1985

Representing Graph Families with Edge Grammars

Francine Berman

Gregory Shannon

Report Number:
85-517

REPRESENTING GRAPH FAMILIES WITH EDGE GRAMMARS

Francine Berman
Gregory Shannon

CSD-TR-517
May 1985
Revised December 1986

Representing Graph Families with Edge Grammars

Francine Berman[*]
Department of Electrical Engineering and Computer Sciences
University of California at San Diego
La Jolla, California 92093

Gregory Shannon[†]
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

Abstract

An *edge grammar* is a formal mechanism for representing families of related graphs (binary trees, hypercubes, meshes, etc.). Given an edge grammar, larger graphs in the family are derived from simple basis graphs using edge rewriting rules. A drawback to many graph grammars is that they cannot represent some important, highly regular graph families such as the family of shuffle-exchange graphs. Edge grammars, however, exist for all "computable" graph families, and simple edge grammars exist for most regular graph families. In this paper, we define and illustrate edge grammars and analyze them in the context of formal language theory. Our results include hierarchy and decidability properties. Since this work originally was motivated by a need to represent graph families found in parallel computation, the application of edge grammars in this context is also discussed.

[*]Supported by the Office of Naval Research Contract No. N00014-86-K-0218.

[†]Supported in part by Hewlett-Packard's Faculty Development Program.

1 Introduction

A graph "family" is a set of graphs which share structural and/or vertex labeling properties, e.g. the family of complete binary trees. Graph families are fundamental in computer science. They are used to represent data structures and algorithms [1], architectures [17,22,24], data flow [18], automata [12], etc.

Since the relationships between the graphs in a graph family are abstract, we need a formalism with which to define and manipulate graph families. In the long run, one desires that the formalization be powerful enough to provide a model for graph related problems in the same way that the formalization of type 2 languages as context-free grammars provides a model for the lexical and parsing phases of compiler construction.

Past work on formalizing graph families has centered on graph grammars or systems which use subgraph rewrite rules to generate new graphs from previous or basis graphs. This work includes Lindenmayer systems [20], graph grammars [8,9], pair grammars [16], NLC grammars [13,14], web grammars [19], and others. A drawback to these grammars is that they cannot express all "computable" graph families.[1] For example, many graph grammars have no representation for the highly regular family of shuffle-exchange graphs.

The original motivation for edge grammars comes from the problem of mapping parallel algorithms into parallel architectures. Regular graphs such as the shuffle-exchange are commonly used in parallel computation [22]. Therefore, previous graph type grammars and systems are unsatisfactory since they lack the necessary power to represent graph families needed in this application. Edge grammars solve the problem of efficiently and clearly representing the shuffle-exchange family and other highly regular graph families. More generally, edge grammars can represent any "computable" graph family.

Edge grammars were introduced in [2] in the context of a graph generating formalism for parallel computation. In [4] we discussed decidability and hierarchy results for edge grammars. In [3], edge grammars are applied to the mapping problem in parallel computation. This paper unifies and expands the edge grammar results in these three papers.

Section 2 provides the basic definitions for edge grammars. We give example edge grammars which represent the graph families of complete binary trees and shuffle-exchange graphs.

[1] We mean computable in the sense of Church's Thesis; the labels and the structure of a graph are computable.

Section 3 delivers our hierarchy results for edge grammars; we compare the "languages" of edge grammars with the Chomsky languages. Section 4 investigates decidability questions which arise when discussing graph-generating grammars. For example, when can we decide if a graph is isomorphic to a member of an edge grammar's graph family? Section 5 applies the idea of edge grammars to the mapping problem in parallel computation by using edge grammar representations to automatically contract many common parallel communication graphs. In Section 6 we evaluate the completeness of these results on edge grammars and discuss some remaining interesting questions.

