monoid with respect to $+$ and $\oplus$, respectively (where $+$ is ordinary addition for natural numbers, with monoid identity 0).

We now show that all graphs can be built from a small number of elementary graphs with the operations of concatenation and sum.

For $m, n \in \mathbf{N}$, let $I_{m,n}$ be the graph of type (m, n) with one node x, no edges, $\mathrm{begin}(I_{m,n}) = x^m = \langle x, \ldots, x \rangle$ (m times), and $\mathrm{end}(I_{m,n}) = x^n = \langle x, \ldots, x \rangle$ (n times). Note that $I_{1,1} = \mathrm{id}_1$. Let π_{12} be the graph of type $(2, 2)$ with two nodes x and y, no edges, $\mathrm{begin}(\pi_{12}) = xy$, and $\mathrm{end}(\pi_{12}) = yx$. For every typed alphabet Σ we define the set of *elementary* graphs over Σ by

$$\mathrm{EL}(\Sigma) = \{\mathrm{atom}(\sigma) \mid \sigma \in \Sigma\} \cup \{I_{0,1}, I_{1,0}, I_{1,2}, I_{2,1}, \pi_{12}, \mathrm{id}_0\}.$$

Theorem 7. *For every typed alphabet Σ, $\mathrm{GR}(\Sigma)$ is the smallest class of graphs containing $\mathrm{EL}(\Sigma)$ and closed under $\circ$ and $\oplus$.*

Proof. We have to show that every graph in $\mathrm{GR}(\Sigma)$ can be written as an expression with the operators $\circ$ and $\oplus$, and constants from $\mathrm{EL}(\Sigma)$. We do this by reducing the problem to smaller and smaller sets of graphs. First we reduce it to the class of discrete graphs, i.e., graphs without edges.

Let $g \in \mathrm{GR}(\Sigma)$, and let e be an edge of g with $l_g(e) = \sigma$. We will remove e from g, and express g in terms of the so obtained graph g' that has one edge less than g (and in terms of discrete graphs). By repeating this procedure, we can express g in terms of discrete graphs only. Let $g' = (V_g, E_g - \{e\}, s, t, l, \mathrm{begin}(g), \mathrm{end}(g) \cdot s_g(e) \cdot t_g(e))$ where s, t, l are the restrictions of s_g, t_g, l_g to $E_g - \{e\}$, respectively. Thus, g' is obtained from g by removing e; moreover, in order to be able to reconstruct g from g', the sources and targets of e are turned into end nodes. It is now easy to verify that

$$g = g' \circ (\mathrm{id}_n \oplus \mathrm{backfold}(\mathrm{atom}(\sigma)))$$

where $n = |\mathrm{end}(g)|$, and the backfold operation is the one defined at the end of Section 2.2. Intuitively, the end nodes of $\mathrm{atom}(\sigma)$ are turned into begin nodes (by the backfold operation), and then they are glued to the new end nodes of g'. It is easy to prove that, for every graph h,

$$\mathrm{backfold}(h) = (h \oplus \mathrm{id}_q) \circ \mathrm{backfold}(\mathrm{id}_q)$$

where $q = |\mathrm{end}(h)|$. Consequently,

$$g = g' \circ (\mathrm{id}_n \oplus (\mathrm{atom}(\sigma) \oplus \mathrm{id}_q) \circ \mathrm{backfold}(\mathrm{id}_q))$$

where $q = |t(e)|$. This shows that g can be expressed in terms of g' and discrete graphs.

It remains to find an expression for every discrete graph. To this aim we define the following special *permutation graphs*. Let $k \geq 1$ and let α be a permutation of $\{1, \ldots, k\}$. Then π_α is the discrete graph with nodes $\{x_1, \ldots, x_k\}$, $\mathrm{begin}(\pi_\alpha) = x_1 \cdots x_k$, and $\mathrm{end}(\pi_\alpha) = x_{\alpha(1)} \cdots x_{\alpha(k)}$. Note that π_{12} is the permutation graph π_α with $\alpha(1) = 2$ and $\alpha(2) = 1$. We need some simple properties of permutation graphs. In what follows we write $[n]$ for $\{1, \ldots, n\}$, for every $n \in \mathbf{N}$. First, if α and β are permutations of $[k]$, then $\pi_\alpha \circ \pi_\beta = \pi_{\alpha \circ \beta}$, and if id is the identity permutation of $[k]$, then $\pi_{\mathrm{id}} = \mathrm{id}_k$. Second, let g be a graph of type (m, n) with $V_g = \{x_1, \ldots, x_k\}$, $\mathrm{begin}(g) = x_{\gamma(1)} \cdots x_{\gamma(m)}$, and $\mathrm{end}(g) = x_{\delta(1)} \cdots x_{\delta(n)}$, where $\gamma : [m] \to [k]$ and $\delta : [n] \to [k]$. If α is a permutation of $[n]$, then $g \circ \pi_\alpha$ is the same graph as g except that $\mathrm{end}(g \circ \pi_\alpha) = x_{\delta(\alpha(1))} \cdots x_{\delta(\alpha(n))}$. This means that $g \circ \pi_\alpha$ is obtained from g by applying permutation α to $\mathrm{end}(g)$. Similarly, if α is a permutation of $[m]$, then $\pi_\alpha \circ g$ is the same graph as g except that $\mathrm{begin}(\pi_\alpha \circ g) = x_{\gamma(\alpha^{-1}(1))} \cdots x_{\gamma(\alpha^{-1}(m))}$. Thus, to obtain $\pi_\alpha \circ g$ from g, permutation α^{-1} is applied to $\mathrm{begin}(g)$.

Now let g be an arbitrary discrete graph, with $\mathrm{type}(g) = (m, n)$, $V_g = \{x_1, \ldots, x_k\}$, $\mathrm{begin}(g) = x_{\gamma(1)} \cdots x_{\gamma(m)}$, and $\mathrm{end}(g) = x_{\delta(1)} \cdots x_{\delta(n)}$, where $\gamma : [m] \to [k]$ and $\delta : [n] \to [k]$. For every $1 \leq i \leq k$, let p_i be the number of occurrences of x_i in $\mathrm{begin}(g)$, and let q_i be the number of occurrences of x_i in $\mathrm{end}(g)$. Let α be any permutation of $[m]$ such that $x_{\gamma(\alpha(1))} \cdots x_{\gamma(\alpha(m))} = x_1^{p_1} \cdots x_k^{p_k}$, and let β be any permutation of $[n]$ such that $x_{\delta(\beta(1))} \cdots x_{\delta(\beta(n))} = x_1^{q_1} \cdots x_k^{q_k}$. Thus, intuitively, α and β order $\mathrm{begin}(g)$ and $\mathrm{end}(g)$, respectively. Clearly, by the above properties of permutation graphs, the graph $\pi_{\alpha^{-1}} \circ g \circ \pi_\beta$ has the same nodes as g, has begin nodes $x_1^{p_1} \cdots x_k^{p_k}$, and has end nodes $x_1^{q_1} \cdots x_k^{q_k}$. Hence $\pi_{\alpha^{-1}} \circ g \circ \pi_\beta = I_{p_1,q_1} \oplus \cdots \oplus I_{p_k,q_k}$. By multiplying with π_α to the left, and with $\pi_{\beta^{-1}}$ to the right, we obtain that $g = \pi_\alpha \circ (I_{p_1,q_1} \oplus \cdots \oplus I_{p_k,q_k}) \circ \pi_{\beta^{-1}}$.

It now remains to find expressions for all graphs $I_{m,n}$ and all graphs π_α. The following equations show how to find an expression for $I_{m,n}$:

$$\begin{array}{ll}
I_{1,1} = I_{1,2} \circ I_{2,1} & \\
I_{m+1,1} = (I_{m,1} \oplus I_{1,1}) \circ I_{2,1} & \text{for } m \geq 2 \\
I_{1,n+1} = I_{1,2} \circ (I_{1,n} \oplus I_{1,1}) & \text{for } n \geq 2 \\
I_{m,n} = I_{m,1} \circ I_{1,n} & \text{for } m, n \in \mathbf{N}.
\end{array}$$

Clearly, the identity graphs can also be expressed: for every $n \in \mathbf{N}$, $\mathrm{id}_n = I_{1,1} \oplus \cdots \oplus I_{1,1}$ (n times). To find an expression for π_α, where α is a permutation of $[k]$, we note that either α is the identity on $[k]$, in which case $\pi_\alpha = \mathrm{id}_k$, or α is the composition of interchanging permutations, where an interchanging permutation α_i interchanges i and $i + 1$ and leaves the other numbers as they are (with $1 \leq i \leq k - 1$). In the latter case, by the property of permutation

graphs mentioned above, π_α is the concatenation of graphs π_{α_i}. Now, clearly, $\pi_{\alpha_i} = \mathrm{id}_{i-1} \oplus \pi_{12} \oplus \mathrm{id}_{k-i-1}$.

This shows that all graphs in $\mathrm{GR}(\Sigma)$ can be expressed in terms of $\circ$, $\oplus$, and the constants in $\mathrm{EL}(\Sigma)$. □

Theorem 7 is analogous to Proposition 3.6 of [BauCou]. It is open whether there exists a complete set of equations (including those of Lemma 6) for the operations in $\{\circ, \oplus\} \cup \mathrm{EL}(\Sigma)$. This would give a result analogous to Theorem 3.10 of [BauCou]. It would characterize $\mathrm{GR}(\Sigma)$ as the free x-category satisfying the equations; such results are shown in [Hot1, Cla] (where $I_{1,0}, I_{1,2}, \pi_{12}$ are denoted U, D, V, respectively).

It is not difficult to show that the set $\mathrm{EL}(\Sigma)$ is minimal, in the sense that if one removes one element from it, then Theorem 7 does not hold any more. Also, it should be clear that the concatenation operation cannot be dropped from Theorem 7, even if one would replace $\mathrm{EL}(\Sigma)$ by another finite set of graphs (because, with sum, only graphs with very small connected components could be built). To show that the sum operation cannot be dropped from Theorem 7, we now discuss the close relationship between the concatenation operation and the notion of pathwidth (introduced in [RobSey]; see also, e.g., [Bod, Klo, EllST]). In the following definition we (slightly) generalize the notion of pathwidth, to (hyper)graphs with begin and end nodes (cf. [Cou3]).

Definition 8. A *path decomposition* of a graph g is a sequence $(V_1, \ldots, V_n)$, $n \geq 1$, of subsets of V_g such that
(1) $\bigcup_{i=1}^{n} V_i = V_g$,
(2) for every $e \in E_g$ there is an i with $s(e) \in V_i^*$ and $t(e) \in V_i^*$,
(3) if $i < k < j$, then $V_i \cap V_j \subseteq V_k$, and
(4) $\mathrm{begin}(g) \in V_1^*$ and $\mathrm{end}(g) \in V_n^*$.

The *width* of $(V_1, \ldots, V_n)$ is $\max\{\#V_i \mid 1 \leq i \leq n\} - 1$, where $\#V_i$ is the cardinality of V_i.

The *pathwidth* of a graph g, denoted $\mathrm{pathwidth}(g)$, is the minimal width of a path decomposition of g. □

The relationship between concatenation and pathwidth is expressed in the following result, which (in view of Theorem 23) is essentially due to [Lau] (see also [Cou3]).

Theorem 9. *Let $k \geq 1$. For every graph g, $\mathrm{pathwidth}(g) \leq k$ if and only if there exist graphs $g_1, \ldots, g_n$, $n \geq 1$, such that $g = g_1 \circ \cdots \circ g_n$ and $\#V_{g_i} \leq k+1$ for every $1 \leq i \leq n$.*

Proof. We prove by induction on n that g has a path decomposition $(V_1, \ldots, V_n)$ of width $\leq k$ if and only if there exist graphs $g_1, \ldots, g_n$ with $\#V_{g_i} \leq k+1$ such that $g = g_1 \circ \cdots \circ g_n$. For $n = 1$ this is obvious.

Assume that g has a path decomposition $(V_1, \ldots, V_n, V_{n+1})$ with $\#V_i \leq k+1$. By condition (3) of Definition 8, $(V_1 \cup \cdots \cup V_n) \cap V_{n+1} = V_n \cap V_{n+1}$. Let g' be the subgraph of g induced by $V_1 \cup \cdots \cup V_n$, such that $\mathrm{begin}(g') = \mathrm{begin}(g)$ and $\mathrm{end}(g)$

consists of the nodes in $V_n \cap V_{n+1}$, in some order. Let g_{n+1} be the subgraph of g induced by V_{n+1} with $\mathrm{begin}(g_{n+1}) = \mathrm{end}(g')$ and $\mathrm{end}(g_{n+1}) = \mathrm{end}(g)$. Clearly, $g = g' \circ g_{n+1}$. Also, $(V_1, \ldots, V_n)$ is a path decomposition of g', and hence, by induction $g' = g_1 \circ \cdots \circ g_n$ with $\#V_{g_i} \leq k+1$, and so $g = g_1 \circ \cdots \circ g_n \circ g_{n+1}$.

