Concatenation of Graphs

Joost Engelfriet

Department of Computer Science, Leiden University,
P.O. Box 9512, NL-2300 RA Leiden, The Netherlands
email: engelfriet@rulwinw.leidenuniv.nl

Jan Joris Vereijken

Department of Computing Science, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
email: janjoris@acm.org

September 27, 1994

Abstract

An operation of concatenation is introduced for graphs. Then strings are viewed as expressions denoting graphs, and string languages are interpreted as graph languages. For a class K of string languages, $\mathrm{Int}(K)$ is the class of all graph languages that are interpretations of languages from K. For the class REG of regular languages, $\mathrm{Int}(\mathrm{REG})$ might be called the class of regular graph languages; it equals the class of graph languages generated by linear Hyperedge Replacement Systems. Two characterizations are given of the largest class K' such that $\mathrm{Int}(K') = \mathrm{Int}(K)$.

Context-free graph languages are generated by context-free graph grammars, which are 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., [9]). A completely different way of generating graphs (introduced in [1]) 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 a regular tree grammar). It is shown in [1] that, for a particular collection of graph operations, this new way of generating graphs is equivalent with the HR grammar. Other work on the generation of graphs through graph expressions is in, e.g., [2, 3, 4, 5, 11].

We introduce a new, natural operation on graphs (which is a simple variation of the graph operations in [1]). Due to its similarity to concatenation of strings, it is called *concatenation of graphs*. Together with the sum operation of graphs (as defined in [1]) and all constant graphs, a collection of graph operations is obtained that is simpler than the one in [1], but also has the power of the HR grammar (which is our first result).

Let us be a bit more precise. We consider the multi-pointed graphs (or multi-pointed hypergraphs) of [9]. For simplicity we will restrict ourselves in this paper to graphs, but all arguments also hold for hypergraphs. A multi-pointed graph is a directed, edge-labeled graph g with a designated sequence $\mathrm{begin}(g)$ of "begin nodes" and a designated

sequence end(g) of "end nodes". If begin(g) has length m and end(g) has length n (with $m, n \in \mathbb{N} = \{0, 1, 2, \ldots\}$), then g is said to be of *type* (m, n) and we write type(g) = (m, n). From now on we will drop the adjective "multi-pointed". 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. Our operations are defined on abstract graphs. The set of abstract graphs will be denoted GR. A (typed) *graph language* is a subset L of GR 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).

If g and h are graphs with type(g) = (k, m) and 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, begin($g \circ h$) = begin(g) and end($g \circ h$) = end(h), and so type($g \circ h$) = (k, n). Note that the concatenation of g and h is defined only when end(g) and begin(h) have the same length. The *sum* $g + h$ of arbitrary graphs g and h (as defined in [1]) is their disjoint union, with begin($g + h$) = begin(g) $\cdot$ begin(h) and end($g + h$) = end(g) $\cdot$ end(h), where $\cdot$ denotes the usual concatenation of sequences. Intuitively, concatenation is sequential composition of graphs, and sum is parallel composition of graphs.

We investigate some simple properties of these graph operations: they lead to a strict monoidal category. The objects of this category are the natural numbers, and each (abstract) graph of type (m, n) is a morphism from m to n in this category. Concatenation is the composition of morphisms. For each n, the identity morphism from n to itself is the (abstract graph corresponding to the) discrete graph id_n with nodes $1, \ldots, n$ and begin(id_n) = end(id_n) = $(1, \ldots, n)$. The fact that the category is strict monoidal means that $\mathrm{id}_{m+n} = \mathrm{id}_m + \mathrm{id}_n$ and $(g + h) \circ (g' + h') = (g \circ g') + (h \circ h')$ (assuming that $g \circ g'$ and $h \circ h'$ are defined).