2 Definitions

In this section we present the basic definitions needed to discuss edge grammars. We also give edge grammar representations of the family of complete binary trees and the family of shuffle-exchange graphs as examples to illustrate the definitions. There are no definitions provided for the traditional formal language grammars and equivalent machines in this paper. We will freely assume and use the definitions and notations for the Chomsky grammars and languages, push-down-automata (PDA's), linear-bounded-automata (LBA's), and Turing machines (TM's) found in [12]. The Chomsky grammars (languages) are the regular (type 3), context-free (type 2), context-sensitive (type 1), and unrestricted (type 0) grammars (languages).

Definition 2.1 *An* **edge grammar** *is a 4-tuple* $\langle N, T, S, P \rangle$ *where*

N *is a finite set of* **non-terminals.**

$T = \{(v, w) | v$ *and* $w \in \Sigma^*\}$ *is a finite set of* **terminal pairs.**

$S \in N$ *is the* **start symbol.**

$P = \{\alpha \rightarrow \beta \,|\, \alpha, \beta \in (N \cup T)^+$ *is a finite set of* **productions.**

Definition 2.2 *To* **combine** *terminal pairs or pairs produced by previously combined pairs, let* $(v, w)(x, y) = (vx, wy)$.

An edge or pair (v, w) *is* **generated** *in* Γ, $S \Rightarrow^* (v, w)$, *if* $S \Rightarrow^* \alpha$, α *results from* S *by a sequence of legal applications of productions in* P, $\alpha \in T^+$, *and* $\alpha = (v, w)$ *after combining pairs.*

An edge or pair (v,w) *is* **derived** *in* Γ, $S \hookrightarrow^* (v, w)$, *if* $S \Rightarrow^* (v, w)$ *and* $|v| = |w|$.

The empty symbol is ϵ. Therefore, $(v,\epsilon)(\epsilon,w) = (v,w)$. $|x|$ is the length of the string x.

Figure 1 is an example of an edge grammar for the family of complete binary trees. Figure 2 demonstrates how the ideas of generation and derivation in definition 2.2 apply to the complete binary tree grammar in figure 1. These two figures also show how edge grammars are closely related to Chomsky grammars. The only difference between generating edges and deriving strings is in how combined terminals are viewed.

Note that the vertex labels for each edge are **required to be of the same length.** Therefore, we can *index* the graphs Γ produces by the length of the vertex labels in the graphs. This indexing enables us to distinguish between distinct graphs in an edge grammar's graph family.

Definition 2.3 *The* n^{th} **graph** *derived by* Γ, $G_n(\Gamma)$, *is an undirected graph with vertex set* $V_n(\Gamma)$ *and edge set* $E_n(\Gamma)$, *where*

$$V_n(\Gamma) = \{\, v \mid |v| = n, \text{ and } \exists w \; S \hookrightarrow^* (v,w) \text{ or } S \hookrightarrow^* (w,v) \,\}.$$
$$E_n(\Gamma) = \{\, (v,w) \mid v,w \in V_n(\Gamma), \text{ and } S \hookrightarrow^* (v,w),\, v \neq w \,\}.$$

Note that $V_0(\Gamma)$, $E_0(\Gamma)$ and $G_0(\Gamma)$ are well-defined; they are either the empty set or the set $\{\epsilon\}$.

Definition 2.4 *The* **graph family** *derived by* Γ *is*

$$G(\Gamma) = \{\, G_n(\Gamma) \mid n \geq 0 \,\}$$

The **vertex set** *or* **language** *derived by* Γ *is*

$$V(\Gamma) = \{\, v \mid \exists n \geq 0,\, v \in V_n(\Gamma) \,\}.$$

The **edge set** *derived by* Γ *is*

$$E(\Gamma) = \{\, (v,w) \mid \exists n \geq 0,\, (v,w) \in E_n(\Gamma) \,\}.$$

Figure 3 provides examples of the graphs with 1, 2 and 3 character length labels using the complete binary tree edge grammar Γ from figure 1. Figure 3 also displays examples of the graph, vertex and edge sets that an edge grammar produces.