Assume now that $g = g_1 \circ \cdots \circ g_n \circ g_{n+1}$ with $\#V_{g_i} \leq k+1$. Let $g' = g_1 \circ \cdots \circ g_n$. Hence $g = g' \circ g_{n+1}$. We may assume that g' and g_{n+1} are disjoint, and that $g = (g' \& g_{n+1})/R$, as in Definition 3. By induction, g' has a path decomposition $(V_1, \ldots, V_n)$ with $\#V_i \leq k+1$. Let $V_{n+1} = V_{g_{n+1}}$. It should now be clear that the sequence $(V_1/R, \ldots, V_n/R, V_{n+1}/R)$ is a path decomposition of g. □

Since there are graphs of arbitrary large pathwidth (such as the complete graph on n nodes, which has pathwidth $n-1$), this theorem implies that, for any typed alphabet Σ, there is no finite subset E of $\mathrm{GR}(\Sigma)$ such that $\mathrm{GR}(\Sigma)$ is the smallest set of graphs containing E and closed under concatenation (because the pathwidth of all graphs in this smallest set is at most equal to the maximal size of the graphs in E).

4 Context-free graph grammars

In this section we use context-free grammars to generate graph expressions that are built from arbitrary constant graphs with the graph operators $\circ$ and $\oplus$. Taking the values of these expressions in GR, each such context-free grammar generates a graph language.

Let CS be the set of operators $\{\circ, \oplus\} \cup \{c_g \mid g \in \mathrm{GR}\}$, where $\circ$ and $\oplus$ denote concatenation and sum of graphs, as usual, and c_g is a constant standing for the graph g.

Expressions over CS are defined in the usual way. Let Σ be a typed alphabet, disjoint with CS (where, intuitively, each $\sigma \in \Sigma$ is a variable that ranges over all graphs with the same type as σ). A (well-formed) *expression* over CS and Σ is a string over $\mathrm{CS} \cup \Sigma \cup \{(,)\}$ defined recursively as follows, together with its *type*: (1) every $\sigma \in \Sigma$ is an expression, with the same type, (2) every constant c_g is an expression, with $\mathrm{type}(c_g) = \mathrm{type}(g)$, (3) if e and f are expressions with $\mathrm{type}(e) = (k, m)$ and $\mathrm{type}(f) = (m, n)$, then $(e \circ f)$ is an expression with $\mathrm{type}(e \circ f) = (k, n)$, and (3) if e and f are expressions with $\mathrm{type}(e) = (m, n)$ and $\mathrm{type}(f) = (p, q)$, then $(e \oplus f)$ is an expression with $\mathrm{type}(e \oplus f) = (m+p, n+q)$. An 'expression over CS' is defined in the same way, without clause (1). If e is an expression over CS, then its *value*, denoted by $\mathrm{val}(e)$, is a graph in GR, defined recursively in the usual way: $\mathrm{val}(c_g) = g$, $\mathrm{val}(e \circ f) = \mathrm{val}(e) \circ \mathrm{val}(f)$, and $\mathrm{val}(e \oplus f) = \mathrm{val}(e) \oplus \mathrm{val}(f)$.

Definition 10. A *context-free graph grammar over CS* is an ordinary context-free grammar $G = (N, T, P, S)$, see Section 2.1, such that N is a typed alphabet, T is a finite subset of $\mathrm{CS} \cup \{(,)\}$, and the right-hand side of each production in P is an expression over CS and N, of the same type as the left-hand side. □

Obviously, the context-free language $L(G)$ generated by G is a set of expressions over CS (and it is also a regular tree language, see [GécSte]). The *graph language*

generated by G is $\mathrm{val}(L(G)) = \{\mathrm{val}(e) \mid e \in L(G)\}$. Note that $\mathrm{type}(\mathrm{val}(L(G))) = \mathrm{type}(S)$. By Val(CFG(CS)) we denote the class of all graph languages generated by context-free graph grammars over CS. By the results of [MezWri], it is the class of *equational* subsets of the algebra of graphs with the operations $\circ$ and $\oplus$.

It should be clear that, due to Theorem 7, we could restrict CS to contain only elementary constants, i.e., constants c_g with $g \in \mathrm{EL}(\Sigma)$ for some Σ. This would give the same class Val(CFG(CS)).

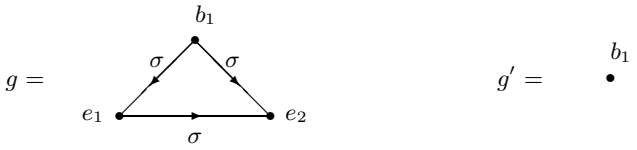

Fig. 2. Graphs g and g'.

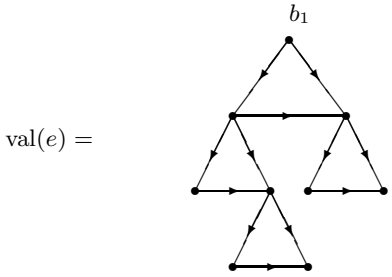

Fig. 3. The value of graph expression e.

As an example, consider the context-free graph grammar G_b that has one nonterminal X, with $\mathrm{type}(X) = (1,0)$, and two productions $X \to c_g \circ (X \oplus X)$ and $X \to c_{g'}$, where g is the triangle of type $(1,2)$ with $V = \{x,y,z\}$, $E = \{(x,y),(x,z),(y,z)\}$, $s(u,v) = u$, $t(u,v) = v$, and $l(u,v) = \sigma$ for every edge (u,v), $\mathrm{begin}(g) = x$ and $\mathrm{end}(g) = yz$, and g' is the graph of type $(1,0)$ with one node x, no edges, $\mathrm{begin}(g') = x$ and $\mathrm{end}(g') = \lambda$. The graphs g and g' are shown in Fig. 2. The expression

$$e = c_g \circ (c_g \circ (c_{g'} \oplus c_g \circ (c_{g'} \oplus c_{g'})) \oplus c_g \circ (c_{g'} \oplus c_{g'}))$$

is in $L(G)$; the graph $\mathrm{val}(e)$ is shown in Fig. 3 (without the edge labels σ). Clearly, $\mathrm{val}(L(G_b))$ is the set of all graphs of type $(1,0)$ that are obtained from (directed, rooted) binary trees by connecting each pair of children by an additional edge; the sequence of begin nodes consists of the root of the binary tree. This graph language is therefore in Val(CFG(CS)).

The main result of this section is that generating graph languages in the above way is equivalent to generating them with HR grammars (see Section 2.3).

Thus, the HR grammars generate exactly the equational subsets of the algebra of graphs with the operations $\circ$ and $\oplus$. As observed in the introduction, this is a simple variant of Theorem 4.11 of [BauCou] (and the proof is analogous).

Theorem 11. $\mathrm{Val}(\mathrm{CFG}(\mathrm{CS})) = \mathrm{HR}$.

Proof. Similar to the restriction on productions of HR grammars, we can assume without loss of generality that no nonterminal occurs more than once in the right-hand side of a production of a context-free graph grammar. Moreover, we can also assume that, in a context-free graph grammar, the nonterminals do not occur as edge labels in the constants c_g that are used in the right-hand sides of its productions.

To turn a context-free graph grammar into an HR grammar, we extend the definition of the 'val' function to expressions over CS and N. This is simply done by extending the recursive definition of 'val' with the requirement that $\mathrm{val}(X) = \mathrm{atom}(X)$ for every $X \in N$.

Let G be a context-free graph grammar and G' an HR grammar. We say that G and G' are *related* if they have the same typed alphabet of nonterminals, with the same initial nonterminal, and $P' = \{X \to \mathrm{val}(t) \mid X \to t \in P\}$, where P is the set of productions of G and P' the one of G'. Trivially, for every context-free graph grammar there is a related HR grammar. The other way around, it suffices to show that for every graph $h \in \mathrm{GR}(N \cup T)$, where N and T are the terminal and nonterminal alphabet of the HR grammar, respectively, there is an expression t over CS and N such that $\mathrm{val}(t) = h$. By Theorem 7 there is an expression e over CS such that $\mathrm{val}(e) = h$, and for every constant c_g that occurs in e, $g \in \mathrm{EL}(N \cup T)$. Let t be the expression that is obtained from e by changing every subexpression $\mathrm{atom}(X)$ into X, for every $X \in N$. Obviously t is the required expression. Hence, for every HR grammar there is a related context-free graph grammar.

It now suffices to show that related grammars G and G' generate the same graph language. To this aim we show that for every nonterminal X and every terminal graph g, $\mathrm{atom}(X) \Rightarrow^* g$ in G' if and only if there exists an expression e over CS such that $X \Rightarrow^* e$ in G and $\mathrm{val}(e) = g$. This can be proved by induction on the length of the derivations, as follows.

Consider a derivation $X \Rightarrow t \Rightarrow^* e$ in G. Let t contain the nonterminals $X_1, \ldots, X_n$ (and recall that each nonterminal X_i occurs exactly once in t). Then there exist expressions e_i such that $X_i \Rightarrow^* e_i$ and $e = \psi(t)$ where ψ is the string homomorphism with $\psi(X_i) = e_i$ and the identity otherwise. Now let ϕ be the replacement with $\phi(X_i) = \mathrm{val}(e_i)$ and $\phi(\sigma) = \mathrm{atom}(\sigma)$ for all terminal symbols.

It is straightforward to show that $\mathrm{val}(\psi(t)) = \phi(\mathrm{val}(t))$, by induction on the structure of the expression t (cf. Proposition 4.7 of [BauCou]). As an example, if $t = t_1 \circ t_2$, then $\mathrm{val}(\psi(t)) = \mathrm{val}(\psi(t_1) \circ \psi(t_2)) = \mathrm{val}(\psi(t_1)) \circ \mathrm{val}(\psi(t_2)) = \phi(\mathrm{val}(t_1)) \circ \phi(\mathrm{val}(t_2)) = \phi(\mathrm{val}(t_1) \circ \mathrm{val}(t_2)) = \phi(\mathrm{val}(t))$, where we have used Lemma 4(1). As another example, if $t = X_i$, then $\phi(\mathrm{val}(t)) = \phi(\mathrm{atom}(X_i)) = \phi(X_i) = \mathrm{val}(e_i) = \mathrm{val}(\psi(t))$.

By induction, $\mathrm{atom}(X_i) \Rightarrow^* \mathrm{val}(e_i)$ in G'. Since G and G' are related, G' has the production $X \to \mathrm{val}(t)$. It is easy to see that $\mathrm{val}(t)$ has n nonterminal edges,

labeled by $X_1, \ldots, X_n$. Hence, by Proposition 2, $\mathrm{atom}(X) \Rightarrow \mathrm{val}(t) \Rightarrow^* \phi(\mathrm{val}(t))$. This shows that $\mathrm{atom}(X) \Rightarrow^* \mathrm{val}(\psi(t)) = \mathrm{val}(e)$.

The proof in the other direction is similar and is left to the reader. □

Since the concept of *related* grammars, as discussed in the above proof, preserves the number of nonterminals in the right-hand sides of productions, the above result is also true in the linear case. By Val(LIN-CFG(CS)) we denote the class of languages generated by linear context-free graph grammars over CS.

Corollary 12. Val(LIN-CFG(CS)) = LIN-HR.

However, in the linear case, the form of the context-free graph grammar can even be restricted to be "right-linear" in the following sense. A context-free graph grammar over CS is *right-linear* if its productions are of the form $X \to c_g \circ Y$ or of the form $X \to c_g$, where X and Y are nonterminals. Note in particular that $\oplus$ is not needed. By Val(RLIN-CFG(CS)) we denote the class of graph languages generated by right-linear context-free graph grammars over CS.

Theorem 13. Val(RLIN-CFG(CS)) = LIN-HR.

Proof. By Corollary 12, it suffices to show that LIN-HR $\subseteq$ Val(RLIN-CFG(CS)). Let L be a graph language in LIN-HR, and let $G = (N, T, P, S)$ be a linear HR grammar generating L.