Let Δ be the set of operators $\{\circ, +\} \cup \{c_g \mid g \in \mathrm{GR}\}$, where $\circ$ and $+$ are concatenation and sum of graphs, as discussed above, and c_g is a constant standing for the graph g. A *regular tree grammar over* Δ is an ordinary context-free grammar G such that the right-hand side of each production of G is a (well-typed) expression over the operators from Δ and the nonterminals of the grammar (which should be treated as constant operators, with a given type). Obviously, the language $L(G)$ generated by G is a set of expressions over Δ (and it is called a regular tree language). But G can also be viewed as a (context-free) graph grammar, generating the graph language val($L(G)$) = $\{\mathrm{val}(e) \mid e \in L(G)\}$, where the graph val($e$) is the value of the expression e. Let Val(REGT) = $\{\mathrm{val}(L(G)) \mid$ G is a regular tree grammar over $\Delta\}$. Intuitively, Val(REGT) is the class of "values of regular tree languages" over Δ (where values of expressions are graphs).

As an example, consider the context-free grammar G_b that has one nonterminal A, with type(A) = $(1, 0)$, and two productions $A \to c_g \circ (A + A)$ and $A \to c_{g'}$, where g is the triangle of type $(1, 2)$ with set of nodes $\{x, y, z\}$, set of edges $\{(x, y), (x, z), (y, z)\}$, begin($g$) = $\langle x \rangle$ and end(g) = $\langle y, z \rangle$, and g' is the graph of type $(1, 0)$ with one node u, no edges, begin(g') = $\langle u \rangle$ and end(g') is the empty sequence. Then val($L(G_b)$) is the set of all graphs of type $(1, 0)$ that are obtained from binary trees by connecting each pair of brothers by an additional edge; the sequence of begin nodes consists of the root of the binary tree. This graph language is therefore in Val(REGT).

Our first result is that generating graph languages in the above way is equivalent to

generating them with HR grammars. As observed above, this is a simple variant of the result of [1]. Let HR denote the class of all graph languages generated by HR grammars.

Theorem 1 Val(REGT) = HR.

Since concatenation of graphs is associative, strings can be viewed as expressions that denote graphs. Thus, as an even simpler variation of the above approach, we can use ordinary string grammars to generate graph languages. More generally, every class K of string languages defines a class Int(K) of graph languages (where Int stands for "interpretation", which is similar to Val above). Formally, a mapping $h : \Sigma \to \mathrm{GR}$ that associates a graph with each symbol from an alphabet Σ, is extended to a (partial) function from Σ^* to GR by:

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

where $a_i \in \Sigma$ for all i, and $n \geq 1$. Note that the extended h is partial because the types of the $h(a_i)$ may not fit (and h is also undefined for the empty string). Thus, the only "technical trouble" is that the concatenation of graphs is typed whereas the concatenation of strings is always possible. For a string language $L \subseteq \Sigma^*$, we define, as usual, the set of graphs $h(L) = \{g \in \mathrm{GR} \mid g = h(w) \text{ for some } w \in L\}$; note that $h(L)$ need not be a graph language (with our particular meaning of the term) because not all graphs need have the same type. Now we define $\mathrm{Int}(K) = \{h(L) \mid L \in K, h : \Sigma \to \mathrm{GR} \text{ with } L \subseteq \Sigma^*, h(L) \text{ is a graph language}\}$. In other words, 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 the concatenation of symbols is interpreted as concatenation of graphs.

To avoid trivialities we will, in what follows, and without always mentioning it, only consider classes K that are closed under (nondeterministic) sequential machine mappings (where a sequential machine is an ordinary finite automaton that, moreover, at each step outputs one symbol). Thus, in particular, K is closed under intersection with regular languages and under finite substitutions. Note that every semi-AFL is closed under sequential machine mappings.

The first class K of interest is the class REG of regular languages. An example of a graph language in Int(REG), of type $(1,0)$, is $h(ab^*)$ with $h(a) = g$ and $h(b) = g'$, where g and g' are as in the example grammar G_b, except that now g is of type $(1,1)$ and has $\mathrm{end}(g) = \langle z \rangle$. This graph language is the subset of $\mathrm{val}(L(G_b))$ consisting of all sequences of concatenated triangles, where two consecutive triangles have one node in common (and each node belongs to at most two triangles).