G_n, V_n and E_n[2] demonstrate how the length of the vertex labels defines a graph derived by a grammar: all of G_n's vertex labels have length n which makes G_n's vertex and edge set distinguishable from G_m's, for $m \neq n$.

Definitions 2.5 and 2.6 below provide a classification of edge grammars and their languages analogous to the language classifications in formal language theory. We use this classification in section 3 to compare the languages of edge grammars and Chomsky grammars.

Definition 2.5 *An edge grammar Γ is of a given* **type** *if all of its productions have the correct form for that type as specified below.*

Type 0: *No restrictions.*

Type 1: *There are at least as many non-terminals and terminal pairs on the production's right-hand-side as on the left-hand-side and $(\epsilon, \epsilon) \notin T$.*

Type 2: *$A \rightarrow BC$, or $A \rightarrow (v, w)$, where A, B and C are non-terminals and (v, w) is a terminal pair.*

Type 3: *$A \rightarrow B$, $A \rightarrow (v, w)B$, or $A \rightarrow (v, w)$, where A and B are non-terminals and (v, w) is a terminal pair.*

Definition 2.6 *For edge grammars of type $I = 0, 2, 3$, let the* **class of edge grammar languages of type** *I be*

$$VI = \{ V(\Gamma) \mid \Gamma \text{ is a type } I \text{ edge grammar} \}.$$

For edge grammars of type 1, let the **class of edge grammar languages of type 1** *be*

$$VI = \{ V(\Gamma), V(\Gamma) \cup \{\epsilon\} \mid \Gamma \text{ is a type 1 edge grammar} \}.$$

For Chomsky grammars of type $I = 0, 2, 3$, let the **class of Chomsky languages of type** *I be*

$$LI = \{ L(\Gamma)^3 \mid \Gamma \text{ is a type } I \text{ Chomsky grammar} \}.$$

For Chomsky grammars of type 1, let the **class of Chomsky languages of type 1** *be*

[2] When the context is clear, we will use G_n to represent $G_n(\Gamma)$ and likewise for V_n and E_n.

[3] $L(\Gamma)$ is the language produced by a Chomsky grammar Γ.

$$L1 = \{ L(\Gamma), L(\Gamma) \cup \{\epsilon\} \mid \Gamma \textit{ is a type 1 Chomsky grammar} \}.$$

The usual classes of regular, context-free and unrestricted languages (including languages which include the empty string) are represented as *L3*, *L2*, *L1* and *L0* respectively. The definition of the class of type 1 edge grammar languages *V1* has been augmented so that it comparable with *V2*, likewise for *L1* and *L2*. In addition to the above definitions, *DL2* refers to the class of deterministic context-free languages or equivalently the class of languages accepted by deterministic push-down automata.

From the above definitions we see that the complete binary tree edge grammar Γ from figure 1 is type 3, and $V(\Gamma)$ is in *V3*. The shuffle-exchange edge grammar Θ discussed below and in figure 4, is also type 3, and $V(\Theta)$ is in *V3*.

As indicated earlier, edge grammars can represent clearly the family of shuffle-exchange graphs. A shuffle-exchange graph SE_n consists of 2^n vertices. Each vertex is labeled by a binary n-bit string. There are two types of edges: shuffle edges and exchange edges. On shuffle edges, the incident vertices are left or right circular shifts of one another, e.g. $(1000, 0001)$ and $(0100, 1000)$. For exchange edges, the incident vertices are identical except that the last bit is complemented, e.g. $(1000, 1001)$. The first three shuffle-exchange graphs SE_1, SE_2 and SE_3 are shown in figure 4. The edge grammar Θ in figure 5 generates these graphs. In other words, $G_n(\Theta) = SE_n, n > 0$. See [22] for a more in-depth discussion of shuffle-exchange graphs.

Additional edge grammar examples are in [3] for cube-connected cycles, linear arrays (lines), meshes, and binary n-cubes.

In the next section, we compare the "power" of edge grammar languages with themselves and the Chomsky languages. This results in a hierarchy of the Chomsky and edge grammar language classes.