We first consider the case that for every $X \in N$ there exists $m \in \mathbf{N}$ such that $\mathrm{type}(X) = (m, 0)$. By the proof of Theorem 11 it suffices to construct a context-free graph grammar G' that is related to G. G' has the same nonterminal alphabet as G, with the same initial nonterminal. G' has the set of productions $P' = \{p' \mid p \in P\}$, where, for each $p \in P$, p' is defined as follows. Let p be the production $X \to g$. If $g \in \mathrm{GR}(T)$, then we define p' to be $X \to c_g$. Otherwise, g has exactly one edge e that is labeled with a nonterminal, say, Y. Note that $\mathrm{end}(g) = \lambda$ and $t_g(e) = \lambda$. Then we define p' to be the production $X \to c_{g'} \circ Y$, where $g' = (V_g, E_g - \{e\}, s, t, l, \mathrm{begin}(g), s_g(e))$ and s, t, l are the restrictions of s_g, t_g, l_g to $E_g - \{e\}$. Clearly, $\mathrm{val}(c_{g'} \circ Y) = g' \circ \mathrm{atom}(Y) = g$. Hence G and G' are indeed related. Note that the construction of g' from g is a special case of that in the first part of the proof of Theorem 7.

We now consider the general case. To be able to use the above special case, define the LIN-HR grammar $\overline{G} = (\overline{N}, T, \overline{P}, S)$, where $\overline{N}$ is the same set as N with a different type function: if $\mathrm{type}(X) = (m, n)$ in N, then $\mathrm{type}(X) = (m + n, 0)$ in $\overline{N}$. For every graph $h \in \mathrm{GR}(N \cup T)$ we define the graph $\overline{h} = (V_h, E_h, s, t, l_h, \mathrm{begin}(h) \cdot \mathrm{end}(h), \lambda)$ where, for $e \in E_h$, $s(e)$ and $t(e)$ are defined as follows: if $l_h(e) \in T$, then $s(e) = s_h(e)$ and $t(e) = t_h(e)$; if $l_h(e) \in N$, then $s(e) = s_h(e) \cdot t_h(e)$ and $t(e) = \lambda$. Note that if $h \in \mathrm{GR}(T)$ then $\overline{h} = \mathrm{backfold}(h)$. We now define $\overline{P} = \{X \to \overline{h} \mid X \to h \in P\}$. This ends the definition of $\overline{G}$. It is straightforward to show that $L(\overline{G}) = \{\mathrm{backfold}(g) \mid g \in L(G)\}$. In fact, the derivations of $\overline{G}$ are exactly all $\mathrm{atom}(S) \Rightarrow \overline{g_1} \Rightarrow \overline{g_2} \Rightarrow \cdots \Rightarrow \overline{g_n}$ where $\mathrm{atom}(S) \Rightarrow g_1 \Rightarrow g_2 \Rightarrow \cdots \Rightarrow g_n$ is a derivation of G. Since $\overline{G}$ satisfies the above special case, we conclude that $\mathrm{backfold}(L)$ is in Val(RLIN-CFG(CS)).

Suppose that $\mathrm{type}(L) = (m, n)$. It is easy to verify, for every g of type (m, n), that

$$g = (\mathrm{id}_m \oplus \mathrm{fold}(\mathrm{id}_n)) \circ (\mathrm{backfold}(g) \oplus \mathrm{id}_n).$$

Hence $L = \{(\mathrm{id}_m \oplus \mathrm{fold}(\mathrm{id}_n)) \circ (h \oplus \mathrm{id}_n) \mid h \in \mathrm{backfold}(L)\}$. Thus, it now suffices to show that if L' is in Val(RLIN-CFG(CS)), then so are all languages $\{h \oplus \mathrm{id}_n \mid h \in L'\}$, for $n \in \mathbf{N}$, and all languages $\{g_0 \circ h \mid h \in L'\}$, for $g_0 \in \mathrm{GR}$. To this aim, let G' be a right-linear context-free graph grammar generating L'.

Change, in the productions of G', every constant c_g into the constant $c_{g \oplus \mathrm{id}_n}$. Clearly, the resulting right-linear context-free graph grammar generates all graphs $(g_1 \oplus \mathrm{id}_n) \circ \cdots \circ (g_k \oplus \mathrm{id}_n)$ with $g_1 \circ \cdots \circ g_k \in L'$. By the law of strict monoidality (Lemma 6(5)),

$$(g_1 \oplus \mathrm{id}_n) \circ \cdots \circ (g_k \oplus \mathrm{id}_n) = (g_1 \circ \cdots \circ g_k) \oplus (\mathrm{id}_n \circ \cdots \circ \mathrm{id}_n) = (g_1 \circ \cdots \circ g_k) \oplus \mathrm{id}_n.$$

This proves that the resulting grammar generates $\{g \oplus \mathrm{id}_n \mid g \in L'\}$.

Introduce a new initial nonterminal S', and add to G' all the productions $S' \to c_{g_0 \circ g} \circ Y$ and $S' \to c_{g_0 \circ g}$ such that $S \to c_g \circ Y$ and $S \to c_g$ are productions of G', respectively (where S is the old initial nonterminal of G'). Clearly, the resulting right-linear grammar generates all graphs $(g_0 \circ g_1) \circ g_2 \circ \cdots \circ g_k$ with $g_1 \circ \cdots \circ g_k \in L'$. In other words (using the associativity of concatenation), it generates the graph language $\{g_0 \circ h \mid h \in L'\}$. Note that we could also have added the one production $S' \to c_{g_0} \circ S$; the reason for not doing so will become clear in the proof of Theorem 27. □

This result suggests that for context-free graph grammars there is no difference between the linear and the right-linear case, as opposed to the case of ordinary context-free grammars (where the right-linear grammars generate the regular languages which form a proper subclass of the linear languages). More support for this intuition will be given in the next section.

5 Strings Denote Graphs

Since concatenation of graphs is associative, strings can be viewed as expressions that denote graphs. Thus, as an even simpler variation of the approach with expression generating context-free grammars in Section 4, we can use all possible string grammars to generate graph languages. More generally, every class K of string languages defines a class $\mathrm{Int}(K)$ of graph languages (where Int stands for 'interpretation', which is similar to Val in Section 4). An "interpretation" is a mapping that associates a graph with each symbol of an alphabet.

Definition 14. Let A be an alphabet. An *interpretation of* A is a mapping $h : A \to \mathrm{GR}$; h is extended to a (partial) function from A^* to GR by

$$h(a_1 a_2 \cdots a_n) = h(a_1) \circ h(a_2) \circ \cdots \circ h(a_n)$$

with $n \geq 1$ and $a_i \in A$ for all $1 \leq i \leq n$. □

Note that the extended h is partial because the types of the $h(a_i)$ may not fit; moreover, $h(\lambda)$ is undefined (where λ is the empty string). Thus, the only "technical trouble" is that the concatenation of graphs is typed whereas the concatenation of strings is always possible. To deal with this, the following lemma is useful. It says that the domain of an interpretation is regular.

Lemma 15. *For every interpretation $h : A \to \mathrm{GR}$, the language $\{w \in A^* \mid h(w)$ is defined$\}$ is regular.*

Proof. Clearly, $h(a_1 a_2 \cdots a_n)$ is defined if and only if $n \geq 1$ and $|\mathrm{end}(h(a_i))| = |\mathrm{begin}(h(a_{i+1}))|$ for every $1 \leq i < n$. It is easy to construct a finite automaton that checks this. □

For a string language $L \subseteq A^*$, we define, as usual, the set of graphs $h(L) = \{g \in \mathrm{GR} \mid g = h(w)$ for some $w \in L\}$; note that $h(L)$ need not be a graph language (in our particular meaning of the term, as defined in Section 2.2) because not all graphs need have the same type.

Definition 16. Let K be a class of string languages. The *interpretation* of K is $\mathrm{Int}(K) = \{h(L) \mid L \in K, h : A \to \mathrm{GR}$ with $L \subseteq A^*, h(L)$ is a graph language$\}$. □

In other words, $\mathrm{Int}(K)$ consists of all graph languages $h(L)$, where L is any language in K and h is any mapping from the symbols of L to graphs. Intuitively, h determines the interpretation of the symbols, and then the concatenation of those symbols is interpreted as concatenation of the corresponding graphs.

It is an immediate consequence of Theorem 9 that every graph language $h(L)$ in $\mathrm{Int}(K)$ is *of bounded pathwidth*, i.e., there exists k such that $\mathrm{pathwidth}(g) \leq k$ for every $g \in h(L)$. In fact, if $L \subseteq A^*$, then $k = \max\{\#V_{h(a)} \mid a \in A\} - 1$.

Corollary 17. *For every K, every graph language in $\mathrm{Int}(K)$ is of bounded pathwidth.*

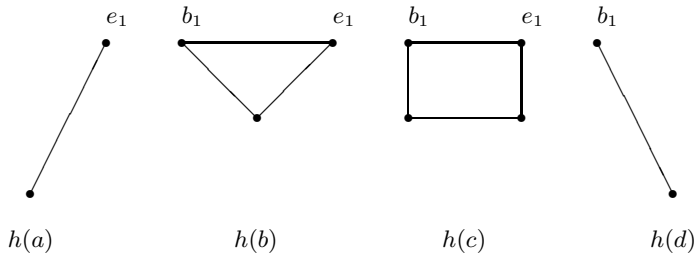

Fig. 4. An interpretation.

The first class K of interest is the class REG of regular languages. An example of a graph language in Int(REG), of type $(0,0)$, is $h(a(b \cup c)^* d)$ where the graphs $h(a)$, $h(b)$, $h(c)$, and $h(d)$ are shown in Fig. 4 (without edge directions and edge labels). The graph $h(abbcbd)$ is shown in Fig. 5. Clearly, the graph language $h(a(b \cup c)^* d)$ consists of all "clothes lines" on which triangles and rectangles are hanging to dry. We first present a characterization of Int(REG) by regular ex-

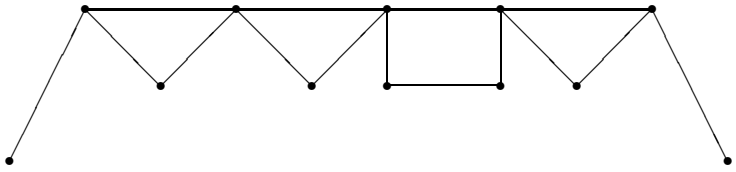

Fig. 5. Graph interpretation of the string *abbcbd*.

pressions, corresponding to the characterization of REG by regular expressions. To this aim we define the operations of union, concatenation, and (Kleene) star for graph languages. The operation of graph concatenation is extended to graph languages L and L' in the usual way: if $\mathrm{type}(L) = (k, m)$ and $\mathrm{type}(L') = (m, n)$, then their *concatenation* is defined by $L \circ L' = \{g \circ g' \mid g \in L, g' \in L'\}$. Then, in the obvious way, the *star* of a graph language is defined by iterated concatenation: for a graph language L with $\mathrm{type}(L) = (k, k)$ for some $k \in \mathbf{N}$, $L^* = \bigcup_{n \in \mathbf{N}} L^n$ where $L^n = L \circ \cdots \circ L$ (n times) for $n \geq 1$, and $L^0 = \{\mathrm{id}_k\}$. Also, $L^+ = \bigcup_{n \geq 1} L^n$ is the (Kleene) plus of L. Finally, the *union* $L \cup L'$ of two graph languages L and L' is defined only when $\mathrm{type}(L) = \mathrm{type}(L')$ (otherwise it would not be a graph language). Thus, the operations of union, concatenation, and star are also typed operations on graph languages (as opposed to the case of string languages for which they are always defined). Let $\mathrm{REX}(\cup, \circ, *, \mathrm{SING})$ denote the smallest class of graph languages containing the empty graph language and all singleton graph languages, and closed under the operations union, concatenation, and star. Thus, it is the class of all graph languages that can be denoted by (the usual) regular expressions, where the symbols of the alphabet denote singleton graph languages. As an example, the above graph language of clothes lines is in $\mathrm{REX}(\cup, \circ, *, \mathrm{SING})$ because it can be written as $\{h(a)\} \circ (\{h(b)\} \cup \{h(c)\})^* \circ \{h(d)\}$.

Theorem 18. $\mathrm{Int}(\mathrm{REG}) = \mathrm{REX}(\cup, \circ, *, \mathrm{SING})$.

Proof. We have to cope with the "technical trouble" of typing, in particular with the empty string. Note that, for a graph language L with $\mathrm{type}(L) = (k, k)$, $L^* = L^+ \cup \{\mathrm{id}_k\}$ and $L^+ = L \circ L^*$. This shows that we can replace star by plus, i.e., $\mathrm{REX}(\cup, \circ, *, \mathrm{SING}) = \mathrm{REX}(\cup, \circ, +, \mathrm{SING})$, the smallest class of graph languages

containing the empty graph language and all singleton graph languages, and closed under the operations union, concatenation, and plus.

