INFORMATIQUE THÉORIQUE ET APPLICATIONS

JOOST ENGELFRIET
GEORGE LEIH

Complexity of boundary graph languages

Informatique théorique et applications, tome 24, nº 3 (1990), p. 267-274
<http://www.numdam.org/item?id=ITA_1990__24_3_267_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 24, n° 3, 1990, p. 267 à 274)

COMPLEXITY OF BOUNDARY GRAPH LANGUAGES (*)

by Joost ENGELFRIET (1) and George LEIH (1, 2)

Communicated by P. VAN EMDE BOAS

Abstract. – *Connected graph languages of bounded degree that are generated by boundary eNCE grammars are in* LOG(*CF*), i.e., *they are* log-*space reducible to a context-free string language.*

Résumé. – *Les langages de graphes connexes et de degré borné qui sont engendrés par des grammaires de type « boundary eNCE » sont dans* LOG(*CF*), *c'est-à-dire qu'il est possible de réduire ces langages à un langage algébrique en espace logarithmique.*

NLC and *NCE* graph grammars ([7, 8]) have been investigated intensively. They have been shown to be adequate for defining sets of (undirected, node labeled) graphs in essentially the same way as context-free string grammars can be used to define sets of strings. As a straightforward generalization of *NLC* and *NCE* graph grammars, *eNLC* and *eNCE* grammars can be used to generate sets of graphs which are edge labeled as well (*see*, e.g., [10, 9, 2, 3, 4]; the *e* stands for edge labeled, *NLC* for Node Label Controlled, and *NCE* for Neighbourhood Controlled Embedding).

One of the most interesting restrictions on *NLC* graph grammars, proposed in the literature, is the "boundary" restriction: edges between nonterminal nodes are not allowed. These boundary *NLC* (or *B-NLC*) grammars were introduced in [12]. Due to the boundary restriction *B-NLC* grammars are even closer to context-free grammars than arbitrary *NLC* grammars, and thus have nicer properties with respect to, e.g., normal forms, decidability, closure properties, and complexity of recognition. Recently we have started

(*) Received May 1988, revised.

(1) Department of Computer Science, University of Leiden, P.O. Box 9512, 2300 RA Leiden, The Netherlands.

(2) The work of this author was conducted as part of the PRISMA project, a joint effort with Philips Research, partially supported by the Dutch "Stimuleringsprojectteam informaticaonderzoek" (SPIN).

to investigate the boundary *eNCE* (or *B-eNCE*) graph grammars, of which the *B-NLC* grammars are a special case (*see* [4]). In our experience these *B-eNCE* grammars enjoy all the nice properties of the *B-NLC* grammars, whereas they are much easier to handle and understand (mainly because of the way edge labels may be manipulated by *eNCE* grammars). Moreover, it is shown in [4] that some additional results, such as a Chomsky and a Greibach normal form, hold for *B-eNCE* grammars which cannot be obtained for *B-NLC* grammars.

In this note we consider the complexity of recognizing *B-eNCE* languages. It was shown in [12] that connected *B-NLC* languages of bounded degree are in *P*, *i.e.*, can be recognized in polynomial time (and the same holds for node relabelings of such languages). We generalize and improve this result by showing that connected *B-eNCE* languages of bounded degree are in LOG(*CF*), *i.e.*, are log-space reducible to a context-free language. Such a result can be used as a quick method for showing that specific sets of graphs are in LOG(*CF*), *cf.* [16].

Independently, Lautemann shows in [11] a similar, but stronger, result for a related type of graph grammar. He shows that a particular subset of the hyperedge replacement languages is in LOG(*CF*). It follows from the results of [5] that this subset (properly) contains the set of connected *B-eNCE* languages of bounded degree.

We consider undirected node and edge labeled graphs without loops; multiple edges are allowed, but not with the same label. Formally, a *graph* is a system $H=(V, E, \Sigma, \Gamma, \varphi)$, where V is the finite set of nodes, $E \subseteq \{\{v, \lambda, w\} \mid v, w \in V, v \neq w, \lambda \in \Gamma\}$ is the set of labeled edges, Σ is the alphabet of node labels, Γ is the alphabet of edge labels, and $\varphi : V \to \Sigma$ is the node labeling function. We use $GR_{\Sigma, \Gamma}$ to denote the set of all graphs with node label alphabet Σ and edge label alphabet Γ. The components of graph H will be indicated by V_H, E_H, Σ_H, Γ_H, and φ_H. A graph language is a set of graphs. A graph language L is connected if all graphs in L are connected, and it is of bounded degree if there is a d such that each node in each graph of L has degree $\leq d$ (*i.e.*, has at most d incident edges).