3 A Hierarchy of Language Classes

In this section, we prove a hierarchy theorem for Chomsky and edge grammar languages. Figure 6 pictorially presents this theorem. The work to prove this theorem is divided into three smaller theorems. The hierarchy theorem follows directly from the results of these three theorems. Theorem 3.1 presents the relationships of edge grammar languages with themselves and Chomsky languages with themselves. Theorem 3.2 shows how *VI* is related to *LI* for $I = 0, 1, 2, 3$. Theorem 3.3 cleans up the interelationships in the lower half of the hierarchy (*V3*, *V2*, *L3*, *L2*, and *DL2*).

Hierarchy Theorem

$L3 \subset V3 \subset L2 \subset V2 \subseteq L1 \equiv V1 \subset L0 \equiv V0$,
V3 is incomparable with DL2,
$L3 \subset (DL2 \cap V3)$.

Theorem 3.1

$L3 \subset DL2 \subset L2 \subset L1 \subset L0$,
$V3 \subseteq V2 \subseteq V1 \subseteq V0$.

Proof: The proof of the proper containment inter-relationships for the Chomsky languages is in [12] and elsewhere. The containment inter-relationship of the edge grammar languages follows directly from definition 2.5; each edge grammar language class is no more restrictive than the next higher numbered class.

Note that the augmentation of *V1* and *L1* doesn't cause problems. By using grammar normalizations [12], any type 1 grammar can be changed to a grammar with at most one ϵ-production $S \rightarrow (\epsilon, \epsilon)$ where S is the start symbol and does not appear on the right-hand side of any production. If this one production is taken out of the grammar, then the language produced by the grammar does not contain ϵ. Therefore, $V2 \subseteq V1$ and $L2 \subseteq L1$. □

Theorem 3.2

$L3 \subset V3$,
$L2 \subset V2$,
$L1 \equiv V1$,

$L0 \equiv V0$.

To prove theorem 3.2 we first establish lemma 3.1 below. The lemma shows that given any Chomsky grammar, we can effectively construct an edge grammar of the same type which produces the same language as the Chomsky grammar. From this lemma, it follows directly that $LI \subseteq VI$, for $I = 0, 1, 2, 3$.

Lemma 3.1 *Let $\Gamma = \langle N, T, S, P \rangle$ be a type I Chomsky grammar, $I = 0, 1, 2, 3$. Then, there is a type I edge grammar $\Theta = \langle N, T', S, P' \rangle$ where,*

$$T' = \{ (t, t) \mid t \in T \},$$

$$P' = \{ \tau(\alpha) \rightarrow \tau(\beta) \mid \alpha \rightarrow \beta \in P \},$$

where $\tau(A\alpha) = A\tau(\alpha)$, *for* $A \in N$,

and $\tau(a\alpha) = (a, a)\tau(\alpha)$,

$\tau(a) = (a, a)$, *for* $a \in T \cup \{\epsilon\}$,

such that $L(\Gamma) = V(\Theta)$.

Proof: By inspection, there is a derivation of a terminal string w from S in Γ iff there is a similar derivation of a pair (w, w) from S in Θ. This follows since 1) τ^{-1} is well-defined for τ constructed terminals, and 2) for all of the derived pairs in Θ, each of the two vertices in a pair are of the same length. Therefore, $L(\Gamma) \equiv V(\Theta)$. □

The next 4 lemmas address each line of theorem 3.2. Collectively, with lemma 3.1, they readily imply theorem 3.2.

Lemma 3.2 $L3 \neq V3$.

Proof: Let Γ be a *type 3* edge grammar with the productions shown below.

$$S \rightarrow (a, \epsilon)S, \qquad S \rightarrow (a, \epsilon)T,$$
$$T \rightarrow (b, bb)T, \qquad T \rightarrow (b, bb).$$

This edge grammar generates pairs of the form $(a^i b^j, b^{2j})$, $i, j > 0$. Any such pair is an edge in Γ only when $i = j$. Therefore, the language of Γ is $V(\Gamma) = \{a^n b^n | n > 0\} \cup \{b^{2n} | n > 0\}$. By the pumping lemma for Chomsky regular languages [12], $V(\Gamma)$ is not a type 3 Chomsky language. Therefore, $L3$ is not equal to $V3$. □

Lemma 3.3 *L2* $\neq$ *V2*.