To show that $\mathrm{REX}(\cup, \circ, +, \mathrm{SING}) \subseteq \mathrm{Int}(\mathrm{REG})$, it suffices to prove that $\mathrm{Int}(\mathrm{REG})$ contains the empty language and all singleton graph languages, and that it is closed under union, concatenation, and plus. Clearly, $h(\emptyset) = \emptyset$ for any interpretation h. Also, if $h(a) = g$, then $h(\{a\}) = \{g\}$. Now let $L_1 \subseteq A_1^*$ and $L_2 \subseteq A_2^*$ be regular languages, and let h_1 and h_2 be interpretations of A_1 and A_2, respectively, such that $h_1(L_1)$ and $h_2(L_2)$ are graph languages in $\mathrm{Int}(\mathrm{REG})$. Obviously, by a renaming of symbols, we may assume that A_1 and A_2 are disjoint. Let $h = h_1 \cup h_2$ be the interpretation of $A_1 \cup A_2$ that extends both h_1 and h_2. It is easy to verify that (with the appropriate conditions on types) $h_1(L_1) \cup h_2(L_2) = h(L_1 \cup L_2)$, $h_1(L_1) \circ h_2(L_2) = h(L_1 \cdot L_2)$, and $h_1(L_1)^+ = h(L_1^+)$, which shows that these graph languages are also in $\mathrm{Int}(\mathrm{REG})$.

To show that $\mathrm{Int}(\mathrm{REG}) \subseteq \mathrm{REX}(\cup, \circ, +, \mathrm{SING})$, we first note that, since an interpretation is undefined for the empty string, $\mathrm{Int}(\mathrm{REG}) = \mathrm{Int}(\mathrm{REG} - \lambda)$, where $\mathrm{REG} - \lambda = \{L - \{\lambda\} \mid L \in \mathrm{REG}\}$ is the class of all λ-free regular languages. It is well known (and easy to prove) that $\mathrm{REG} - \lambda$ is the smallest class of languages containing the empty language and all languages $\{a\}$ where a is a symbol, and closed under the operations union, concatenation, and plus. By induction on this characterization we show that for every language $L \in \mathrm{REG} - \lambda$ and every interpretation h of the alphabet of L, if $h(w)$ is defined for every $w \in L$, and $h(L)$ is a graph language, then $h(L) \in \mathrm{REX}(\cup, \circ, +, \mathrm{SING})$. Note that by Lemma 15 (and the fact that REG is closed under intersection) we can indeed assume that h is defined for all strings in L. The inductive proof is as follows. If L is empty, then so is $h(L)$. If $L = \{a\}$, then $h(L)$ is a singleton. If $L = L_1 \cup L_2$, then $h(L) = h(L_1) \cup h(L_2)$. Now let $L = L_1 \cdot L_2$ and assume that L_1 and L_2 are nonempty (otherwise L is empty). Since, by assumption, $h(L_1 \cdot L_2)$ is a graph language and $h(w)$ is defined for every $w \in L_1 \cdot L_2$, $h(L_1)$ and $h(L_2)$ are also graph languages; for $h(L_1)$ this is proved as follows: if $w_1, w_1' \in L_1$, then, for any $w_2 \in L_2$, $h(w_1 \cdot w_2) = h(w_1) \circ h(w_2)$ and similarly for w_1', and so $|\mathrm{begin}(h(w_1))| = |\mathrm{begin}(h(w_1 \cdot w_2))| = |\mathrm{begin}(h(w_1' \cdot w_2))| = |\mathrm{begin}(h(w_1'))|$ and $|\mathrm{end}(h(w_1))| = |\mathrm{begin}(h(w_2))| = |\mathrm{end}(h(w_1'))|$. Hence $h(L) = h(L_1 \cdot L_2) = h(L_1) \circ h(L_2)$. Finally, let $L = L_1^+$. Then $h(L_1)$ is a graph language of some type (k, k) by an argument similar to the one above, and $h(L) = h(L_1)^+$. □

This result holds in fact for sets of morphisms of arbitrary categories (instead of the category GR of graphs, cf. Lemma 6). It generalizes a well-known characterization of the rational subsets of a monoid (see, e.g., Proposition III.2.2 of [Ber]).

The characterization of Theorem 18 still holds after adding the sum operation, extended to graph languages in the usual way: for arbitrary graph languages L and L', $L \oplus L' = \{g \oplus g' \mid g \in L, g' \in L'\}$. In other words, $\mathrm{Int}(\mathrm{REG}) = \mathrm{REX}(\cup, \circ, *, \oplus, \mathrm{SING})$, the smallest class of graph languages containing the empty graph language and all singleton graph languages, and closed under the operations union, concatenation, star, and sum. This is because of the following simple reason.

Lemma 19. *For every class of languages K, if* $\mathrm{Int}(K)$ *is closed under concatenation, then it is closed under sum.*

Proof. We first show that if M is in $\mathrm{Int}(K)$ then so is $M \oplus \{\mathrm{id}_k\}$ for every k. This was shown for Val(RLIN-CFG(CS)) in the proof of Theorem 13, and the following argument is the same as the one used there. Let $M = h(L)$ for some $L \in K$ and some interpretation h of the alphabet A of L. Define $h'(a) = h(a) \oplus \mathrm{id}_k$ for every $a \in A$. Then $h'(a_1 \cdots a_n) =$

$$(h(a_1) \oplus \mathrm{id}_k) \circ \cdots \circ (h(a_n) \oplus \mathrm{id}_k) = (h(a_1) \circ \cdots \circ h(a_n)) \oplus (\mathrm{id}_k \circ \cdots \circ \mathrm{id}_k)$$

because of strict monoidality (Lemma 6(5)), and the last expression is equal to $h(a_1 \cdots a_n) \oplus \mathrm{id}_k$. This implies that $h'(L) = h(L) \oplus \{\mathrm{id}_k\} = M \oplus \{\mathrm{id}_k\}$. Similarly it can be shown that $\{\mathrm{id}_k\} \oplus M$ is in $\mathrm{Int}(K)$.

Now, for arbitrary graph languages M and M' with $\mathrm{type}(M) = (m, n)$ and $\mathrm{type}(M') = (m', n')$, $M \oplus M' = (M \circ \{\mathrm{id}_n\}) \oplus (\{\mathrm{id}_{m'}\} \circ M') = (M \oplus \{\mathrm{id}_{m'}\}) \circ (\{\mathrm{id}_n\} \oplus M')$ by strict monoidality. Hence, by the above, and the fact that $\mathrm{Int}(K)$ is closed under $\circ$, $M \oplus M'$ is in $\mathrm{Int}(K)$. □

If we allow $\oplus$ in our regular expressions, then, as should be clear from Theorem 7, we do not need all singleton graph languages to start with, but only those that contain elementary graphs (i.e., graphs that belong to some $\mathrm{EL}(\Sigma)$, as defined in Section 3). Recall that a graph is elementary if it is an atom or one of the graphs $I_{0,1}$, $I_{1,0}$, $I_{1,2}$, $I_{2,1}$, π_{12}, or id_0. Let $\mathrm{REX}(\cup, \circ, *, \oplus, \mathrm{ELSING})$ denote the smallest class of graph languages containing the empty graph language and all singleton graph languages with an elementary graph as element, and closed under the operations union, concatenation, star, and sum.

Theorem 20. $\mathrm{Int}(\mathrm{REG}) = \mathrm{REX}(\cup, \circ, *, \oplus, \mathrm{ELSING})$.

Note that this result is closer to the corresponding result for regular languages, for which only singleton languages $\{a\}$ are needed where a is a symbol.

The next class K of interest is the class CF of context-free languages. We will show that Int(CF) is a (proper) subclass of HR, the class of graph languages generated by HR grammars. In fact, it is rather obvious that Int(CF) is exactly the class of languages generated by context-free graph grammars over CS that do not use the sum operation. Thus, Int(CF) is the class of equational subsets of the algebra of graphs with the concatenation operation. Inclusion in HR then follows from Theorem 11. As in the previous theorems, we have to cope with the technical trouble of typing.

Let $\mathrm{CS}_\circ = \mathrm{CS} - \{\oplus\} = \{\circ\} \cup \{c_g \mid g \in \mathrm{GR}\}$. If $\oplus$ does not occur in the productions of a context-free graph grammar over CS, then we also say that it is over $\mathrm{CS}_\circ$. By $\mathrm{Val}(\mathrm{CFG}(\mathrm{CS}_\circ))$ we denote the class of all graph languages generated by context-free graph grammars over $\mathrm{CS}_\circ$. Note that, by definition, $\mathrm{Val}(\text{RLIN-CFG}(\mathrm{CS})) \subseteq \mathrm{Val}(\mathrm{CFG}(\mathrm{CS}_\circ))$.

Theorem 21. $\mathrm{Int}(\mathrm{CF}) = \mathrm{Val}(\mathrm{CFG}(\mathrm{CS}_\circ))$.

Proof. We first show that $\mathrm{Val}(\mathrm{CFG}(\mathrm{CS}_\circ)) \subseteq \mathrm{Int}(\mathrm{CF})$. Let $G = (N, T, P, S)$ be a context-free graph grammar over $\mathrm{CS}_\circ$. Every production of G is of the form $X \to \alpha_1 \circ \alpha_2 \circ \cdots \circ \alpha_k$ with $k \geq 1$ and $\alpha_i \in N$ or $\alpha_i = c_g$ for some $g \in \mathrm{GR}$. Note that we can drop the parentheses from the right-hand sides, due to the associativity of concatenation. Thus, G generates expressions of the form $c_{g_1} \circ \cdots \circ c_{g_n}$ with $n \geq 1$, and $\mathrm{val}(L(G)) = \{g_1 \circ \cdots \circ g_n \mid c_{g_1} \circ \cdots \circ c_{g_n} \in L(G)\}$.

Define the (ordinary) context-free grammar $G' = (N, T', P', S)$ where T' is the set of all c_g in T, and $P' = \{X \to \alpha_1\alpha_2 \cdots \alpha_k \mid X \to \alpha_1 \circ \alpha_2 \circ \cdots \circ \alpha_k \in P\}$. Obviously, $L(G') = \{c_{g_1} \cdots c_{g_n} \mid c_{g_1} \circ \cdots \circ c_{g_n} \in L(G)\}$. Now let $h : T' \to \mathrm{GR}$ be the interpretation such that $h(c_g) = g$. Then, clearly, $\mathrm{val}(L(G)) = h(L(G'))$. Hence $\mathrm{val}(L(G))$ is in $\mathrm{Int}(\mathrm{CF})$.

We now show that $\mathrm{Int}(\mathrm{CF}) \subseteq \mathrm{Val}(\mathrm{CFG}(\mathrm{CS}_\circ))$. By Lemma 15 (and the fact that CF is closed under intersection with regular languages), every $\mathrm{Int}(\mathrm{CF})$ graph language is of the form $h(L)$ where L is a context-free language such that h is defined for every string in L. In particular, L is λ-free. Let $G = (N, T, P, S)$ be a context-free grammar generating L. We may assume that the right-hand sides of the productions of G are non-empty. Define the context-free grammar $G' = (N, T', P', S)$ such that $T' = \{\circ\} \cup \{c_{h(a)} \mid a \in T\}$, and $P' = \{X \to \psi(\alpha_1) \circ \psi(\alpha_2) \circ \cdots \circ \psi(\alpha_k) \mid X \to \alpha_1\alpha_2 \cdots \alpha_k \in P\}$, where $\psi(a) = c_{h(a)}$ for every $a \in T$, and $\psi(X) = X$ for every $X \in N$. Obviously, $L(G') = \{c_{h(a_1)} \circ \cdots \circ c_{h(a_n)} \mid a_1 \cdots a_n \in L(G)\}$, and so $\mathrm{val}(L(G')) = h(L(G)) = h(L)$.

The only thing that remains to be proved (and this is the "technical trouble") is that G' is a context-free graph grammar over CS (and it obviously is over $\mathrm{CS}_\circ$). In other words, we have to turn N into a typed alphabet, such that the right-hand sides of the productions are expressions with the same type as the left-hand sides. To this aim we investigate the grammar G in more detail.