Our second result says that the graph languages that are interpretations of a regular language are precisely those that can be generated by *linear* HR grammars, where "linear" means that there is at most one nonterminal in each right-hand side of a production of the HR grammar. This can be proved by comparing right-linear string grammars (which have productions of the form $A \to aB$ and $A \to a$) with linear HR grammars (of which, roughly speaking, the productions can be written as $A \to h(a) \circ B$ and $A \to h(a)$ for an appropriate h). Let LIN-HR denote the class of graph languages generated by linear HR grammars.

Theorem 2 Int(REG) = LIN-HR.

Before continuing with other classes K, we first consider a third characterization of the class $\mathrm{Int}(\mathrm{REG})$, corresponding to the characterization of REG by regular expressions. 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 (Kleene) star of a graph language is defined (by iterated concatenation): for a graph language L with $\mathrm{type}(L) = (k,k)$ for some $k \in \mathbb{N}$, $L^* = \bigcup_{n\in\mathbb{N}} L^n$ where $L^n = L \circ \cdots \circ L$ (n times) for $n \geq 1$, and $L^0 = \{\mathrm{id}_k\}$. Also, $L^\oplus = L^* - \{\mathrm{id}_k\}$ 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. From this it should be clear that it equals the class $\mathrm{Int}(\mathrm{REG})$ of interpretations of regular languages. In the proof one has to cope with the "technical trouble" of typing, in particular with the empty string. Note that, for a graph language L with $\mathrm{type}(L) = (k,k)$, $L^* = L^\oplus \cup \{\mathrm{id}_k\}$; from this it should be clear that $\mathrm{REX}(\cup, \circ, *, \mathrm{SING}) = \mathrm{REX}(\cup, \circ, \oplus, \mathrm{SING})$, which solves the problem with the empty string.

Theorem 3 $\mathrm{Int}(\mathrm{REG}) = \mathrm{REX}(\cup, \circ, *, \mathrm{SING})$.

By Theorems 2 and 3, $\text{LIN-HR} = \mathrm{REX}(\cup, \circ, *, \mathrm{SING})$. This suggests that the class LIN-HR of linear HR graph languages might be called the class of "regular" graph languages, because they can be denoted by regular expressions. The above characterization still holds after adding the sum operation (extended to graph languages in the usual way). This is because of the following simple reason.

Lemma 4 *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 $M + \{id_k\}$ for every k. Let $M = h(L)$ for some $L \in K$. Define $h'(a) = h(a) + id_k$ for every symbol a. Then $h'(a_1 \cdots a_n) =$

$$(h(a_1) + id_k) \circ \cdots \circ (h(a_n) + id_k) = (h(a_1) \circ \cdots \circ h(a_n)) + (id_k \circ \cdots \circ id_k)$$

because of strict monoidality, and the last expression equals $h(a_1 \cdots a_n) + id_k$. This implies that $h'(L) = h(L) + \{id_k\} = M + \{id_k\}$. Similarly it can be shown that $\{id_k\} + M$ is in $\mathrm{Int}(K)$.

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

It turns out that, if we allow + in our regular expressions, then we do not need all singleton graph languages to start with, but only a "small" number of them, with very simple graphs only. In fact, graphs can be decomposed into very simple graphs, using concatenation and sum. Assume that the edge labels of our graphs are taken from a given set A.