Proof: Let Γ be a *type 2* edge grammar with the productions shown below.

$$
\begin{array}{llllllllll}
S & \rightarrow & LR, & & & & & & & \\
L & \rightarrow & AL', & L' & \rightarrow & LB_l, & L & \rightarrow & AB_l, \\
R & \rightarrow & R'C, & R' & \rightarrow & B_rR, & R & \rightarrow & B_rC, \\
A & \rightarrow & (a,a), & & & & & & & \\
B_l & \rightarrow & (b,\epsilon), & B_r & \rightarrow & (\epsilon,b), & & & & \\
C & \rightarrow & (c,c). & & & & & & &
\end{array}
$$

The language of this grammar is $V(\Gamma) = \{a^n b^n c^n | n > 0\}$. To see this, note that $L \Rightarrow^* A^i B_l^i, i \geq 1$ and $R \Rightarrow^* B_r^j C^j, j \geq 1$. Only pairs of the form $(a^i b^i c^j, a^i b^j c^j)$ are generated from S. Therefore, an edge is derived only when $i = j$. By the pumping lemma for Chomsky context-free languages [12], $V(\Gamma)$ is not a type 2 Chomsky language. Therefore, *L2* is not equal to *V2*. □

Lemma 3.4 *V1* $\subseteq$ *L1*.

Proof: For an arbitrary type 1 edge grammar Γ, we construct an LBA which accepts exactly that edge grammar's language.

Let M be a 3-tape nondeterministic Turing machine: one tape for input, another for non-terminals and the left components of terminal pairs, and the third for nonterminals and the right components of terminal pairs. The non-input tapes are called the work tapes. M operates as described below. Without loss of generality, each component of a terminal is restricted to have length one or zero. For example, if a terminal (aaa, b) is required, then it is represented as $(a,b)(a,\epsilon)(a,\epsilon)$.

Begin with Γ's start symbol in the second and third tapes' left-most cells. Nondeterministicly simulate the derivation of an edge pair on the second and third tracks. This is done by repeating the steps below until an appropriate match is found between the input tape and a work tape.

1. If the length of either work tape is more than twice[4] the length of the input tape, then halt with failure.

[4]Twice the length is needed since terminals can have ϵ as one component. This means that an edge with a label of length n might need $2n$ terminals to derive it.

2. Use M's finite state control to nondeterministically find some production's left-hand-side which matches a segment of what is currently on the work tapes. Let $\alpha \rightarrow \beta$ be the matching production. If there is no matching production then halt with failure.

3. Make room on the work tapes to replace α with β. Do this by making room on the second tape for the left-hand components of α and the nonterminals, and by making room on the third tape for the right-hand components of α and the nonterminals. (For example, if $\alpha \rightarrow \beta$ is $(0,1)A(1,0) \rightarrow (0,1)(00,11)(1,0)$, then $0A1$ on the second tape is replaced with 0001 and $1A0$ on the third tape is replaced with 1110).

4. Replace the matched occurrence of the left-hand side α with the right-hand side of β on the work tapes.

5. If there are any non-terminals on the second or third tapes, go to step 1.

6. Compare the second and third tapes to the input to determine if one matches. When comparing, skip occurrences of ϵ. If there is a match *and* the number of symbols on the second tape is equal to the number on the third (again skipping ϵ's), halt with acceptance, else, go to step 1.

Figure 7 shows how M might look if it were partially done simulating a derivation from the edge grammar given in lemma 3.3.

M is not, strictly speaking, an LBA. However, using standard compaction techniques from [12], it is easy to simulate M on an LBA M'.