We claim that for all strings $w \in T^*$ generated by a given nonterminal X of G, $h(w)$ is defined and $\mathrm{type}(h(w))$ is the same for all such w. Here we will use the fact that h is defined for all strings in L. The proof is similar to the argument used at the end of the proof of Theorem 18. Let $X \Rightarrow^* w_1$ and $X \Rightarrow^* w_2$, with $w_i \in T^*$. Consider some $u, v \in T^*$ such that $S \Rightarrow^* uXv$ (assuming G to be reduced). Then $uw_1v, uw_2v \in L$. We now show that $|\mathrm{begin}(h(w_1))| = |\mathrm{begin}(h(w_2))|$. Let $\mathrm{type}(L) = (m, n)$. If $u = \lambda$, then $w_iv \in L$ and $|\mathrm{begin}(h(w_i))| = |\mathrm{begin}(h(w_iv))| = m$. Note that if h is defined for a string w, then it is also defined for every non-empty substring of w. Now let $u \neq \lambda$. Since h is defined on uw_iv, $|\mathrm{begin}(h(w_i))| = |\mathrm{end}(h(u))|$. In the same way it can be shown that $|\mathrm{end}(h(w_1))| = |\mathrm{end}(h(w_2))|$.

We turn N into a typed alphabet by defining $\mathrm{type}(X) = \mathrm{type}(h(w))$ if $X \Rightarrow^* w$ in G. We now have to show that for every production $X \to \alpha_1 \cdots \alpha_k$ of G, $\psi(\alpha_1) \circ \cdots \circ \psi(\alpha_k)$ is a well-formed expression over CS and N, of the same type as X. Note first that for every $\alpha \in N \cup T$ and every $w \in T^*$, if $\alpha \Rightarrow^* w$, then $h(w)$ is defined and $\mathrm{type}(\psi(\alpha)) = \mathrm{type}(h(w))$. Now consider $\alpha_1, \ldots, \alpha_k$ in $N \cup T$, and let $\alpha_i \Rightarrow^* w_i \in T^*$, for $1 \leq i \leq k$. Then $\psi(\alpha_1) \circ \cdots \circ \psi(\alpha_k)$ is a well-formed expression (over CS and N) of type (m, n) if and only if $h(w_1 \cdots w_k)$ is defined and $\mathrm{type}(h(w_1 \cdots w_k)) = (m, n)$. Consider the derivation $S \Rightarrow^* uXv \Rightarrow u\alpha_1 \cdots \alpha_kv \Rightarrow^* uw_1 \cdots w_kv = z$. Since $z \in L$, $h(z)$ is defined, and so $h(w_1 \cdots w_k)$

is defined. Moreover, since $X \Rightarrow^* w_1 \cdots w_k$, $\mathrm{type}(h(w_1 \cdots w_k)) = \mathrm{type}(X)$. This shows that $\psi(\alpha_1) \circ \cdots \circ \psi(\alpha_k)$ is a well-formed expression over CS and N, of the same type as X, as required. □

Theorem 22. Int(CF) ⊂ HR.

Proof. Inclusion follows immediately from Theorems 21 and 11. Proper inclusion is a consequence of Corollary 17: the set of all trees is in HR, but is not of bounded pathwidth, as can easily be seen (for a characterization of the trees of pathwidth k, see [EllST]). □

Since REG is closed under intersection, the proof of Theorem 21 also works for Int(REG). In fact, the proof preserves the right-linearity of the grammars. Hence, Int(REG) = Val(RLIN-CFG(CS)). As an example, the graph language of clothes lines is generated by the right-linear context-free graph grammar with productions $S \to c_{h(a)} \circ X$, $X \to c_{h(b)} \circ X$, $X \to c_{h(c)} \circ X$, and $X \to c_{h(d)}$, where $h(a), h(b), h(c), h(d)$ are the graphs in Fig. 4, $\mathrm{type}(S) = (0, 0)$, and $\mathrm{type}(X) = (1, 0)$.

Together with Theorem 13, this shows that the graph languages that are interpretations of a regular language are precisely those that can be generated by linear HR grammars.

Theorem 23. Int(REG) = LIN-HR.

Similarly, the proof of Theorem 21 preserves linearity of the grammars (and LIN is closed under intersection with regular languages). Since Val(LIN-CFG($\mathrm{CS}_\circ$)) is inbetween Val(RLIN-CFG(CS)) and Val(LIN-CFG(CS)), we obtain the next result by Corollary 12 and Theorem 13.

Theorem 24. Int(LIN) = LIN-HR.

The results of this section suggest that for graph languages, regularity and linearity are the same. We have a class of graph languages that may be called the class of regular graph languages on the one hand (because it is equal to Int(REG) and to REX($\cup, \circ, *, \mathrm{SING}$)), and may be called the class of linear graph languages on the other hand (because it is equal to Int(LIN) and to LIN-HR).

It will be shown in Section 7 that Int(REG) is a proper subclass of Int(CF).

6 Characterizations of Int(K)

In this section and the next, we investigate properties of the class of graph languages Int(K) for arbitrary classes of string languages K. However, to avoid trivialities we will mainly be interested in classes K that are *closed under sequential machine mappings*, where a sequential machine is a transducer which works like an ordinary nondeterministic finite automaton that, moreover, at each step outputs one symbol (thus it is a special case of the generalized sequential machine, or *gsm*, which outputs a string at each step, see, e.g., [HopUll]).

Equivalently, K is closed under intersection with regular languages and under *alphabetical substitutions* (where an alphabetical substitution from alphabet A to alphabet B is a relation $\rho \subseteq A \times B$ that is extended to a function from A^* to the finite subsets of B^* by $\rho(a_1 \cdots a_n) = \{b_1 \cdots b_n \mid (a_i, b_i) \in \rho, 1 \leq i \leq n\}$).

In this section we present two characterizations of $\mathrm{Int}(K)$. The first characterization of $\mathrm{Int}(K)$ is through the notion of replacement, as defined in Section 2.2. The following closure property of $\mathrm{Int}(K)$ will be useful.

Lemma 25. *For every class K, $\mathrm{Int}(K)$ is closed under replacements.*

Proof. Clearly, $\phi(h(L)) = h'(L)$, where h' is defined by $h'(a) = \phi(h(a))$ for every $a \in A$. In fact, for $a_1, \ldots, a_n \in A$, $\phi(h(a_1 \cdots a_n)) = \phi(h(a_1) \circ \cdots \circ h(a_n)) = \phi(h(a_1)) \circ \cdots \circ \phi(h(a_n)) = h'(a_1) \circ \cdots \circ h'(a_n) = h'(a_1 \cdots a_n)$, where we have used Lemma 4(1). □

The characterization of $\mathrm{Int}(K)$ is based on the fact that every interpretation can be decomposed into an "atomic" interpretation and a replacement. An interpretation $h : A \to \mathrm{GR}$ of A is *atomic* if A is a typed alphabet and $h(a) = \mathrm{atom}(a)$ for every $a \in A$. By $\mathrm{AtInt}(K)$ we denote the set of all $h(L) \in \mathrm{Int}(K)$ such that h is an atomic interpretation. Recall that Repl denotes the class of all replacements.

Theorem 26. *For every class K, $\mathrm{Int}(K) = \mathrm{Repl}(\mathrm{AtInt}(K))$.*

Proof. One inclusion follows from Lemma 25. For the other inclusion, let $h : A \to \mathrm{GR}$ be an interpretation. Turn A into a typed alphabet by defining $\mathrm{type}(a) = \mathrm{type}(h(a))$ for every $a \in A$. Let t be the unique atomic interpretation of the typed alphabet A, and let $\phi : A \to \mathrm{GR}$ be the replacement defined by $\phi(a) = h(a)$ for every $a \in A$ (i.e., ϕ is h viewed as a replacement). Clearly, for every string $w \in A^*$, $\phi(t(w)) = h(w)$. In fact, if $w = a_1 \cdots a_n$, then $\phi(t(w)) = \phi(t(a_1) \circ \cdots \circ t(a_n)) = \phi(t(a_1)) \circ \cdots \circ \phi(t(a_n)) = \phi(a_1) \circ \cdots \circ \phi(a_n) = h(a_1) \circ \cdots \circ h(a_n) = h(w)$, by Lemma 4(1) and because $\phi(\mathrm{atom}(a)) = \phi(a)$ for every $a \in A$. □

Note that Lemma 25 and Theorem 26 together show that $\mathrm{Int}(K)$ is the smallest class of graph languages containing $\mathrm{AtInt}(K)$ and closed under replacements.

The second characterization of $\mathrm{Int}(K)$ generalizes the characterization of Int(REG) in Theorem 23. To this aim we consider *controlled* linear HR grammars, in the obvious sense. Let $G = (N, T, P, S)$ be an HR grammar, and let C be a string language over P (where P is viewed as an alphabet). The graph language generated by G under control C is the set of all graphs $g \in \mathrm{GR}(T)$ for which there is a derivation $\mathrm{atom}(S) \Rightarrow_{p_1} g_1 \Rightarrow_{p_2} g_2 \cdots \Rightarrow_{p_n} g_n$ with $g_n = g$, such that the string $p_1 p_2 \cdots p_n$ is in C. Recall that $\Rightarrow_p$ denotes a derivation step of G that uses production p. Thus, the control language C specifies the sequences of productions that the grammar G is allowed to use in its derivations. If C is taken from a class of languages K, the grammar G together with the control language C is also called a *K-controlled HR grammar*.

For a class K of string languages, we denote by $\text{LIN-HR}(K)$ the class of graph languages generated by K-controlled linear HR grammars. Generalizing Theorem 23 and its proof we obtain the next result.

Theorem 27. *For every class K that is closed under sequential machine mappings,* $\mathrm{Int}(K) = \text{LIN-HR}(K)$.

Proof. In what follows we assume, without loss of generality, that all languages in K are λ-free (if $K' = \{L - \{\lambda\} \mid L \in K\}$, then $\mathrm{Int}(K') = \mathrm{Int}(K)$, $\text{LIN-HR}(K') = \text{LIN-HR}(K)$, $K' = \{L \in K \mid \lambda \notin L\}$ because a sequential machine mapping can be used to remove λ, and K' is closed under sequential machine mappings, because K is closed under sequential machine mappings and sequential machine mappings are length-preserving).

With analogous definitions as above, we can define the controlled versions of context-free grammars and of context-free graph grammars over CS. We first show that K is the class of languages generated by the K-controlled right-linear context-free grammars. In one direction, let $L \in K$ with $L \subseteq T^*$. Define the right-linear context-free grammar $G = (\{S\}, T, P, S)$, where P consists of all productions $p_a : S \to aS$ and $p'_a : S \to a$ for all $a \in T$. Let L' be the control language that is obtained from L by a sequential machine mapping that changes each string $a_1 \cdots a_{n-1}a_n$ into the string $p_{a_1} \cdots p_{a_{n-1}} p'_{a_n}$. Then L is the language generated by G under control L'. In the other direction, let $G = (N, T, P, S)$ be a right-linear context-free grammar, let $C \in K$ be a control language, and let L be the language generated by G under control C. Let ϕ be the sequential machine mapping that, for a given string $p_1 \cdots p_n \in P^*$, checks whether there is a derivation $S \Rightarrow_{p_1} w_1 \cdots \Rightarrow_{p_n} w_n$ with $w_n \in T^*$ (by simulating G in its finite control) and changes each production into the unique terminal symbol that occurs in its right-hand side. Then, clearly, $\phi(C) = L$ and hence L is in K.

It is now easy to generalize the proof of Theorem 21 to the case of K-controlled right-linear grammars. This shows that $\mathrm{Int}(K)$ is equal to the K-controlled version of Val(RLIN-CFG(CS)), i.e., to the class of graph languages generated by K-controlled right-linear context-free graph grammars over CS. Thus, it now suffices to check that the proof of Theorem 13 can be generalized to the K-controlled case.

First we check that the proof of Theorem 11 can be generalized to the K-controlled case, for linear grammars. We have proved a relationship between the derivations $X \Rightarrow^* e$ of a context-free graph grammar G and the derivations $\mathrm{atom}(X) \Rightarrow^* \mathrm{val}(e)$ of an HR grammar G'. This proof can be extended to show that if the sequence of productions used in the first derivation is $p_1 p_2 \cdots p_n$, then the sequence of productions used in the second derivation is $p'_1 p'_2 \cdots p'_n$, where, for each production $p = X \to t$ of G, p' is the production $X \to \mathrm{val}(t)$ of G'; to see this, note that in the linear case of Proposition 2, the sequence of productions used in $h \Rightarrow^* g$ equals the sequence of productions used in $\mathrm{atom}(Y) \Rightarrow^* \phi(Y)$, where Y is the unique nonterminal in $\mathrm{lab}(h)$. Thus, if C is the control language of G, then the control language for G' can be obtained from C by a sequential machine mapping that changes every p into p'.