For $a \in A$ let $\mathrm{gr}(a)$ be the graph of type $(1,1)$ with two nodes x and y, an a-labeled edge from x to y, $\mathrm{begin}(\mathrm{gr}(a)) = \langle x \rangle$, and $\mathrm{end}(\mathrm{gr}(a)) = \langle y \rangle$. For $m, n \in \mathbb{N}$, let $E_{m,n}$ be the graph of type (m,n) with one node x, no edges, $\mathrm{begin}(E_{m,n}) = \langle x, \ldots, x \rangle$ (m times), and $\mathrm{end}(E_{m,n}) = \langle x, \ldots, x \rangle$ (n times). Finally, let X be the graph of type $(2,2)$ with two nodes x and y, no edges, $\mathrm{begin}(X) = \langle x, y \rangle$, and $\mathrm{end}(X) = \langle y, x \rangle$. Note also that id_0 is the empty graph. Now define the set of graphs $\mathrm{C}_0 = \{\mathrm{gr}(a) \mid a \in A\} \cup \{\mathrm{id}_0, X, E_{0,1}, E_{1,0}, E_{1,2}, E_{2,1}\}$, and let $\mathrm{REX}(\cup, \circ, *, +, \mathrm{SING}(\mathrm{C}_0))$ denote the smallest class of graph languages containing the empty graph language and all singleton graph languages with a graph from C_0 as element, and closed under the operations union, concatenation, star, and sum.

Theorem 5 $\mathrm{REX}(\cup, \circ, *, \mathrm{SING}) = \mathrm{REX}(\cup, \circ, *, +, \mathrm{SING}(\mathrm{C}_0))$.

From Theorem 2 we know that Int(REG) = LIN-HR. It is not difficult to prove that also Int(LIN) = LIN-HR, where LIN is the (usual) class of languages generated by linear context-free grammars. This suggests that for graph languages the notions "regular" and "linear" coincide, as opposed to the string case. Even Int(DB) = LIN-HR, where DB is the class of derivation bounded context-free languages. One now wonders how much larger the class K can be made without getting a larger class $\mathrm{Int}(K)$.

It is easy to see that for every given class K there is always a largest class K' such that $\mathrm{Int}(K') = \mathrm{Int}(K)$. We will call this the *extension of* K, denoted $\mathrm{Ext}(K)$. In fact, $\mathrm{Ext}(K) = \{L \mid \text{for every } h : \Sigma \to \mathrm{GR} \text{ with } L \subseteq \Sigma^*, \text{ if } h(L) \text{ is a graph language, then } h(L) \in \mathrm{Int}(K)\}$. 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 $w = a_1 \cdots a_n$, $\mathrm{gr}(w)$ is the graph of type $(1,1)$ with nodes $1, \ldots, n+1$, an a_i-labeled edge from i to $i+1$ for every $1 \leq i \leq n$, begin node 1, and end node $n+1$. Thus, $\mathrm{gr}(w)$ encodes w in the obvious way: it is a path with the symbols of w as edge labels. Recall that we assume K to be closed under sequential machine mappings.

Theorem 6 $\mathrm{Ext}(K) = \mathrm{Str}(\mathrm{Int}(K))$.

In the proof of this theorem it has to be shown that $\mathrm{Int}(\mathrm{Str}(\mathrm{Int}(K))) = \mathrm{Int}(K)$, and that if $\mathrm{Int}(K') = \mathrm{Int}(K)$ then $K' \subseteq \mathrm{Str}(\mathrm{Int}(K))$. The first statement is the most involved one to show. The second statement is easy to see. In fact, $\mathrm{gr}(K') \subseteq \mathrm{Int}(K')$: just take $h(a) = \mathrm{gr}(a)$. Then we get $K' \subseteq \mathrm{Str}(\mathrm{gr}(K')) \subseteq \mathrm{Str}(\mathrm{Int}(K')) = \mathrm{Str}(\mathrm{Int}(K))$.

As a corollary of Theorem 6 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).

Next we will aim at another characterization of $\mathrm{Ext}(K)$. First we consider *controlled* linear HR grammars, in the obvious sense. Let G be a linear HR grammar with (finite) set of productions P, 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 for which there is a derivation $g_0 \Rightarrow_{p_1} g_1 \Rightarrow_{p_2} g_2 \cdots \Rightarrow_{p_n} g_n$ with $g_n = g$, g_0 is the axiom of G, and such that the string $p_1 p_2 \cdots p_n$ is in C. Of course, $\Rightarrow_p$ denotes a derivation step of G that uses production $p \in P$. Thus, the control language C specifies the sequences of productions that the grammar G is allowed to use in its derivations.

