1984

Edge Grammars: Decidability Results and Formal Language Issues

Francine Berman

Gregory Shannon

Report Number:
84-489

TR-489

Edge Grammars: Decidability Results and Formal Language Issues

Francine Berman
Gregory Shannon
Purdue University

Motivation

Graphs and graph families are a fundamental tool in almost every area of computer science. They are used to represent algorithms, architectures, data structures, data flow, automata, etc.

Many applications use graphs and graph families as an abstraction and require a formal means of defining graphs, in the same way that context-free grammars are used to defined Type 2 languages in the construction of compilers. In [B], we introduced *edge grammars* as a formalism in which to define and manipulate graphs and graph families. Essentially, edge grammars are systems which define graph families by generating pairs of strings which represent edges in the individual graphs. These pairs are generated by derivations using the productions of the edge grammar in much the same way strings in conventional formal languages are generated. Edge grammars were developed to fill a gap left by other graph generating systems: In particular, we were interested in representing graph families commonly used as interconnection networks in parallel computation, (e.g. shuffle-exchange graphs, cube-connected cycles, meshes, trees, etc.) many of which could not be represented by existing graph grammars and graph generating systems [CER], [ENR]. To this end, we explored the relationship of edge grammars to parallel computation in the introductory paper.

In this paper, we provide a solid context for the study of edge grammars by relating edge grammars to conventional formal language theory and to the existing work on graph grammars. We also explore decidability issues for edge grammars. In presenting the decidability results, we come full circle and resolve questions important in the motivating applications.

Section 1: Definitions

In this section, we give a brief review of edge grammars and an example of a graph family generated by one. For a more thorough description of edge grammars, see [B].

Definition: A (Type 0) **edge grammar** is a 4-tuple $<N,T,G,P>$ where N is a set of nonterminal symbols, T is a set of terminal pairs $\{(v,w)\}$ (v and w are strings over a finite alphabet), G in N is the start symbol and P is a set of productions. Productions in P have the form $x \rightarrow y$ where x and y are strings in $(N \cup T)^*$ and x is not empty. Note that we interpret the concatenation of two pairs (a,b)(c,d) to be the pair (ac,bd).

Essentially, a Type 0 edge grammar is exactly like a Type 0 (Chomsky) grammar except that instead of single terminal symbols, edge grammars have terminal pairs.

Definition: An edge grammar is **Type 3** (right linear) if the productions have the form $A \rightarrow B$, $A \rightarrow (a,b)B$ or $A \rightarrow (a,b)$ where A and B are nonterminals and (a,b) is a terminal pair; **Type 2** if the productions have the form $A \rightarrow xy$ or $A \rightarrow x$ where A is a nonterminal and x and y are either nonterminals or terminal pairs; and **Type 1** if for every production in P, the number of terminals and nonterminals on the left-hand side of the production does not exceed the number of terminals and nonterminals on the right-hand side.

Definition: Let Γ be an edge grammar. The nth graph generated by Γ, $\mathbf{G_n}$, is the undirected graph with vertices

$$V_n = \{v \mid \text{For some } w, (G \rightarrow {}^*(v,w) \text{ or } G \rightarrow {}^*(w,v)) \text{ and } |v|=|w|=n\}$$

and edges

$$E_n = \{(v,w) \mid G \rightarrow {}^*(v,w), v \neq w \text{ and } |v|=|w|=n\}.$$

In other words, the nth graph G_n is the graph all of whose edges have length n labels.

Definition: Let Γ be an edge grammar. The graph family generated by Γ, $\mathbf{G(\Gamma)}$ is the set $\{G_n\}_{n>0}$ where G_n is the nth graph generated by Γ.

Edge grammars generate pairs of labels. If those labels have the same length, say length n, then we consider them to be an edge in graph G_n. In this way, the edges generated by an edge grammar can be naturally partitioned into a family of graphs $\{G_n\}$. Note that the index n in G_n refers to the length of the labels in G_n and not necessarily the number of nodes in G_n.