An *eNCE grammar* is a system $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$, where Σ is the alphabet of node labels, $\Delta \subseteq \Sigma$ is the alphabet of terminal node labels (elements of Σ-Δ are called nonterminal node labels), Γ is the alphabet of edge labels, $\Omega \subseteq \Gamma$ is the alphabet of final edge labels, P is the finite set of productions, and $S \in \Sigma$-Δ is the initial nonterminal. A production $\pi \in P$ is of the form $\pi=(X, D, B)$, where $X \in \Sigma$-Δ is the left-hand side of π [denoted $lhs(\pi)$],

$D \in GR_{\Sigma, \Gamma}$ is the right-hand side of π [denoted $rhs(\pi)$], and $B \subseteq V_D \times \Gamma \times \Gamma \times \Sigma$ is the embedding relation of π [denoted $B(\pi)$].

A production $\pi = (X, D, B)$ is applied to a nonterminal node v in a graph $H \in GR_{\Sigma, \Gamma}$, where $\varphi_H(v) = X$, as follows. First, v is removed from H, together with all edges incident with v. Next, D is added to the remainder of H, in place of v. Finally, D is embedded in the remainder of H by adding edges between nodes in V_D and former neighbours of v in H as follows. If $x \in V_D$ and $y \in V_H - \{v\}$, then an edge labeled μ is added between x and y if and only if there was an edge between v and y labeled λ in H, and $(x, \lambda, \mu, \varphi_H(y))$ is in B. Thus, x inherits some of the edges that connect v to its neighnours, possibly with a different label. The result of this transformation is a graph K in $GR_{\Sigma, \Gamma}$. $H \Rightarrow_{(v, \pi)} K$ or just $H \Rightarrow K$ will be used to denote the transformation (for a more formal definition *see* [4]). The *language generated by* G is $L(G) = \{H \in GR_{\Delta, \Omega} \mid \underline{S} \Rightarrow^* H\}$ (where $\underline{S}$ is a graph with just one node labeled S, and $\Rightarrow^*$ is the transitive and reflexive closure of $\Rightarrow$). Thus, $L(G)$ contains all graphs derivable from $\underline{S}$ which have only terminal nodes and final edges. The class of all languages generated by *eNCE* grammars is denoted *eNCE*.

Next, we introduce the subclass of *eNCE* grammars we are interested in in this paper. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an *eNCE* grammar. G is a *B-eNCE grammar* (*B* for boundary) if, for every $\pi \in P$, $rhs(\pi)$ contains no edges between nonterminal nodes. The class of all languages generated by a *B-eNCE* grammar is denoted *B-eNCE*. It is easy to see that in sentential forms of *B-eNCE* grammars no edges can appear between nonterminal nodes. Therefore, the order of rewriting two nonterminal nodes in a sentential form of a *B-eNCE* grammar does not influence the result (in contrast to arbitrary *eNCE* grammars). This observation is important to be able to understand the proof of the result of this note.

In order to prove the LOG(*CF*) result, it is convenient to use the following normal forms. An *eNCE* grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is in *Greibach normal form* if, for every production $\pi \in P$, $rhs(\pi)$ contains exactly one terminal node. G is *neighbourhood preserving* if for all H and K such that $\underline{S} \Rightarrow^* \mathrm{H} \Rightarrow_{(x, \pi)} K$ in G, if $\{x, \lambda, y\} \in E_H$, then there is a node $z \in V_{rhs(\pi)}$ and a $\mu \in \Gamma$ such that $(z, \lambda, \mu, \varphi_H(y)) \in B(\pi)$. Thus, all edges incident with x are "used" to establish a new edge in K. Using the results in the literature, it can be shown that there is a neighbourhood preserving *B-eNCE* grammar in Greibach normal form for each *B-eNCE* language (the neighbourhood preserving result is shown in [5], the Greibach result in [4]; it is not difficult to see that the proof in [5] preserves the Greibach property).