We now consider the proof of Theorem 13. Clearly, for the LIN-HR grammar $\overline{G}$ we can take the same control language that is used by G (modulo a renaming), because there is a clear one-to-one correspondence between their productions. In the remaining two constructions, we can take the same control language in the

first case, and in the second case we can apply a sequential machine mapping to the control language that changes the first production of a production sequence into the corresponding production for the new initial nonterminal S' (which was the reason to use that particular construction). □

7 Comparison of Int(K) and Int(K')

From Theorems 23 and 24 we know that Int(REG) = Int(LIN) = LIN-HR. It can be shown by a direct construction (see [Ver]) that even Int(DB) = LIN-HR, where DB is the class of derivation bounded context-free languages (see, e.g., Section VI.10 of [Sal], where they are called languages of "finite index"). One now wonders when Int(K) = Int(K'), and in particular one wonders how much larger the class K can be made without enlarging the class Int(K).

It is easy to see that for every given class K there is a largest class K' such that Int(K') = Int(K). We will call this the *extension of* K, denoted Ext(K). In fact, Ext(K) = $\bigcup\{K' \mid \mathrm{Int}(K') = \mathrm{Int}(K)\}$. Note that, for arbitrary classes K and K', Int(K) = Int(K') if and only if Ext(K) = Ext(K').

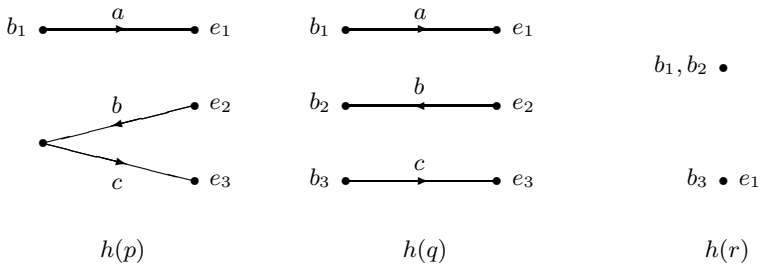

Fig. 6. Interpretation h.

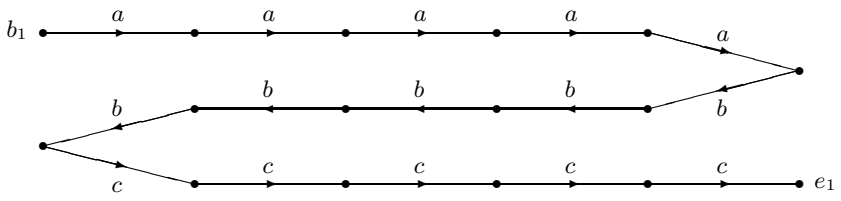

Fig. 7. The graph gr($a^5b^5c^5$) = $h(pq^4r)$.

In the next theorem we give a characterization of $\mathrm{Ext}(K)$. For a class G of graph languages, let $\mathrm{Str}(G)$ denote the class of string languages L such that $\mathrm{gr}(L)$ is in G. Here, $\mathrm{gr}(L) = \{\mathrm{gr}(w) \mid w \in L\}$, and, for a string $w = a_1 \cdots a_n$ with $n \geq 1$, $\mathrm{gr}(w) = (V, E, s, t, l, \mathrm{begin}, \mathrm{end})$ is the (ordinary) graph of type $(1,1)$ with $V = \{x_1, \ldots, x_{n+1}\}$, $E = \{e_1, \ldots, e_n\}$, $s(e_i) = x_i$, $t(e_i) = x_{i+1}$, and $l(e_i) = a_i$ for every $1 \leq i \leq n$, $\mathrm{begin} = x_1$ and $\mathrm{end} = x_{n+1}$. Thus, $\mathrm{gr}(w)$ encodes w in the obvious way: it is a path with the symbols of w as edge labels.

As a classical example, the language $L = \{a^n b^n c^n \mid n \geq 1\}$ is in $\mathrm{Str}(\mathrm{Int}(\mathrm{REG}))$, because $L = h(M)$ with $M = pq^*r$ and h is shown in Fig. 6. The graph $\mathrm{gr}(a^5b^5c^5) = h(pq^4r)$ is shown in Fig. 7.

Note that $\mathrm{gr}(\lambda)$ is not defined. This implies that $L \in \mathrm{Str}(G)$ iff $L - \{\lambda\} \in \mathrm{Str}(G)$. It is also implies that 'gr' is the unique atomic interpretation which is obtained by viewing every symbol as having type $(1,1)$; hence $\mathrm{gr}(L) \in \mathrm{Int}(K)$ for every $L \in K$, which means that $K \subseteq \mathrm{Str}(\mathrm{Int}(K))$.

Theorem 28. *For every class K that is closed under sequential machine mappings,* $\mathrm{Ext}(K) = \mathrm{Str}(\mathrm{Int}(K))$.

Proof. Clearly, if $\mathrm{Int}(K') = \mathrm{Int}(K)$, then $K' \subseteq \mathrm{Str}(\mathrm{Int}(K')) = \mathrm{Str}(\mathrm{Int}(K))$. Thus, it remains to show that $\mathrm{Int}(\mathrm{Str}(\mathrm{Int}(K))) = \mathrm{Int}(K)$. Since $K \subseteq \mathrm{Str}(\mathrm{Int}(K))$, $\mathrm{Int}(K) \subseteq \mathrm{Int}(\mathrm{Str}(\mathrm{Int}(K)))$. For the other inclusion it suffices, by Lemma 25 and Theorem 26, to show that $\mathrm{AtInt}(\mathrm{Str}(\mathrm{Int}(K))) \subseteq \mathrm{Int}(K)$. To prove this, let $L_1 \in K$, let h_1 be an interpretation of the alphabet A of L_1 such that $h_1(L_1) = \mathrm{gr}(L_2)$ for some λ-free string language L_2, and let h_2 be an atomic interpretation of the alphabet B of L_2. Thus, $h_1 : A \to \mathrm{GR}(B)$ where each symbol from B has type $(1,1)$. However, for the atomic interpretation h_2 each symbol b from B has another (arbitrary) type that we will denote by $\mathrm{type}(b)$. Note that $h_2(b) = \mathrm{atom}(b)$, where $\mathrm{type}(\mathrm{atom}(b)) = \mathrm{type}(b)$; hence $\mathrm{type}(b) = (|\mathrm{begin}(h_2(b))|, |\mathrm{end}(h_2(b))|)$.

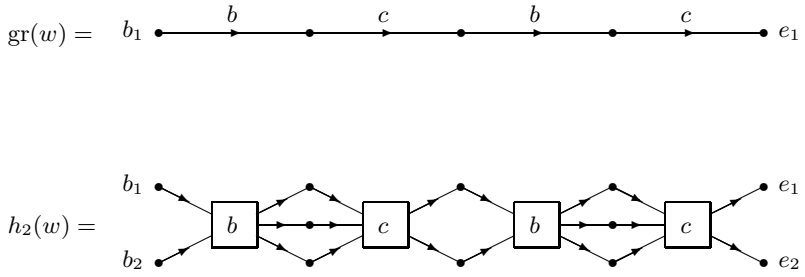

Fig. 8. Graphs $\mathrm{gr}(w)$ and $h_2(w)$ for $w = bcbc$, with $\mathrm{type}(b) = (2,3)$ and $\mathrm{type}(c) = (3,2)$.

We have to construct a language $L \in K$ and an interpretation h such that $h(L) = h_2(L_2)$. For a string $w \in L_2$ such that $h_2(w)$ is defined, the graph $h_2(w)$

can be obtained from the graph $\mathrm{gr}(w)$ (which is an element of $h_1(L_1)$) in an easy way, as follows (see Fig. 8). Each node v of $\mathrm{gr}(w)$ has to be "expanded" into a sequence of distinct nodes $(v,1),(v,2),\ldots,(v,\mu(v))$, where μ stands for "multiplicity". Clearly, $\mu(v)$ is determined by $\mathrm{type}(b)$, where b is the label of an edge e incident with v: if e enters v, then $\mu(v) = |\mathrm{end}(h_2(b))|$, and if e leaves v, then $\mu(v) = |\mathrm{begin}(h_2(b))|$. Every edge e of $gr(w)$, with source u and target v, should be replaced by an edge e with sources $(u,1),\ldots,(u,\mu(u))$ and targets $(v,1),\ldots,(v,\mu(v))$ (and the same label). In Fig. 8, the multiplicity of the nodes of $\mathrm{gr}(w)$ is 2, 3, 2, 3, 2, respectively. The edges of $h_2(w)$ are drawn as squares, with "tentacles" from their sources and to their targets (where we asume that the tentacles are ordered, e.g., from top to bottom).

We now define this expansion process formally, for arbitrary graphs in $\mathrm{GR}(B)$. Let M be a number such that for all $b \in B$, if $\mathrm{type}(b) = (m,n)$, then $m,n \leq M$. A *decoration* of a graph $g \in \mathrm{GR}(B)$ is a mapping $\mu : V_g \to \{1,\ldots,M\}$ such that for every edge e of g with $s(e) = u$, $t(e) = v$, and $l(e) = b$: $\mathrm{type}(b) = (\mu(u),\mu(v))$, i.e., $\mu(u) = |\mathrm{begin}(h_2(b))|$ and $\mu(v) = |\mathrm{end}(h_2(b))|$. Note that for every graph $\mathrm{gr}(w)$, $h_2(w)$ is defined if and only if there is a decoration μ of $\mathrm{gr}(w)$, and in that case μ is unique. For a graph $g \in \mathrm{GR}(B)$ and a decoration μ of g, the *expansion* of g by μ, denoted $\exp(g,\mu)$, is the graph that has all nodes (v,i) where v is a node of g and $1 \leq i \leq \mu(v)$; it has the same edges as g (with the same labels), but if $s(e) = u$ and $t(e) = v$ in g, then $s(e) = (u,1)\cdots(u,\mu(u))$ and $t(e) = (v,1)\cdots(v,\mu(v))$ in $\exp(g,\mu)$; finally, if $\mathrm{begin}(g) = v_1v_2\cdots v_k$, then $\mathrm{begin}(\exp(g,\mu)) =$

$$(v_1,1)\cdots(v_1,\mu(v_1))\cdots(v_2,1)\cdots(v_2,\mu(v_2))\cdots(v_k,1)\cdots(v_k,\mu(v_k)),$$

and similarly for $\mathrm{end}(g)$ and $\mathrm{end}(\exp(g,\mu))$. It should be clear that for every $w \in L_2$ for which $h_2(w)$ is defined, $h_2(w) = \exp(\mathrm{gr}(w),\mu)$ where μ is the unique decoration mentioned above.

Based on this idea of expansion we now change L_1 into L and h_1 into h, as follows. Let A' be the alphabet consisting of all pairs (a,μ) with $a \in A$ and μ is a decoration of $h_1(a)$. Intuitively, μ is a guess of the multiplicities of the nodes of $h_1(a)$ as they will occur in a graph of $h_1(L_1)$; for nodes that are incident with an edge, this multiplicity is determined by the label of that edge, but for the other nodes (which are necessarily begin or end nodes because $\mathrm{gr}(w)$ has no isolated nodes) their multiplicity will only be clear after concatenation. Let ρ be the alphabetical substitution that substitutes all possible (a,μ) for a, i.e., $\rho = \{(a,(a,\mu)) \mid (a,\mu) \in A'\}$. Thus, $\rho(L_1) = \{(a_1,\mu_1)\cdots(a_n,\mu_n) \mid a_1\cdots a_n \in L_1, (a_j,\mu_j) \in A'\}$. Let R be the regular language over A' that consists of all strings $(a_1,\mu_1)\cdots(a_n,\mu_n)$ such that $h_1(a_1\cdots a_n)$ is defined and $\mu_j(\mathrm{end}(h_1(a_j))(i)) = \mu_{j+1}(\mathrm{begin}(h_1(a_{j+1}))(i))$ for all relevant j and i (to be precise: for all $1 \leq j < n$ and all $1 \leq i \leq |\mathrm{end}(h_1(a_j))|$; note that $|\mathrm{end}(h_1(a_j))| = |\mathrm{begin}(h_1(a_{j+1}))|$ because $h_1(a_1\cdots a_n)$ is defined). In words, the language R checks that the guessed multiplicity of the ith end node of $h_1(a_j)$ equals that of the ith begin node of $h_1(a_{j+1})$. Thus, R checks that the guessed multiplicities are consistent with the identification of nodes when concatenating the graphs

$h_1(a_1), \ldots, h_1(a_n)$ (and hence are the correct multiplicities). It should be clear that R is indeed regular (cf. Lemma 15). We now define $L = \rho(L_1) \cap R$; since, by assumption, K is closed under sequential machine mappings, L is in K. Finally, for $(a, \mu) \in A'$ we define $h(a, \mu) = \exp(h_1(a), \mu)$.