For a class K of string languages, we denote by LIN-HR(K) the class of graph languages generated by linear HR grammars under a control language from K. Generalizing Theorem 2 and its proof we obtain the next result.

Theorem 7 $\mathrm{Int}(K) = \text{LIN-HR}(K)$.

By 2DGSM(K) we denote the class of images of languages from K under 2dgsm mappings, i.e., $\mathrm{2DGSM}(K) = \{f(L) \mid f \text{ is a 2dgsm mapping and } L \in K\}$. 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. Generalizing the proof in [6] that Str(LIN-HR) equals the class of output languages of 2dgsm mappings, we prove the next result.

Theorem 8 $\mathrm{Str}(\text{LIN-HR}(K)) = \mathrm{2DGSM}(K)$.

Taking these (quite obvious) generalizations together, we obtain from Theorems 6, 7, and 8 our second characterization of the class $\mathrm{Ext}(K)$.

Theorem 9 $\mathrm{Ext}(K) = \mathrm{2DGSM}(K)$.

This characterization allows us to use known formal language theoretic results regarding 2DGSM(K) to investigate $\mathrm{Int}(K)$. In fact, quite a lot is known about the class $\mathrm{2DGSM}(K)$, see, e.g., [7]. As an example, it equals the class of languages generated by K-controlled ETOL systems of finite index.

The other way around, we note that from $\mathrm{Int}(\mathrm{Str}(\mathrm{Int}(K))) = \mathrm{Int}(K)$ follows that $\mathrm{Str}(\mathrm{Int}(\mathrm{Str}(\mathrm{Int}(K)))) = \mathrm{Str}(\mathrm{Int}(K))$, i.e., the known result that $\mathrm{2DGSM}(\mathrm{2DGSM}(K)) = \mathrm{2DGSM}(K)$. This shows that $\mathrm{Ext}(K)$ is closed under 2dgsm mappings.

Thus, for $K = \mathrm{REG}$, $\mathrm{Ext}(K)$ 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., [10]), this implies the previously mentioned result that $\mathrm{Int}(\mathrm{LIN}) = \mathrm{Int}(\mathrm{DB}) = \mathrm{Int}(\mathrm{REG})$. Also, since there is a context-free language not in 2DGSM(REG), see [8, 7], $\mathrm{Int}(\mathrm{REG})$ is properly included in $\mathrm{Int}(\mathrm{CF})$, where CF is the class of context-free languages. We finally note that $\mathrm{Int}(\mathrm{CF})$ is properly included in HR, the class of graph languages generated by HR grammars. The inclusion follows from the fact that context-free grammars generating graph expressions built from concatenation and all constant graphs, can be simulated by HR grammars, by Theorem 1. Properness of the inclusion follows from the more general fact that for any class K, $\mathrm{Int}(K)$ contains graph languages of bounded path-width only. Thus, the set of all binary trees (which is in HR) does not belong to any $\mathrm{Int}(K)$.

References

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

[2] 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.

[3] B. Courcelle, J. Engelfriet, G. Rozenberg; Handle-rewriting hypergraph languages, JCSS 46 (1993), 218–270.

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

[5] 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.

[6] J. Engelfriet, L.M. Heyker; The string generating power of context-free hypergraph grammars, JCSS 43 (1991), 328–360.

[7] J. Engelfriet, G. Rozenberg, G. Slutzki; Tree transducers, L systems, and two-way machines, JCSS 20 (1980), 150–202.

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

[9] A. Habel; *Hyperedge replacement: grammars and languages*, Lecture Notes in Computer Science 643, Springer-Verlag, Berlin, 1992.

[10] V. Rajlich; Absolutely parallel grammars and two-way finite state transducers, JCSS 6 (1972), 324–342.

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