We are now ready to prove that connected *B-eNCE* languages of bounded degree are in LOG (*CF*). Since LOG $(CF) \subseteq P$ this improves the result in [12] for *B-NLC* languages. Moreover, since every context-free language can be generated by a *B-eNCE* grammar (coding a string as an edge labeled chain), this is the best possible result, using log-space reductions. Our proof is based on the fact that LOG (*CF*) is the class of languages accepted by alternating Turing machines that use logarithmic space and polynomial tree-size ([13, 14]). It also uses the recent result that NSPACE $(\log n)$ is closed under complement ([6, 15]).

THEOREM: *If $L \in$ B-eNCE is a connected graph language of bounded degree, then $L \in$* LOG (*CF*).

Proof: Let $d \geqq 1$ be such that each graph in L has degree $\leqq d$. Let $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a neighbourhood preserving *B-eNCE* grammar in Greibach normal form with $L(G)=L$. Assume that S does not appear as label in the right-hand side of a production. Let $H \in GR_{\Delta, \Omega}$ be a connected graph (this can be checked in nondeterministic $O(\log n)$ space, where n is the length of H when encoded as a string in the usual way). We give an algorithm for an alternating Turing machine that checks in $O(\log n)$ space and polynomial tree-size that $H \in L$. The algorithm consists of one recursive boolean function, invoked as Parse $(S, \varnothing, \varnothing)$. It is a generalized version of Algorithm 20 in [2], which in turn is based on Algorithm 3.5 of [1]. The statement "CHECK THAT α" is syntactic sugar for "IF NOT α THEN RETURN (FALSE) FI".

1. FUNCTION Parse ($X \in \Sigma$-Δ, Crit-edges $\subseteq V_H \times \Omega \times V_H$, Boundary $\subseteq V_H \times \Gamma$): Boolean;
   BEGIN
2. CHOOSE $v_0 \in V_H$ and $\pi \in P$;
3. LET $k = \# V_{rhs(\pi)} - 1$, and $V_{rhs(\pi)} = \{\xi_0, \xi_1, \ldots, \xi_k\}$, where
   $\varphi_{rhs(\pi)}(\xi_0) \in \Delta$ and $\varphi_{rhs(\pi)}(\xi_i) \in \Sigma$-$\Delta$, for $1 \leqq i \leqq k$;
4. CHECK THAT $lhs(\pi) = X$ AND $\varphi_{rhs(\pi)}(\xi_0) = \varphi_H(v_0)$;
5. IF $X \neq S$ THEN
6. CHECK THAT there is a $(u, \lambda, v) \in$ Crit-edges such that there is a path between v and v_0 in H only through edges not in Crit-edges
   FI;
7. CHECK THAT $\{(u, \mu, v_0) \mid u \in V_H, \mu \in \Gamma, \exists \lambda \in \Gamma$:
   $(u, \lambda) \in$ Boundary, $(\xi_0, \lambda, \mu, \varphi_H(u)) \in B(\pi)\}$
   $= \{(u, \mu, v_0) \in$ Crit-edges $\mid u \in V_H, \mu \in \Gamma\}$;
8. Crit-edges := Crit-edges
   $- \{(u, \mu, v_0) \in$ Crit-edges $\mid u \in V_H, \mu \in \Gamma\}$
   $\cup \{(v_0, \mu, u) \mid u \in V_H, \mu \in \Gamma, \{v_0, \mu, u\} \in E_H, (u, \mu, v_0) \notin$ Crit-edges$\}$;
9. IF $k \geqq 1$ THEN
10. CHECK THAT $\#$ Crit-edges $\geqq k$;
11. CHOOSE a partition $\{P_1, P_2, \ldots, P_k\}$ of Crit edges;
12. FOR each $i, j \in \{1, 2, \ldots, k\}$ with $i \neq j$, and each $(u, \lambda, v) \in P_i$ and $(x, \mu, y) \in P_j$ # DO

13. CHECK THAT there is no path between v and y in H only through edges not in Crit-edges
OD;
14. FOR each $i \in \{1, 2, \ldots, k\}$ DO
15. Boundary$_i$:= $\{(u, \mu) \mid \exists \lambda \in \Gamma : (u, \lambda) \in$ Boundary, $(\xi_i, \lambda, \mu, \varphi_H(u)) \in B(\pi)\}$
$\cup \{(v_0, \mu) \mid \{\xi_0, \mu, \xi_i\} \in E_{rhs(\pi)}\}$;
16. CHECK THAT Parse $(\varphi_{rhs(\pi)}(\xi_i), P_i$, Boundary$_i)$
OD
17. ELSE CHECK THAT Crit-edges $= \varnothing$
FI;
18. RETURN (TRUE)
END.