M accepts at least the strings in $V(\Gamma)$ since 1) the productions for Γ are properly applied in M, and 2) the length of a derived string of terminals and nonterminals increases with each application of a production for a type 1 edge grammar. Therefore, a derivation in Γ never uses an intermediate string of terminals and non-terminals of length greater than twice the length of the final derived edge. No extra strings are accepted since only productions from Γ are simulated and the conditions for accepting are consistent with definitions 2.2 and 2.3.

Therefore, $L(M)$[5] $= L(M') = V(\Gamma)$. Every type 1 edge grammar is accepted by some constructible LBA. Therefore, $V1 \subseteq L1$. □

Lemma 3.5 $V0 \subseteq L0$.

[5] $L(M)$ is the language accepted by machine M.

Proof: For an arbitrary type 0 edge grammar Γ, construct a TM M which accepts $V(\Gamma)$ as in the proof of lemma 3.4. However, do not restrict the length of the strings on the work tapes as in step 1 of the algorithm for lemma 3.4. By a discussion similar to that in lemma 3.4, $L(M) = V(\Gamma)$. Note that non-accepting computations may not terminate. Every type 0 edge grammar Γ is accepted by some such constructible TM. Therefore, $V0 \subseteq L0$. □

With theorem 3.2, we now see that edge grammars can represent any computable graph family; this is a good start on a hierarchy. However, the locations of *V3* and *V2* in the hierarchy are unclear. Is *V3* larger than *L2*? The theorem below shows that it is not and that *V3* does not quite fit in with all of the Chomsky languages. Is *V2* as powerful as *V1* and LBAs? This remains an open question and appears to be as difficult as determining if 2-way-PDAs are as powerful as LBAs.

Theorem 3.3

V3 $\subset$ *L2*,

V3 is incomparable with DL2.

L3 $\subset$ (*DL2* $\cap$ *V3*).

The next four lemmas prove this theorem. Lemma 3.6 demonstrates that $V3 \subseteq L2$. Proper containment follows directly after lemmas 3.7 and 3.8 prove that *V3* is incomparable with *DL2*. Lemma 3.9 proves that the intersection of *DL2* and *V3* contains more than just *L3*.

Lemma 3.6 $V3 \subseteq L2$.

Proof: Let $\Gamma = \langle N, T, P, S \rangle$ be a type 3 edge grammar. Assume without loss of generality that for each nonterminal $(a, b) \in T$, $|a|$ is 0 or 1 and $|b|$ is 0 or 1.

We construct a nondeterministic PDA M which accepts exactly the language of Γ, $V(\Gamma)$. M nondeterministically chooses where to look in a derivation of an edge/pair for w. That is, it looks to match w with either the left or right component of the edge. The subscripts on the state symbols indicate which coordinate has been chosen. The stack contains markers to indicate how much longer or shorter the left coordinate is compared to the right coordinate in the pair derived so far. If the superscript on the current state symbol is + (i.e. A_l^+) then the stack markers represent how much longer the left coordinate is compared to the right

coordinate. If the superscript on the current state symbol is $-$ (i.e. A_l^-) then the stack markers represent how much shorter the left coordinate is compared to the right coordinate.

More specifically, we construct a nondeterministic PDA $M = \langle Q, \Sigma, \Delta, \delta, q_0, Z_0, F \rangle$, with respect to Γ, as described below.

$$
\begin{aligned}
Q &= \{A_l^+, A_l^-, A_r^+, A_r^- \mid A \in N\} \cup \{S, f\}, \\
\Sigma &= \{a \mid \exists b \text{ such that } (a,b) \text{ or } (b,a) \in T\}, \\
\Delta &= \{\bullet, Z_0\}, \\
q_0 &= S, \\
F &= \{f\}.
\end{aligned}
$$

Let $\delta(S, \epsilon, Z_0) = \{(S_l^+, Z_0), (S_r^+, Z_0)\}$.

