Edge Grammars: Decidability Results and Formal Language Issues

Francine Berman

Gregory Shannon

Report Number:
84-489
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 define 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), \(G\) is a 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 \((NUT)^*\) 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 $\Gamma$ be an edge grammar. The $n$th graph generated by $\Gamma$, $G_n$, is the undirected graph with vertices

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

and edges

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

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

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

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 $S_{Ek}$ 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 $S_{E1}$, $S_{E2}$ and $S_{E3}$ are shown in Figure 1.

![Shuffle-exchange graphs SE1, SE2 and SE3.](image)

**Figure 1**

Shuffle-exchange graphs $S_{E1}$, $S_{E2}$ and $S_{E3}$.

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

<table>
<thead>
<tr>
<th>Production</th>
<th>Nonterminals</th>
<th>Terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G \rightarrow E$</td>
<td>$G \rightarrow S$</td>
<td>$G \rightarrow S$</td>
</tr>
<tr>
<td>$E \rightarrow (0,0)E$</td>
<td>$S_0 \rightarrow (\lambda)N_0$</td>
<td>$S_0 \rightarrow (\lambda)N_0$</td>
</tr>
<tr>
<td>$E \rightarrow (1,0)E$</td>
<td>$N_0 \rightarrow (0,0)N_0$</td>
<td>$N_0 \rightarrow (0,0)N_0$</td>
</tr>
<tr>
<td>$E \rightarrow (0,1)E$</td>
<td>$N_0 \rightarrow (1,1)N_0$</td>
<td>$N_0 \rightarrow (1,1)N_0$</td>
</tr>
<tr>
<td>$E \rightarrow (1,1)E$</td>
<td>$N_0 \rightarrow (\lambda,0)$</td>
<td>$N_0 \rightarrow (\lambda,0)$</td>
</tr>
</tbody>
</table>
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:

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

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

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 develop 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 $\Gamma$, is $H$ isomorphic to a member of the graph family $G(\Gamma)$?

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 $\Gamma$. For example, if the graph vertex sets $V_\Gamma$ of $\Gamma$ are bounded from below by an unbounded monotone increasing function $f$, 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 $\Gamma$, is $H$ isomorphic to a member of $G(\Gamma)$?

2) Given a graph $H$ and a Type 3 edge grammar $\Gamma$, is $H$ isomorphic to a subgraph of any member of $G(\Gamma)$?

3) Given an edge grammar $\Gamma$, are the graphs in $G(\Gamma)$ planar? connected? hamiltonian?

4) Given edge grammars $\Gamma_1$ and $\Gamma_2$, 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 $\Gamma$, is $G(\Gamma)$ finite? infinite? empty?
6) If \( H \) is a graph and there exists a monotone, unbounded function \( f \) with 
\[ |V_n| = f(n) \] 
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 \( \Gamma \) 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:

\[
\#\text{counter}\#\text{graph}\#\text{work area}\#
\]

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 looks like

\[
\#\text{counter}\#\text{graph}\#
\]
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.

\[ \text{#counter#graph#}((v,l_v),(w,l_w)) \]

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

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

Finally, M replaces the current contents of the tape with the edge \((x,y)\) where \(|x|=|y|\), \(x=1^0 N^{-v}\), and \(y=1^0 N^{-w}\). 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 \(\Gamma\) 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 [IR] and hence by our construction, for edge grammars.

To prove 2), we first show that the question "Given a Type 3 edge grammar \(\Gamma\), does any member \(G_i\) 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 \(\Gamma\) with productions

\[
\begin{align*}
G &\to (u_i,v_i)G \\
G &\to (u_{i+1},v_{i+1})
\end{align*}
\]

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 \(\Gamma\) 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 \(\Gamma\), the vertex set \(V_\Gamma\) of \(\Gamma\) 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 \(V_\Gamma\). 5) follows from observing that if \(V_\Gamma\) is finite, infinite or empty, then \(G(\Gamma)\) must also be finite, infinite or empty. (Note that even if \(V_\Gamma\) 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 \(|V_\Gamma|=f(n)\) 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 \(G_i\) with \(n=N\) 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 $\Gamma$ 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 $\Gamma$ be an edge grammar and let $V(\Gamma)$ denote the set of all vertex labels in the graphs of $G(\Gamma)$, i.e., $V(\Gamma) = \cup V_n$ where for each $n$, $V_n$ is the vertex set of $G_n$. For $X=0,1,2$ or 3, let $V_X = \{V(\Gamma) : \Gamma$ is a Type X edge grammar}. In other words, we let $V_X$ denote the class of all vertex sets for Type $X$ edge grammars. Let $(D)L_X be the class of (deterministic) languages generated by Type $X$ Chomsky grammars. The following theorem relates the vertex sets $V_X$ of the edge grammar hierarchy with the languages $L_X$ of the Chomsky hierarchy.

Hierarchy Theorem
a) $L_3 \subseteq V_3 \subseteq L_2 \subseteq V_2 \subseteq V_1 = L_1 \subseteq V_0 = L_0$.
b) $V_3$ is incomparable with $(D)L_2$.

Figure 2
Relationship of the Chomsky and edge grammar hierarchies.
Proof:

Berman in [B] showed \( L_3 \subseteq V_3, L_2 \subseteq V_2, V_1 = L_1, \) and \( V_0 = L_0. \) \( V_1 \subseteq V_0 \) follows from the demonstration in [HU] that \( L_1 \subseteq L_0. \) \( V_2 \subseteq V_1 \) is true by definition.