The idea behind this algorithm is that a derivation for H in G is guessed. For each nonterminal node ξ that is generated during this derivation, a process is created (16) that checks a subgraph D of H by applying function Parse, with an appropriate set of parameters (1 and 16). Since D itself cannot be encoded in $O(\log n)$ space, we take the following parameters: X is the label of the nonterminal node ξ that has to generate subgraph D, Crit-edges (critical edges) is the set of all tuples (u, μ, v) for which $\{u, \mu, v\} \in E_H$, $u \in V_H\text{-}V_D$, and $v \in V_D$ (since H is connected, this set uniquely determines D), and Boundary is the set of all tuples (u, λ) such that $u \in V_H\text{-}V_D$ is connected to ξ by a λ-edge. A process, applying the function Parse, accepts if and only if RETURN (TRUE) (18) is executed. Otherwise it rejects (one of the checks failed).

The process guesses a node $v_0 \in V_H$ and a production $\pi \in P$ that has to generate v_0 (see 2 to 4; since G is in Greibach normal form, each production generates one terminal node). It is checked that v_0 is in D (this can be done

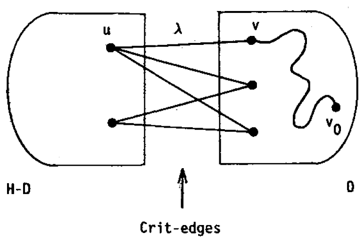

Figure 1

as in 6 since H is connected, see *Fig.* 1; this trick was first used in [1]; note that only the first process has $X = S$ in 5, see our assumption). In 7 it is checked that all edges between nodes outside D and v_0 are established, and no more. These edges are thrown out of Crit-edges in 8, and the remaining edges incident with v_0 in H now get critical. If there are $k \geq 1$ nonterminal nodes in $rhs(\pi)$ (*see* 9), then k new processes have to be created. It is guessed

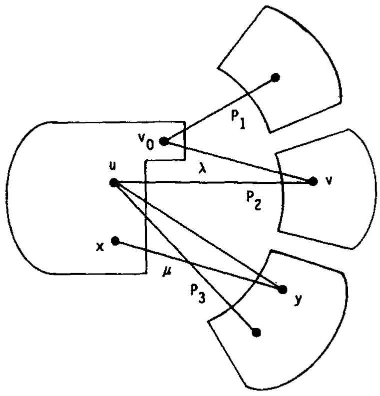

Figure 2

in 11 which nonterminal node has to generate which part of the remainder of D, by partitioning Crit-edges into k parts P_1 to P_k (see *Fig.* 2, where $k=3$ and $\# P_i=i$). This partitioning is possible, *see* 10. 12-13 ensures that the processes work on disconnected parts of this remaining graph (we now know that P_i determines a unique subgraph of D). In 15 the set of edges incident with ξ_i is defined. We see that the k new processes, which start working in parallel, are created in 16. If all these subprocesses return true, then the process accepts (18). If, in 17, no nonterminal node is left then it is checked that Crit-edges $=\varnothing$; the connectedness of H ensures in this case that the process is ready generating D.

Finally we discuss the log-space and polynomial tree-size realization of the algorithm. We first consider space. It is clear that 6 can be done in NSPACE$(\log n)$, and 13 in co-NSPACE $(\log n)$, and hence also in NSPACE$(\log n)$ [6, 15]. Second, we show that the parameters take $O(\log n)$ space at most. It suffices to prove that #Boundary and #Crit-edges are bounded by a constant. We claim that #Boundary $\leqq \#\Gamma . \#\Delta . d$ and #Crit-edges $\leqq \#\Gamma . \#\Delta . d^2$ (if $H\in L$ and the derivation is guessed correctly). Since G is neighbourhood preserving, each edge indicated by a tuple in Boundary (thus incident with ξ) will be used to establish an edge in H. Hence, there may not be an edge label λ and a node label b such that there are more than d tuples (u, λ) in Boundary with $\varphi_H(u)=b$ (in fact, all these u's would finally get connected to the same node of H, which then would have degree $>d$). Thus, the bound on #Boundary is correct. Similarly, for a fixed $u\in V_H$, there can be at most d tuples (u, λ, v) in Crit-edges. Since such an edge can only be established if u also appears in Boundary (*i.e.*, is connected to ξ), the bound on #Boundary implies the bound on #Crit-edges. Thus, the amount of space given to the processes suffices. Next we consider the tree-size. Clearly the tree of recursive calls of the algorithm has the same size as the guessed derivation tree, which is of linear size since G is in Greibach normal form. It now suffices to observe that each recursive call takes (nondeterministic) polynomial time: this is because it works in NSPACE ($\log n$), as discussed before. Thus the tree-size is polynomial. ■