To illustrate how edge grammars work, consider the family of shuffle-exchange graphs. A shuffle-exchange graph SE_n consists of 2^n vertices. Each vertex is labelled by a binary n bit string. Adjacent vertices have labels which are either left or right (circular) shifts of one another (the shuffle edges) such as (1000, 0001) and (1000, 0100), or have labels in which the last bit is complemented (the exchange edges) such as (1000, 1001). Shuffle-exchange graphs are particularly effective as parallel architectures for algorithms for sorting, fast fourier transform, performing permutations, etc. [P], [S]. Shuffle-exchange graphs SE_1, SE_2 and SE_3 are shown in Figure 1.

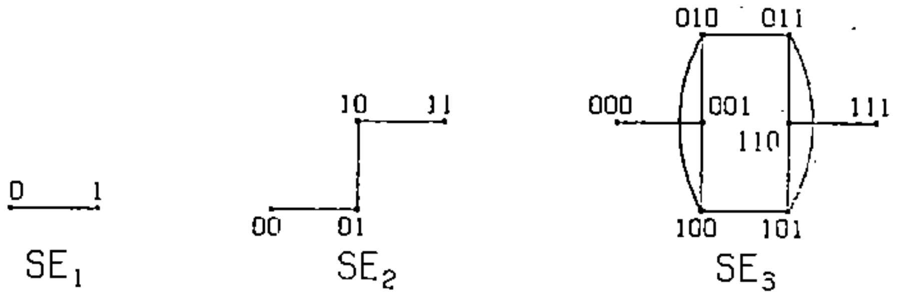


Figure 1
Shuffle-exchange graphs SE_1, SE_2 and SE_3.

The following is a (Type 3) edge grammar for generating the family of shuffle-exchange graphs $\{SE_n\}$.

$G \rightarrow E$	$G \rightarrow S_1$	$G \rightarrow S_0$
$E \rightarrow (0,0)E$	$S_1 \rightarrow (1,\lambda)N_1$	$S_0 \rightarrow (0,\lambda)N_0$
$E \rightarrow (1,1)E$	$N_1 \rightarrow (0,0)N_1$	$N_0 \rightarrow (0,0)N_0$
$E \rightarrow (0,1)$	$N_1 \rightarrow (1,1)N_1$	$N_0 \rightarrow (1,1)N_0$
	$N_1 \rightarrow (\lambda,1)$	$N_0 \rightarrow (\lambda,0)$

Edges in $\{SE_n\}$ are generated by applying a sequence of productions as in conventional formal language theory. For example, the following derivation yields the edge (001, 010) in this edge grammar:

$$G \rightarrow S_0 \rightarrow (0,\lambda)N_0 \rightarrow (0,\lambda)(0,0)N_0 \rightarrow (\lambda,0)(0,0)(1,1)N_0$$

$$\rightarrow (0,\lambda)(0,0)(1,1)(\lambda,0) = (001,\ 010)$$

Since both labels in (001, 010) have length 3, (001, 010) is an edge in E_3 (in SE_3) and 001 and 010 are vertices in V_3.

The usefulness of edge grammars as a representation for graph families is dependent upon how much information about a graph or graph family we can determine from its edge grammar. In the next section, we focus on decidability properties and develope a correspondence between edge grammars and graph grammars.

Section 2: Decidability Results for Edge Grammars.

Edge grammars were originally developed as an automatable representation for graph families as part of a solution to the problem of mapping large-sized instances of parallel algorithms onto small-sized parallel machines [BS], [B]. In this context, it is important to resolve decision problems such as the **membership problem:** *Given a graph H and an edge grammar Γ, is H isomorphic to a member of the graph family G(Γ)?*

The membership problem and other decision problems are critical in assessing the effectiveness of edge grammars as a representation for graphs and graph families. Although we found that the membership problem was undecidable in general, the problem is decidable when there are additional constraints on Γ. For example, if the graph vertex sets V_n of Γ are bounded from below by an unbounded monotone increasing function f_n then the membership problem is decidable. (We call such edge grammars **monotone** edge grammars). Since many of the commonly used parallel interconnection architectures can be generated by monotone edge grammars, the undecidability of the general problem has not proved debilitating to the use of edge grammars in the parallel computation application. The following theorem catalogues some useful decidability results on edge grammars.