To show \( V_3 \subseteq L_2, \) let \( \Gamma \) be a Type 3 edge grammar. Assume without loss of generality that for every production \( A \rightarrow (a,b)B \) in \( \Gamma, \) \( |a|=0 \) or \( 1 \) and \( |b|=0 \) or \( 1. \) We will construct a PDA \( M \) which accepts vertices in \( V_\Gamma. \) \( M \) 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 \( \Gamma \) be an edge grammar. Construct a PDA \( M \) with the following productions:

\[
(S,\lambda,\lambda) \rightarrow (G_L,\lambda)
\]
\[
(S,\lambda,\lambda) \rightarrow (G_R,\lambda)
\]
\[
(A_L,a,y) \rightarrow (B_L,x), \quad (A_R,b,y) \rightarrow (B_R,x)
\]

where \( A \rightarrow (a,b)B \) is a production in \( \Gamma \) 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 \( x=y \)

\[
(A_L,a,y) \rightarrow (f,x), \quad (A_R,b,y) \rightarrow (f,x)
\]

where \( A \rightarrow (a,b) \) is a production in \( \Gamma \) 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 \( V_\Gamma. \) Proper containment of \( V_3 \) in \( L_2 \) comes from showing that \( V_3 \) and \( DL_2 \) are incomparable.

To show that \( V_3 \) and \( DL_2 \) are incomparable, we show that \( V_3 \) is not contained in \( DL_2, \) and \( DL_2 \) is not contained in \( V_3 \) by techniques similar to those found in [F].

To show that \( V_3 \) is not contained in \( DL_2, \) consider the language

\[
L=\{a^n b^m a^{2n} b^m : n, m > 0\}
\]

By the pumping lemma for context-free languages, it is straightforward to show that \( L \) is not in \( L_2. \) However, \( L^- \) is in \( V_3. \) To see this, observe that \( L^- = L_1 U L_2 U L_3 U L_4 U L_5 \) where

\[
L_1 = \{a^n b^m a^{2n} b^m : n, m > 0\}
\]
\[
L_2 = \{a^n b^m a^{2n} b^m : 2n < m, n > 0\}
\]
\[
L_3 = \{a^n b^m a^{2n} b^m : 2n > m, m > 0\}
\]
\[
L_4 = \{a^n b^m a^{2n} b^m : 2n < m, n > 0\}
\]
\[
L_5 = \{a^n b^m a^{2n} b^m : 2n > m, m > 0\}
\]

\( L_1 \) is in \( L_3 \) and hence in \( V_3. \) \( L_2 \) is contained in the vertex set \( V(\Gamma') \) generated by the following Type 3 edge grammar. \( \) (The other strings in \( V(\Gamma') \) are in \( L_1).\)

\[
G \rightarrow (\lambda,a)G' \quad G' \rightarrow (\lambda,a)G' \\
G' \rightarrow (a,aa)A \quad A \rightarrow (a,aa)A \\
B \rightarrow (b,aa)B \quad B \rightarrow (b,aa)B \\
B \rightarrow (aa,a)C \quad C \rightarrow (aa,a)C
\]
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 $L_7$. Since $V_3$ is closed under union, $L^- = U L_3$ is in $V_3$. Assume towards a contradiction that $L^-$ is also in $DL_2$. Then the complement of $L^-$, $L$, must also be in $DL_2$. But $L$ is not even in $L_2$. Hence $L^-$ cannot be in $DL_2$, and $V_3$ cannot be a subset of $DL_2$.

We next show that $DL_2$ cannot be contained in $V_3$. Let $L = \{a^n b^n\}$. $L$ is in $DL_2$. We show that $L$ is not in $V_3$. Assume towards a contradiction that $L$ is the vertex set of a Type 3 edge grammar. Then all terminal pairs of $L$ must be self-loops of the form $(a^n b^n, a^n b^n)$. We show that $L$ is not in $V_3$ 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 $\Gamma$. Consider a derivation of the terminal pair $(a^n b^n, a^n b^n)$ where $M = N$.

Assume the derivation contains a loop $A - (a_1, \beta_1)A$ whose net coordinate difference is zero ($|a_1| = |\beta_1|$) i.e.

$$G - (a_1, \beta_1)A - (a_1 a_2, \beta_1 \beta_2)A - (a_1 a_2 a_3, \beta_1 \beta_2 \beta_3) = (a^N b^N, a^N b^N).$$

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 $(a^N b^N, a^N b^N)$ 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.

Consider the first loop $A - (x_2, y_2)A$ in the derivation of $(a^N b^N, a^N b^N)$. 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 $B - (x_4, y_4)B$ in the derivation somewhere after the positive loop. We can then rewrite the derivation of $(a^M b^N, a^M b^N)$ as

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

Figure 3

A derivation and the graph of its net coordinate differences.
follows

\[ G \Rightarrow \sigma(x_1y_1A) \Rightarrow \sigma(x_1x_2y_2y_3) \]

\[ \Rightarrow \sigma(x_1x_2x_3y_4y_5)B \Rightarrow \sigma(a^M b^M, a^M b^M) \]

where \((a^M b^M, a^M b^M) = (x_1, x_2, x_3, x_4, x_5, y_1, y_2, y_3, y_4, y_5)\). Let \(M = |x_1| - |y_1|\) and \(Q = |x_4| - |y_4|\).

By assumption, \(M > 0\) and \(Q < 0\). Let \(n\) be a positive integer. Consider the pair

\[(w_1, w_2) = (x_1x_2(x_2)^Q | x_3x_4(y_4)^M x_5, y_2y_3y_4(y_4)^M y_5).\]

Note that \(|w_1| = |w_2|\) and the derivation for \((a^M b^M, a^M b^M)\) 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 proven 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