It should now be clear that $h(L) = h_2(L_2)$. A formal proof can be given as follows.

Let a *decorated graph* be a pair (g, μ) with $g \in \mathrm{GR}(B)$ and μ a decoration of g. Define the concatenation of decorated graphs, as follows. For decorated graphs (g_1, μ_1) and (g_2, μ_2), if $g_1 \circ g_2$ is defined and $\mu_1(\mathrm{end}(g_1)(i)) = \mu_2(\mathrm{begin}(g_2)(i))$ for all i, then their concatenation is $(g_1, \mu_1) \circ (g_2, \mu_2) = (g_1 \circ g_2, \mu)$ with $\mu([x]_R) = \mu_j(x)$ if $x \in V_{g_j}$ (where we assume the terminology of Definition 3). It is easy to see that, the other way around, if $g_1 \circ g_2$ is defined and $(g_1 \circ g_2, \mu)$ is a decorated graph, then there exist decorations μ_1 and μ_2 of g_1 and g_2, respectively, such that $(g_1, \mu_1) \circ (g_2, \mu_2) = (g_1 \circ g_2, \mu)$. As a basic property of 'exp' it can be shown that it is a homomorphism with respect to the concatenation of decorated graphs: $\exp((g_1, \mu_1) \circ (g_2, \mu_2)) = \exp(g_1, \mu_1) \circ \exp(g_2, \mu_2)$.

Now consider some string $a_1 \cdots a_n \in L_1$. Then $h_2(\mathrm{gr}^{-1}(h_1(a_1 \cdots a_n))$ is defined if and only if there exist decorations μ_i such that $(h_1(a_i), \mu_i)$ is a decorated graph, for all i, and their concatenation $(h_1(a_1), \mu_1) \circ \cdots \circ (h_1(a_n), \mu_n)$ is defined. And this is if and only if there exist μ_i such that $(a_i, \mu_i) \in A'$ and $(a_1, \mu_1) \cdots (a_n, \mu_n) \in R$. Moreover, in that case, for the unique decoration μ of $h_1(a_1 \cdots a_n)$,

$$
\begin{aligned}
&h_2(\mathrm{gr}^{-1}(h_1(a_1 \cdots a_n)) = \exp(h_1(a_1 \cdots a_n), \mu) = \\
&\exp((h_1(a_1), \mu_1) \circ \cdots \circ (h_1(a_n), \mu_n)) = \\
&\exp(h_1(a_1), \mu_1) \circ \cdots \circ \exp(h_1(a_n), \mu_n) = \\
&h(a_1, \mu_1) \circ \cdots \circ h(a_n, \mu_n) = h((a_1, \mu_1) \cdots (a_n, \mu_n)).
\end{aligned}
$$

This shows that $h_2(L_2) = h(L)$. □

As a corollary of Theorem 28 we obtain that for arbitrary K and K' (both closed under sequential machine mappings), $\mathrm{Int}(K) = \mathrm{Int}(K')$ if and only if $\mathrm{Str}(\mathrm{Int}(K)) = \mathrm{Str}(\mathrm{Int}(K'))$. This means that the graph generating power of K is completely determined by its string generating power (with strings coded as graphs by the mapping gr).

We now show that $\mathrm{Ext}(K)$ is a class of languages that is well known in formal language theory. By $\mathrm{2DGSM}(K)$ we denote the class of images of languages from K under *2dgsm mappings*, i.e., the class of all $f(L)$ where f is a 2dgsm mapping and $L \in K$. A *2dgsm* (i.e., a two-way deterministic generalized sequential machine) is a deterministic finite automaton that can move in two directions on its input tape (with endmarkers), and outputs a (possibly empty) string at each step. As an example, $\{a^n b^n c^n \mid n \geq 1\}$ is in $\mathrm{2DGSM}(\mathrm{REG})$, because it is easy to construct a 2dgsm that translates $pq^n r$ into $a^{n+1} b^{n+1} c^{n+1}$ for every $n \in \mathbf{N}$. The proof of the next result is obtained by generalizing the proof in [EngHey] that $\mathrm{Str}(\mathrm{LIN\text{-}HR})$ equals the class of output languages of 2dgsm mappings. We say

that K is *nontrivial* if it is not a subset of $\{\emptyset, \{\lambda\}\}$; in other words, K contains at least one language that contains a nonempty string.

Lemma 29. *For every nontrivial class K that is closed under sequential machine mappings,* $\mathrm{Str}(\text{LIN-HR}(K)) = \text{2DGSM}(K)$.

Proof. To reduce the proof to a generalization of the proof in [EngHey], we have to deal with some technical details. In particular, the coding 'gr' (and hence the operation 'Str') is defined in a different way (see Definition 2.2 of [EngHey]). We will discuss this in steps. Note that, by Theorem 27, $\mathrm{Str}(\text{LIN-HR}(K)) = \mathrm{Str}(\mathrm{Int}(K))$.

First of all, let gr_1 be defined in the same way as gr, except that additionally $\mathrm{gr}_1(\lambda) = \mathrm{id}_1$; and let Str_1 be defined in the same way as Str, with gr_1 instead of gr. We claim that $\mathrm{Str}_1(\mathrm{Int}(K)) = \mathrm{Str}(\mathrm{Int}(K))$. Thus, we have to show that for every language L, $\mathrm{gr}_1(L) \in \mathrm{Int}(K)$ iff $\mathrm{gr}(L) \in \mathrm{Int}(K)$. This is obvious if $\lambda \notin L$. If $\lambda \in L$, then $\mathrm{gr}_1(L) = \mathrm{gr}(L) \cup \{\mathrm{id}_1\}$. Assume first that $\mathrm{gr}(L)$ is in $\mathrm{Int}(K)$. If $\mathrm{gr}(L) = \emptyset$, we have to show that $\{\mathrm{id}_1\} \in \mathrm{Int}(K)$. Since K is not a subset of $\{\emptyset, \{\lambda\}\}$, K contains a language $M \subseteq A^*$ such that M contains at least one nonempty string. Define the interpretation h with $h(a) = \mathrm{id}_1$ for all $a \in A$. Then $h(M) = \{\mathrm{id}_1\}$ (because $\mathrm{id}_1 \circ \mathrm{id}_1 = \mathrm{id}_1$). Now let $\mathrm{gr}(L) \neq \emptyset$. Let $\mathrm{gr}(L) = h(M)$ for some interpretation h and some $M \in K$. Then M must contain a nonempty string w. Let b be a new symbol, not in the alphabet A of M, and define the interpretation h' of $A \cup \{b\}$ such that $h'(a) = h(a)$ for every $a \in A$ and $h(b) = \mathrm{id}_1$. Then $h'(M \cup \{b^{|w|}\}) = h(M) \cup \{\mathrm{id}_1\} = \mathrm{gr}_1(L)$. It is easy to see that there is a sequential machine mapping that transforms M into $M \cup \{b^{|w|}\}$. This proves one direction of the equivalence. To show the other direction, assume that $\mathrm{gr}_1(L) \in \mathrm{Int}(K)$. If $\mathrm{id}_1 \in \mathrm{gr}(L)$, then there is nothing to prove. Now let $\mathrm{id}_1 \notin \mathrm{gr}(L)$. Then $\mathrm{gr}(L) = \mathrm{gr}_1(L) - \{\mathrm{id}_1\}$. Let $\mathrm{gr}(L) = h(M)$ for some interpretation h and some $M \in K$, $M \subseteq A^*$. Let B be the set of all $a \in A$ such that $h(a)$ has at least one edge. Then the regular language A^*BA^* is the set of all $w \in A^*$ such that $h(w)$ contains at least one edge. Hence $\mathrm{gr}(L) = h(M \cap A^*BA^*) \in \mathrm{Int}(K)$.

Second, let $\mathrm{gr}_2(w) = \mathrm{backfold}(\mathrm{gr}_1(w))$, and let Str_2 be defined on the basis of gr_2. Then $\mathrm{Str}_2(\mathrm{Int}(K)) = \mathrm{Str}_1(\mathrm{Int}(K))$. This is because for every graph language L, $L \in \mathrm{Int}(K)$ iff $\mathrm{backfold}(L) \in \mathrm{Int}(K)$. To see this, note that for every graph g with $\mathrm{type}(g) = (m, n)$, $\mathrm{backfold}(g) = (g \oplus \mathrm{id}_n) \circ \mathrm{backfold}(\mathrm{id}_n)$ (see the proof of Theorem 7) and $g = (\mathrm{id}_m \oplus \mathrm{fold}(\mathrm{id}_n)) \circ (\mathrm{backfold}(g) \oplus \mathrm{id}_n)$ (see the proof of Theorem 13). Thus, it suffices to show that $\mathrm{Int}(K)$ is closed under the operations $L' \oplus \{\mathrm{id}_n\}$, $\{h\} \circ L'$, and $L' \circ \{h\}$. For the first operation this has been shown in the proof of Lemma 19. The other two operations are left to the reader (see the end of the proof of Theorem 13 and the end of the proof of Theorem 27).

Third, define $\mathrm{gr}_3(w)$ in the same way as $\mathrm{gr}_2(w)$, except that the type of the edges is changed from $(1, 1)$ to $(2, 0)$. To be precise, an edge e with $s(e) = u$ and $t(e) = v$ in $\mathrm{gr}_2(w)$, has $s(e) = uv$ and $t(e) = \lambda$ in $\mathrm{gr}_3(w)$. It should be clear that $\mathrm{Str}_3(\mathrm{Int}(K)) = \mathrm{Str}_2(\mathrm{Int}(K))$, where Str_3 is based on gr_3 in the usual way.

In [EngHey], the coding gr_3 is used instead of gr. We have just shown that this does not change the class $\mathrm{Str}(\text{LIN-HR}(K))$, in the sense that $\mathrm{Str}_3(\text{LIN-HR}(K)) =$

Str(LIN-HR(K)). Another small difference is that in [EngHey] all (terminal and nonterminal) edges of graphs have type $(m, 0)$ for some m. Let us indicate this here by HR$'$. Since all edges of $\mathrm{gr}_3(w)$ have type $(2, 0)$, and since backfold$(\mathrm{gr}_3(w)) = \mathrm{gr}_3(w)$, it should be clear from the construction of $\overline{G}$ in the proof of Theorem 13 that for every language L, $\mathrm{gr}_3(L) \in$ LIN-HR$'(K)$ iff $\mathrm{gr}_3(L) \in$ LIN-HR(K). This shows that Str(LIN-HR(K)) = Str$_3$(LIN-HR$'(K)$), the class considered in [EngHey].

It is proved in [EngHey] that Str(LIN-HR) equals the class of ranges of 2dgsm mappings. Since it is easy to see that LIN-HR(REG) = LIN-HR and that 2DGSM(REG) is the class of output languages of 2dgsm's (by incorporating the regular control language in the finite control of the grammar and the 2dgsm, respectively), this proves the theorem for $K =$ REG. The proof of the general case consists of a careful analysis of the proof in [EngHey], which shows that it can be generalized to K-controlled grammars, under the assumption that K is closed under sequential machine mappings (and K is nontrivial). This analysis can easily be carried out. □

From Theorems 27, 28, and Lemma 29, we obtain our second characterization of the class Ext(K).

Theorem 30. *For every nontrivial class K that is closed under sequential machine mappings,* Ext(K) = 2DGSM(K).

Corollary 31. *For all nontrivial classes K and K' that are closed under sequential machine mappings,*
Int(K) = Int(K') *if and only if* 2DGSM(K) = 2DGSM(K').

Quite a lot is known about the class 2DGSM(K), see, e.g., [EngRS]. As an example, it equals the class of languages generated by K-controlled ETOL systems of finite index (Corollary 4.10 of [EngRS]). The trivial fact that Ext(Ext(K)) = Ext(K) corresponds to the known result that 2DGSM(2DGSM(K)) is equal to 2DGSM(K); this shows that Ext(K) is closed under 2dgsm mappings (cf. Corollary 5.8 of [EngRS]).