Theorem

The following questions are **undecidable.**

1) Given a graph H and an edge grammar Γ, is H isomorphic to a member of G(Γ)?
2) Given a graph H and a Type 3 edge grammar Γ, is H isomorphic to a subgraph of any member of G(Γ)?
3) Given an edge grammar Γ, are the graphs in G(Γ) planar? connected? hamiltonian?
4) Given edge grammars Γ_1 and Γ_2, are the graphs in $G(\Gamma_1)$ isomorphic to the graphs in $G(\Gamma_2)$? Is there a graph isomorphic to a member of $G(\Gamma_1)$ and a member of $G(\Gamma_2)$? (intersection)

The following questions are **decidable.**

5) Given a Type 3 Edge grammar Γ, is G(Γ) finite? infinite? empty?

6) If H is a graph and there exists a monotone, unbounded function f with $|V_n| \geq f(n)$ for all n, is H isomorphic to a member of $G(\Gamma)$?

Proof:

To prove 1), 3) and 4), we reduce arbitrary NLC grammars [JR] to edge grammars and use the undecidability results for NLC grammars. An NLC grammar is a graph grammar in which graphs are generated from a set of initial graphs by applying productions from a fixed set. Upon applying a production $A \rightarrow J$ to a graph H, a single node of H labelled by nonterminal A is replaced by graph J, and J is joined to H according to a given connection chart.

Let G be an NLC grammar. We will construct an edge grammar whose graph family is the family L(G) generated by G. We will do this by constructing a Turing Machine which nondeterministically halts with an edge from L(G) on the tape. By the Hierarchy Theorem in section 3, it is then straightforward to construct an edge grammar Γ which yields the edges on the tape and for which $G(\Gamma)=L(G)$.

Given an NLC G, we construct a Turing Machine M which performs the following procedures:

1) Construct M by encoding the productions and connection chart in the TM program. The tape will be divided into 3 sections:

   #counter#graph#work area#

   The counter is initialized to the maximum number of nodes in an initial graph plus one. The state is initialized to q_1. The graph is initialized to one of the initial graphs in G. Graphs are represented in the middle section as a sequence of pairs $((v,l_v)(w,l_w))$ where (v,w) is an edge in the graph, l_v is the label of v, and l_w is the label of w. The work area is initialized to the empty string.

2) If A is a nonterminal and H is a graph, M simulates the application of production $A \rightarrow H$ by first building subgraph H in the work area. (H is represented as pairs of (vertex,label) pairs analogous to the graph in the graph area). The TM then scans the current graph (in the graph area) and changes the subgraph in the work area according to the connection chart. In the graph area, the interconnection structure of the node labelled with A is also changed. The connected subgraph H in the work area is unioned with the modified graph in the graph area yielding a new current graph (in the graph area). The state is reset to q_1 and another production can be applied to the new current graph.

   Throughout this procedure, the counter is used to create new nodes. When H is constructed in the work area, the counter contains the unary number for the next new vertex. When a new vertex is created, its vertex number becomes the number in the counter, and the counter is incremented with the number for the next new vertex.

3) After the application of any production in a derivation, the Turing Machine may go into state q_2 and check if the label of every vertex is a nonterminal in the NLC grammar. If so, the Turing Machine enters state q_3, otherwise, it may re-enter q_1.

4) In state q_3, we may assume that the graph has only terminal labels. The workspace can then be erased so that the tape loods like

   #counter#graph#

At this point, M is ready to nondeterministically choose one of the edges in the NLC graph and convert it into an edge for an edge grammar graph. The TM scans the graph area, nondeterministically chooses an edge and copies the edge to the right of the graph area.

$$\#counter\#graph\#((v,l_v),(w,l_w))$$

The TM then encodes the *counter#graph* section of the tape into a unique unary number N and replaces this section with this number.

$$\#N \text{ (in unary)}\#((v,l_v),(w,l_w))$$

Finally, M replaces the current contents of the tape with the edge (x,y) where $|x|=|y|$, $x=1^v0^{N-v}$, and $y=1^w0^{N-w}$. After returning the tape head to an initial position, the TM halts. The tape now contains a terminal pair whose left and right coordinates have equal length.