Finally, we wish to mention that it easily follows from the proof above that connected linear *eNCE* languages of bounded degree are in NSPACE($\log n$) (a linear grammar has the property that the right-hand side of each production contains at most one nonterminal node, *see* [2]; thus, linear grammars are a special type of boundary grammars). If, namely, G in the proof is linear, then k is at most one, and hence there are no concurrent processes. This was first proved in [2].

REFERENCES

1. IJ. J. Aalbersberg, J. Engelfriet and G. Rozenberg, *The Complexity of Regular DNLC Graph Languages*, Report 86-03, Leiden, April 1986, JCSS (to appear).
2. J. Engelfriet and G. Leih, *Linear Graph Grammars: Power and Complexity*, Information and Computation, 81, 1989, pp. 88-121.
3. J. Engelfriet, G. Leih and G. Rozenberg, *Apex Graph Grammars*, in *Graph-Grammars and their Application to Computer Science*, H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld Eds., Lecture Notes in Computer Science, 291, Springer-Verlag, Berlin, 1987, pp. 167-185.
4. J. Engelfriet, G. Leih and E. Welzl, *Boundary Graph Grammars with Dynamic Edge Relabeling*, Report 87-30, Leiden, December 1987, J. Comput. System Sci. (to appear).
5. J. Engelfriet and G. Rozenberg, *A Comparison of Boundary Graph Grammars and Context-Free Hypergraph Grammars*, Information and Computation, 84, 1990, pp. 163-206.
6. N. Immermann, *Nondeterministic Space is Closed Under Complement*, Yale University, Technical Report, YALEU/DCS/TR 552, July 1987.
7. D. Janssens and G. Rozenberg, *On the Structure of Node-Label-Controlled Graph Languages*, Information Sciences, 20, 1980, pp. 191-216.
8. D. Janssens and G. Rozenberg, *Graph Grammars with Neighbourhood-Controlled Embedding*, Theor. Comp. Science, 21, 1982, pp. 55-74.
9. D. Janssens, G. Rozenberg and R. Verraedt, *On Sequential and Parallel Node-Rewriting Graph Grammars*, Computer Graphics and Image Processing, 18, 1982, pp. 279-304.
10. M. Kaul, *Syntaxanalyse von Graphen bei Präzedenz-Graph-Grammatiken*, Dissertation, Universität Osnabrück, 1985.

11. C. LAUTEMANN, *Efficient Algorithms on Context-Free Graph Languages*, in Proc. 15th I.C.A.L.P., T. LEPISTÖ and A. SALOMMA Eds., Lecture Notes in Computer Science, 317, Springer-Verlag, Berlin, 1988, pp. 362-378.
12. G. ROZENBERG and E. WELZL, *Boundary NLC Graph Grammars-Basic Definitions, Normal Forms, and Complexity*, Inform. Contr., Vol. 69, 1986, pp. 136-167.
13. W. L. RUZZO, *Tree-Size Bounded Alternation*, J.C.S.S., 21, 1980, pp. 218-235.
14. I. H. SUDBOROUGH, *On the Tape Complexity of Deterministic Context-Free Languages*, J.A.C.M., Vol. 25, 1978, pp. 405-414.
15. R. SZELEPCSÉNYI, *The Method of Forcing for Nondeterministic Automata*, Bulletin of the E.A.T.C.S., Vol. 33, 1987, pp. 96-99.
16. E. WELZL, *Boundary NLC and Partition Controlled Graph Grammars*, in "*Graph-Grammars and their Application to Computer Science*", H. EHRIG M. NAGL, G. ROZENBERG and A. ROSENFELD Eds., Lecture Notes in Computer Science, 291, Springer-Verlag, Berlin, 1987, pp. 593-609.