Theorem 30 and Corollary 31 allow us to use known formal language theoretic results for the classes 2DGSM(K) to find out the power of the classes Int(K). Thus, for $K =$ REG, Ext(K) is the class 2DGSM(REG) of output languages of 2dgsm mappings. Since it is well known that the class DB of derivation-bounded context-free languages is contained in 2DGSM(REG) (see, e.g., [Raj]), this implies the previously mentioned result that Int(DB) = Int(REG). Also, since there is a context-free language not in 2DGSM(REG), see Lemma 4.24 of [Gre] (or Theorem 3.2.17 of [EngRS]), Int(REG) is properly included in Int(CF).

Theorem 32. Int(REG) = Int(LIN) = Int(DB) $\subset$ Int(CF) $\subset$ HR.

Finally, we would like to know whether Int(CF) is the largest class Int(K) that is included in HR. This is true if and only if Ext(CF) is the largest class K such that Int(K) is included in HR. Trying to find an answer to this question, we first characterize this largest class.

Theorem 33. Str(HR) *is the largest class* K *such that* $\mathrm{Int}(K) \subseteq \mathrm{HR}$.

Proof. The proof is similar to the one of Theorem 28. We first observe that HR is closed under replacements. In fact, if G is a context-free graph grammar over CS (see Theorem 11) and ϕ is a replacement, then $\phi(\mathrm{val}(L(G))) = \mathrm{val}(L(G'))$, where G' is obtained from G by changing every constant c_g that occurs in the productions of G into $c_{\phi(g)}$. The correctness of this construction follows from Lemma 4.

As in the proof of Theorem 28 it now suffices to show that $\mathrm{AtInt}(\mathrm{Str}(\mathrm{HR})) \subseteq \mathrm{HR}$. Instead of HR we consider the class $\mathrm{Val}(\mathrm{CFG}(\mathrm{CS}))$, see Theorem 11. Let $G = (N, T, P, S)$ be a context-free graph grammar over CS such that $\mathrm{val}(L(G)) = \mathrm{gr}(L)$ for some language $L \subseteq B^*$, and let h be an atomic interpretation of B such that $h(L)$ is a graph language. Note that, as in the proof of Theorem 28, each symbol $b \in B$ has type $(1,1)$ in $\mathrm{val}(L(G))$, and has an arbitrary type with respect to h. We may assume that G is in *normal form*, i.e., that all its productions are of the form $X \to c_g$ or $X \to Y \circ Z$ or $X \to Y \oplus Z$, where $X, Y, Z \in N$ and $g \in \mathrm{GR}$ (this is the usual normal form of regular tree grammars, see, e.g., [GécSte]).

We have to construct a context-free graph grammar $G' = (N', T', P', S')$ such that $\mathrm{val}(L(G')) = h(L)$. The idea of the construction is the same as in the proof of Theorem 28: each graph $\mathrm{gr}(w)$ for which $h(w)$ is defined, is transformed into $\exp(\mathrm{gr}(w), \mu)$, where μ is the unique decoration of $\mathrm{gr}(w)$ (see the proof of Theorem 28 for the terminology used). Let M be the maximal number occurring in the types of the symbols of B (with respect to h). We define N' to consist of all triples (X, μ_b, μ_e) such that $X \in N$ and $\mu_b, \mu_e \in \{1, \ldots, M\}^*$ with $\mathrm{type}(X) = (|\mu_b|, |\mu_e|)$; moreover, $\mathrm{type}(X, \mu_b, \mu_e) = \mathrm{type}(X)$. The intuition is that if $\mathrm{atom}(X) \Rightarrow^* g$ where g is terminal, then $\mathrm{atom}(X, \mu_b, \mu_e) \Rightarrow^* \exp(g, \mu)$ where μ is the decoration of g such that $\mu(\mathrm{begin}(g)(i)) = \mu_b(i)$ for all $1 \leq i \leq |\mu_b|$ and $\mu(\mathrm{end}(g)(j)) = \mu_e(j)$ for all $1 \leq j \leq |\mu_e|$ (note that, assuming G to be reduced, μ is unique because all isolated nodes of g are begin or end nodes). The initial nonterminal S' of G' is $(S, \langle m \rangle, \langle n \rangle)$, where $(m, n) = \mathrm{type}(h(L))$. The productions in P' are defined as follows. If $X \to Y \circ Z$ is in P, then $(X, \mu_b, \mu_e) \to (Y, \mu_b, \mu) \circ (Z, \mu, \mu_e)$ is in P' for all appropriate strings μ_b, μ_e, and μ over $\{1, \ldots, M\}$. If $X \to Y \oplus Z$ is in P, then $(X, \mu_b \cdot \mu'_b, \mu_e \cdot \mu'_e) \to (Y, \mu_b, \mu_e) \oplus (Z, \mu'_b, \mu'_e)$ is in P' for all $(Y, \mu_b, \mu_e), (Z, \mu'_b, \mu'_e) \in N'$. Finally, if $X \to c_g$ is in P, then $(X, \mu_b, \mu_e) \to c_{\exp(g,\mu)}$ is in P' for all strings μ_b, μ_e and all decorations μ of g such that $\mu(\mathrm{begin}(g)(i)) = \mu_b(i)$ and $\mu(\mathrm{end}(g)(j)) = \mu_e(j)$ for all appropriate i and j.

This ends the construction of G'. A formal correctness proof is left to the reader. It should be based on a definition of the sum of decorated graphs, and the fact that 'exp' is a homomorphism with respect to this sum (and the corresponding fact for the concatenation of decorated graphs, as observed in the proof of Theorem 28). □

This proves that $\mathrm{Int}(\mathrm{Str}(\mathrm{HR}))$ is the largest class $\mathrm{Int}(K)$ that is included in HR.

It is shown in [EngHey] that the class Str(HR) of string languages generated by HR grammars is equal to the class OUT(DTWT) of output lan-

guages of deterministic tree-walking transducers. It now follows from Theorems 30 and 33 that Int(CF) is the largest class Int(K) that is included in HR if and only if Ext(CF) is the largest class K such that Int(K) $\subseteq$ HR if and only if 2DGSM(CF) = OUT(DTWT). Note that it follows from our results that 2DGSM(CF) = Ext(CF) = Str(Int(CF)) $\subseteq$ Str(HR) = OUT(DTWT), which was proved in a completely different way in Corollary 5.6 of [EngRS] (where 2DGSM(CF) is denoted DCS(CF), and OUT(DTWT) is denoted DCT(REC) or yT_{fc}(REC)). However, equality of 2DGSM(CF) and OUT(DTWT) is mentioned as an open problem after Corollary 5.6 of [EngRS]. Hence, it is an *open problem* whether or not Int(CF) is the largest class Int(K) that is included in HR. As *another open problem* we mention the following: is it true that every HR graph language of bounded pathwidth is in Int(Str(HR))? Or even in Int(CF)? Note that all HR graph languages in Int(Str(HR)) are of bounded pathwidth by Corollary 17. Some *more open problems* are: is it decidable whether an HR graph language is in Int(REG)? and the same question for Int(CF) and Int(Str(HR)).

We finally mention that it would be interesting to find another natural operation of concatenation of graphs that can be used to characterize the graph languages generated by the linear edNCE grammars (which are node replacement graph grammars, see, e.g., [CouER]).

Acknowledgment. We wish to thank Hans Bodlaender for the references to pathwidth.

References

[BauCou] M.Bauderon, B.Courcelle; Graph expressions and graph rewritings, Math. Syst. Theory 20 (1987), 83-127

[Ben] D.B.Benson; The basic algebraic structures in categories of derivations, Inform. and Control 28 (1975), 1-29

[Ber] J.Berstel; *Transductions and Context-Free Languages*, Teubner, Stuttgart, 1979

[Bod] H.L.Bodlaender; A partial k-arboretum of graphs with bounded treewidth, Preliminary version, Utrecht University, September 1995

[BosDW] F.Bossut, M.Dauchet, B.Warin; A Kleene theorem for a class of planar acyclic graphs, Inform. and Comput. 117 (1995), 251-265

[Cla] V.Claus; Ein Vollständigkeitssatz für Programme und Schaltkreise, Acta Informatica 1 (1971), 64-78

[Cou1] B.Courcelle; An axiomatic definition of context-free rewriting and its application to NLC graph grammars, Theor. Comput. Sci. 55 (1987), 141-181

[Cou2] B.Courcelle; Graph rewriting: an algebraic and logic approach, in *Handbook of Theoretical Computer Science, Vol.B* (J.van Leeuwen, ed.), Elsevier, 1990, pp.193-242

[Cou3] B.Courcelle; The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues, RAIRO Theoretical Informatics and Applications 26 (1992), 257-286

[CouER] B.Courcelle, J.Engelfriet, G.Rozenberg; Handle-rewriting hypergraph languages, J. of Comp. Syst. Sci. 46 (1993), 218-270

[Dre] F.Drewes; Transducibility - symbolic computation by tree-transductions, University of Bremen, Bericht Nr. 2/93, 1993

[EhrKKK] H.Ehrig, K.-D.Kiermeier, H.-J.Kreowski, W.Kühnel; *Universal Theory of Automata*, Teubner, Stuttgart, 1974

[EllST] J.A.Ellis, I.H.Sudborough, J.S.Turner; The vertex separation and search number of a graph, Inform. and Comput. 113 (1994), 50-79

[Eng] J.Engelfriet; Graph grammars and tree transducers, Proc. CAAP'94 (S.Tison, ed.), Lecture Notes in Computer Science 787, Springer-Verlag, Berlin, 1994, pp.15-36

[EngHey] J.Engelfriet, L.M.Heyker; The string generating power of context-free hypergraph grammars, J. of Comp. Syst. Sci. 43 (1991), 328-360

[EngRS] J.Engelfriet, G.Rozenberg, G.Slutzki; Tree transducers, L systems, and two-way machines, J. of Comp. Syst. Sci. 20 (1980), 150-202

[EngVer] J.Engelfriet, J.J.Vereijken; Concatenation of graphs, in *Graph-Grammars and their Application to Computer Science*, Proceedings of the 5th International Workshop, Williamsburg, 1994, to appear as Lecture Notes in Computer Science, Springer-Verlag, Berlin

[GécSte] F.Gécseg, M.Steinby; *Tree Automata*, Akadémiai Kiadó, Budapest, 1984

[Gre] S.Greibach; One-way finite visit automata, Theor. Comput. Sci. 6 (1978), 175-221

[Hab] A.Habel; *Hyperedge Replacement: Grammars and Languages*, Lecture Notes in Computer Science 643, Springer-Verlag, Berlin, 1992

[HabKre] A.Habel, H.-J.Kreowski; May we introduce to you: hyperedge replacement, in *Graph-Grammars and their Application to Computer Science* (H.Ehrig, M.Nagl, G.Rozenberg, A.Rosenfeld, eds.), Lecture Notes in Computer Science 291, Springer-Verlag, Berlin, 1987, pp.15-26

[HabKV] A.Habel, H.-J.Kreowski, W.Vogler; Metatheorems for decision problems on hyperedge replacement graph languages, Acta Informatica 26 (1989), 657-677

[HopUll] J.E.Hopcroft, J.D.Ullman; *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass., 1979

[Hot1] G.Hotz; Eine Algebraisierung des Syntheseproblems von Schaltkreisen, EIK 1 (1965), 185-205, 209-231

[Hot2] G.Hotz; Eindeutigkeit und Mehrdeutigkeit formaler Sprachen, EIK 2 (1966), 235-246

[HotKM] G.Hotz, R.Kolla, P.Molitor; On network algebras and recursive equations, in *Graph-Grammars and Their Application to Computer Science* (H.Ehrig, M.Nagl, G.Rozenberg, A.Rosenfeld, eds.), Lecture Notes in Computer Science 291, Springer-Verlag, Berlin, 1987, pp.250-261

[Klo] T.Kloks; *Treewidth*, Lecture Notes in Computer Science 842, Springer-Verlag, Berlin, 1994

[Lau] C.Lautemann; Decomposition trees: structured graph representation and efficient algorithms, Proc. CAAP'88, Lecture Notes in Computer Science 299, Springer-Verlag, Berlin, 1988, pp.28-39

[MezWri] J.Mezei, J.B.Wright; Algebraic automata and context-free sets, Inform. and Control 11 (1967), 3-29

[Raj] V.Rajlich; Absolutely parallel grammars and two-way finite state transducers, J. of Comp. Syst. Sci. 6 (1972), 324-342

[RobSey] N.Robertson, P.D.Seymour; Graph minors I. Excluding a forest, J. Comb. Theory Ser.B 35 (1983), 39-61

[Sal] A.Salomaa; *Formal Languages*, Academic Press, New York, 1973

[Ver] J.J.Vereijken; *Graph Grammars and Operations on Graphs*, Master's Thesis, Leiden University, May 1993