5) By the Hierarchy Theorem (section 3), it is straightforward to construct an edge grammar which generates (x,y). Each NLC graph in L(G) will now appear as a member of the graph family generated by the edge grammar Γ which simulates M, i.e. $L(G)=G(\Gamma)$.

(Note that the TM will not halt if the terminal graph family generated by the NLC grammar G is empty. However this can be decided in advance and if L(G) is empty, it is trivial to construct an edge grammar whose graph family is also empty).

To show that 1), 3) and 4) are undecidable, it is now sufficient to observe that these problems are undecidable for arbitrary NLC grammars [JR] and hence by our construction, for edge grammars.

To prove 2), we first show that the question *"Given a Type 3 edge grammar Γ, does any member G_n of $G(\Gamma)$ contain a self-loop?"* is undecidable. This is a straightforward reduction from the Post Correspondence Problem. Let C be a correspondence system with pairs $\{(u_i,v_i)\}$. Construct a Type 3 grammar Γ with productions

$G \rightarrow (u_i,v_i)G$
$G \rightarrow (u_i,v_i)$

for each i. Then G contains a self-loop iff C has a match. Since the Post Correspondence Problem is undecidable, the self-loop problem is undecidable. To show that the subgraph problem is undecidable, note that if we could determine for a given graph H and a given Type 3 edge grammar Γ whether H was isomorphic to a subgraph of $G(\Gamma)$, then we could instantiate H to be a self-loop and decide the self-loop problem.

To prove 5), note that given a Type 3 edge grammar Γ, the vertex set V_Γ of Γ is contained in the family of context-free languages (see the Hierarchy Theorem in Section 3). Since finiteness, emptiness, and infiniteness are decidable for CFLs, they are decidable for V_Γ. 5) follows from observing that if V_Γ is finite, infinite or empty, then $G(\Gamma)$ must also be finite, infinite or empty. (Note that even if V_Γ is infinite, $G(\Gamma)$ may still be isomorphically equivalent to a finite set of graphs).

To prove 6), we use a simple counting argument. Let f be a monotone, unbounded function such that $|V_n| \geq f(n)$ for all n. Let H be a graph and let $|V_H|=m$. Since f is unbounded and monotone, there exists an N such that $f(n)>m$ for all $n>N$. Test all G_n with $n \leq N$ for isomorphism with H. If some one of these G_i is isomorphic to H, then H is isomorphic to a member of $G(\Gamma)$; if not, then H is not isomorphic to a member of $G(\Gamma)$.

In the same way that conventional grammars form the Chomsky Hierarchy, the graph families generated by Type 0, Type 1, Type 2 and Type 3 edge grammars also form a hierarchy. In the next section, we describe the relationship between the edge grammar hierarchy and the conventional Chomsky Hierarchy.

Section 3: Edge Grammars and the Chomsky Hierarchy

To compare the edge grammar hierarchy to the Chomsky hierarchy, we need to compare the same types of objects. Conventional (Chomsky) grammars generate languages whose members are single strings. The language of an edge grammar Γ is a graph family $G(\Gamma)=\{G_n\}$ each of whose graphs G_n consists of a set of edges E_n and their length n incident vertices V_n. We can compare the edge grammar and Chomsky hierarchies by comparing the vertex sets of graphs in the edge grammar hierarchy with languages in the Chomsky hierarchy.

Let Γ be an edge grammar and let $V(\Gamma)$ denote the set of all vertex labels in the graphs of $G(\Gamma)$, i.e. $V(\Gamma) = \cup_n V_n$ where for each n, V_n is the vertex set of G_n. For X=0,1,2 or 3, let **VX** $= \{V(\Gamma) \mid \Gamma$ is a Type X edge grammar$\}$. In other words, we let VX denote the class of all vertex sets for Type X edge grammars. Let **(D)LX** be the class of (deterministic) languages generated by Type X Chomsky grammars. The following theorem relates the vertex sets VX of the edge grammar hierarchy with the languages LX of the Chomsky hierarchy.

Hierarchy Theorem

a) $L3 \subset V3 \subset L2 \subset V2 \subsetneq V1 = L1 \subset V0 = L0.$

b) V3 is incomparable with DL2.

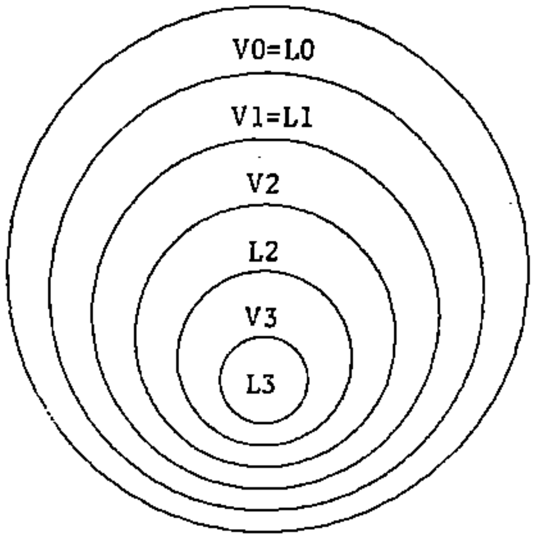


Figure 2
Relationship of the Chomsky and edge grammar hierarchies.

Proof:

Berman in [B] showed L3 $\subset$ V3, L2 $\subset$ V2, V1 $\equiv$ L1, and V0 $\equiv$ L0. V1 $\subset$ V0 follows from the demonstration in [HU] that L1 $\subset$ L0. V2 $\subseteq$ V1 is true by definition.

To show V3$\subset$L2, let Γ be a Type 3 edge grammar. Assume without loss of generality that for every production $A \rightarrow (a,b)B$ in Γ, $|a|=0$ or 1 and $|b|=0$ or 1. We will construct a PDA M which accepts vertices in V_Γ. M takes as input a string w and nondeterministically chooses a derivation of w as the left coordinate of a terminal pair or as the right coordinate of a terminal pair. During the derivation, the stack is used to keep track of the disparity in lengths between the left and right coordinate strings of the currently derived terminal pair. More specifically, let Γ be an edge grammar. Construct a PDA M with the following productions:

$(S,\lambda,\lambda) \rightarrow (G_L,\lambda)$
$(S,\lambda,\lambda) \rightarrow (G_R,\lambda)$
$(A_L,a,y) \rightarrow (B_L,x)$, $(A_R,b,y) \rightarrow (B_R,x)$
where $A \rightarrow (a,b)B$ is a production in Γ and

if $|a|=0$, $|b|=1$, and $y=+$, then $x=++$
if $|a|=0$, $|b|=1$, and $y=-$, then $x=\lambda$
if $|a|=1$, $|b|=0$, and $y=+$, then $x=\lambda$
if $|a|=1$, $|b|=0$, and $y=-$, then $x=--$
if $|a|=|b|$, then $x=y$

$(A_L,a,y) \rightarrow (f,x)$, $(A_R,b,y) \rightarrow (f,x)$
where $A \rightarrow (a,b)$ is a production in Γ and x and y are as given above.

It is straightforward to show that M nondeterministically simulates the derivation of a string as the right or left coordinate in a terminal pair and accepts with an empty stack and in the final state only those strings in V_Γ. Proper containment of V3 in L2 comes from showing that V3 and DL2 are incomparable.

To show that V3 and DL2 are incomparable, we show that V3 is not contained in DL2, and DL2 is not contained in V3 by techniques similar to those found in [F]. To show that V3 is not contained in DL2, consider the language

$$L=\{a^n b^m a^{2n} b^{2m} \mid n,m>0\}.$$

By the pumping lemma for context-free languages, it is straightforward to show that L^- is not in L2. However, L^- is in V3. To see this, observe that $L^-=L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5$ where

$L_1=\{a,b\}^*-a^+b^+a^+b^+$
$L_2=\{a^n b^+ a^m b^+ \mid 2n<m,\ n>0\}$
$L_3=\{a^n b^+ a^m b^+ \mid 2n>m,\ m>0\}$
$L_4=\{a^+ b^n a^+ b^m \mid 2n<m,\ n>0\}$
$L_5=\{a^+ b^n a^+ b^m \mid 2n>m,\ m>0\}$

L_1 is in L3 and hence in V3. L_2 is contained in the vertex set $V(\Gamma)$ generated by the following Type 3 edge grammar. (The other strings in $V(\Gamma)$ are in L_1).

$G \rightarrow (\lambda,a)G'$	$G' \rightarrow (\lambda,a)G'$
$G' \rightarrow (a,aa)A$	$A \rightarrow (a,aa)A$
$A \rightarrow (b,aa)B$	$B \rightarrow (b,a)B$
$B \rightarrow (aa,a)C$	$C \rightarrow (aa,a)C$

$C \rightarrow (aa,a)D \qquad D \rightarrow (b,a)D$
$D \rightarrow (b,a)$

By slightly modifying the edge grammar given above, it is easy to construct Type 3 edge grammars whose vertex sets contain L_3, L_4, L_5 and parts of L_1. Since V3 is closed under union, $L^- = UL_i$ is in V3. Assume towards a contradiction that L^- is also in DL2. Then the complement of L^-, L, must also be in DL2. But L is not even in L2. Hence L^- cannot be in DL2, and V3 cannot be a subset of DL2.

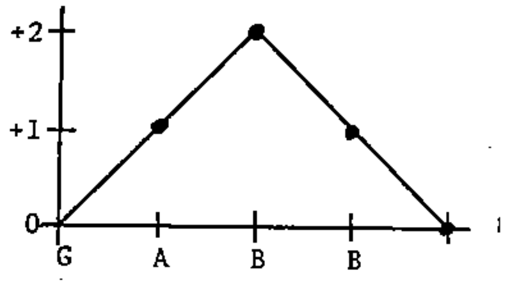
We next show that DL2 cannot be contained in V3. Let $L = \{a^n b^n\}$. L is in DL2. We show that L is not in V3. Assume towards a contradiction that Γ is a Type 3 edge grammar whose vertex set is L. Then all terminal pairs of Γ must be self-loops of the form $(a^n b^n, a^n b^n)$. We show that L is not in V3 by showing that any "long-enough" derivation can eventually pump the left and right coordinates producing strings not in L. Let N be the number of nonterminals in Γ. Consider a derivation of the terminal pair $(a^M b^M, a^M, b^M)$ where $M=N^2$.

Assume the derivation contains a loop $A \rightarrow {}^*(\alpha_2,\beta_2)A$ whose net coördinate difference is zero ($|\alpha_2|=|\beta_2|$) i.e.

$$G \rightarrow {}^*(\alpha_1,\beta_1)A \rightarrow {}^*(\alpha_1\alpha_2,\beta_1\beta_2)A \rightarrow {}^*(\alpha_1\alpha_2\alpha_3,\beta_1\beta_2\beta_3)=(a^N b^N, a^N b^N).$$

Then this loop can be pumped in the same way looping derivations in regular languages can be pumped. In our context, the result will be a terminal pair (w,w) such that w is in $V(\Gamma)$ but not in L. This contradicts the assumption that L is the vertex set of a Type 3 edge grammar.

Now assume that the derivation of $(a^M b^M, a^M b^M)$ contains no loop whose net coordinate difference is zero. The derivation can be visualized as a graph in which the horizontal axis is labelled by the nonterminals at each step of the derivation sequence with the start symbol at the origin (Figure 3). The vertical axis gives the net difference $|x|-|y|$ of the left and right coordinates of the currently derived pair (x,y). For the final terminal pair to be in $V(\Gamma)$, the graph must start and stop with difference 0. We can assume without loss of generality that after each application of a production (at each nonterminal along the horizontal axis), the height of the graph changes by +1, -1 or 0.



$G \rightarrow (a,\lambda)A \rightarrow (ab,\lambda)B \rightarrow (ab,a)B \rightarrow (ab,ab)$

Figure 3
A derivation and the graph of its net coordinate differences.

Consider the first loop $A \rightarrow {}^*(x_2,y_2)A$ in the derivation of $(a^M b^M, a^M b^M)$. Assume without loss of generality that the difference in the x and y coordinates is positive (the case is analogous if the first loop is negative). Since the derivation has length N^2, there are no zero-difference loops and the net difference must end up to be zero, there must be a negative loop $B \rightarrow {}^*(x_4,y_4)B$ in the derivation somewhere after the positive loop. We can then rewrite the derivation of $(a^M b^M, a^M b^M)$ as follows

$$G \rightarrow {}^{*}(x_1,y_1)A \rightarrow {}^{*}(x_1x_2,y_1y_2)A$$
$$\rightarrow {}^{*}(x_1x_2x_3,y_1y_2y_3)B \rightarrow {}^{*}(x_1x_2x_3x_4,y_1y_2y_3y_4)B \rightarrow {}^{*}(a^Mb^M,a^Mb^M)$$

where $(a^Mb^M,a^Mb^M)=(x_1x_2x_3x_4x_5\ ,\ y_1y_2y_3y_4y_5)$. Let $M=|x_2|-|y_2|$ and $Q=|x_4|-|y_4|$. By assumption, $M>0$ and $Q<0$. Let n be a positive integer. Consider the pair

$$(w_1,w_2)=(x_1x_2(x_2)^{|Q|n}x_3x_4(x_4)^{Mn}x_5\ ,\ y_1y_2(y_2)^{|Q|n}y_3y_4(y_4)^{Mn}y_5).$$

Note that $|w_1|=|w_2|$ and the derivation for (a^Mb^M,a^Mb^M) can be pumped to yield a derivation for (w_1,w_2). Hence (w_1,w_2) is in $V(\Gamma)$. But w_1 and w_2 are not in L since by pumping, there are either too many alternations of a's and b's or the a's and b's are unbalanced. Hence DL2 cannot be contained in V3.

Summary

Edge grammars were first introduced as an automatable representation for graph families in the context of parallel computation. They have proved useful not only for this application but also as a general system for generating many commonly used graphs and graph families which cannot be defined by existing graph grammars and graph generating systems.

In this paper, we have developed a solid context for edge grammars by relating them to conventional formal language theory and graph grammars. We explored the decidability of determining if a given graph is isomorphic to a member or a subgraph of a member of a given edge grammar. By showing that families of graphs generated by NLC grammars can also be generated by edge grammars, we showed that the membership problem, planarity, connectedness and other problems were undecidable for edge grammars. However when additional constraints are made on edge grammars (e.g. monotonicity), many of these problems have decision procedures.

In Section 3, we related edge grammars to conventional formal languages. We showed that the classes of vertex sets in the edge grammar hierarchy fit within the language classes of the Chomsky hierarchy in a "sawtoothed" fashion. This is illustrated in Figure 2.

Acknowledgements

We would like to thank Art Sorkin for his interest and encouragement in this work.

Bibliography

[B] Berman, F., "Edge Grammars and Parallel Computation," Proceedings of the 1983 Allerton Conference, Urbana, Illinois.

[BS] Berman, F. and L. Snyder, "On Mapping Parallel Algorithms into Parallel Architectures," Proceedings of the 1984 International Parallel Processing Conference, Bellaire, Michigan.

[CER] Claus, V., Ehrig, H. and G. Rozenberg, Lecture Notes in Computer Science 73: *Graph Grammars and Their Application to Computer Science and Biology*, Springer-Verlag, 1979.

[ENR] Ehrig, H., Nagle, M. and G. Rozenberg, Lecture Notes in Computer Science 153: *Graph Grammars and Their Application to Computer Science -- 2nd International Workshop*, Springer-Verlag, 1983.

[F] Fischer, P., "Turing Machines with Restricted Memory Access," Information and Control 9, p. 364-370, 1966.

[HU] Hopcroft, J. and J. Ullman, *Introduction to Automata, Theory, Languages and Computation*, Addison-Wesley, 1979.

[JR] Janssens, D. and G. Rozenberg, "Decision Problems for Node Label Controlled Graph Grammars," JCSS 22, p. 144-177, 1981.

[P] Parker, D., "Notes on Shuffle-Exchange Type Switching Networks," IEEE Transactions on Computers C-29, p. 213-222, March, 1980.

[S] Stone, H., "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers C-20, p. 153-161, February, 1971.