For each $A \in N$, let

$$
\begin{aligned}
&\delta(A_l^+, \epsilon, Z_0) \ni (A_l^-, Z_0), && \delta(A_r^+, \epsilon, Z_0) \ni (A_r^-, Z_0), \\
&\delta(A_l^-, \epsilon, Z_0) \ni (A_l^+, Z_0), && \delta(A_r^-, \epsilon, Z_0) \ni (A_r^+, Z_0).
\end{aligned}
$$

For each production of the form $A \to (a,b)B$ in P, let

$$
\delta(A_l^+, a, Z_0) \ni \begin{cases} (B_l^+, \bullet Z_0), & \text{if } |a| > |b| \\ (B_l^+, Z_0), & \text{if } |a| = |b| \\ (B_l^-, \bullet Z_0), & \text{if } |a| < |b|, \end{cases} \qquad \delta(A_r^+, b, Z_0) \ni \begin{cases} (B_r^+, \bullet Z_0), & \text{if } |a| > |b| \\ (B_r^+, Z_0), & \text{if } |a| = |b| \\ (B_r^-, \bullet Z_0), & \text{if } |a| < |b|, \end{cases}
$$

$$
\delta(A_l^+, a, \bullet) \ni \begin{cases} (B_l^+, \bullet\bullet), & \text{if } |a| > |b| \\ (B_l^+, \bullet), & \text{if } |a| = |b| \\ (B_l^+, \epsilon), & \text{if } |a| < |b|, \end{cases} \qquad \delta(A_r^+, b, \bullet) \ni \begin{cases} (B_r^+, \bullet\bullet), & \text{if } |a| > |b| \\ (B_r^+, \bullet), & \text{if } |a| = |b| \\ (B_r^+, \epsilon), & \text{if } |a| < |b|, \end{cases}
$$

$$
\delta(A_l^-, a, \bullet) \ni \begin{cases} (B_l^-, \epsilon), & \text{if } |a| > |b| \\ (B_l^-, \bullet), & \text{if } |a| = |b| \\ (B_l^-, \bullet\bullet), & \text{if } |a| < |b|, \end{cases} \qquad \delta(A_r^-, b, \bullet) \ni \begin{cases} (B_r^-, \epsilon), & \text{if } |a| > |b| \\ (B_r^-, \bullet), & \text{if } |a| = |b| \\ (B_r^-, \bullet\bullet), & \text{if } |a| < |b|. \end{cases}
$$

For each production of the form $A \to B$ in P and $x \in \Delta$, let

$$
\begin{aligned}
&\delta(A_l^+, \epsilon, x) \ni (B_l^+, x), && \delta(A_r^+, \epsilon, x) \ni (B_r^+, x), \\
&\delta(A_l^-, \epsilon, x) \ni (B_l^-, x), && \delta(A_r^-, \epsilon, x) \ni (B_r^-, x).
\end{aligned}
$$

For each production of the form $A \to (a,b)$ in P, let

$$
\begin{aligned}
&\delta(A_l^+, a, \bullet Z_0) \ni (f, \epsilon), \text{ if } |a| < |b|, && \delta(A_r^+, b, \bullet Z_0) \ni (f, \epsilon), \text{ if } |a| < |b|, \\
&\delta(A_l^+, a, Z_0) \ni (f, \epsilon), \text{ if } |a| = |b|, && \delta(A_r^+, b, Z_0) \ni (f, \epsilon), \text{ if } |a| = |b|, \\
&\delta(A_l^-, a, \bullet Z_0) \ni (f, \epsilon), \text{ if } |a| > |b|, && \delta(A_r^-, b, \bullet Z_0) \ni (f, \epsilon), \text{ if } |a| > |b|.
\end{aligned}
$$

δ contains only the elements described above. For any undefined state, δ maps it to the empty set. The symbol "$\ni$" is used in defining δ to indicate that M is nondeterministic. That is, δ maps states of the form $Q \times \{\Sigma \cup \{\epsilon\}\} \times \Delta$ into subsets of $Q \times \Delta^*$.

It is straightforward to show that M nondeterministically simulates the derivation of a string as the right or left coordinate in a derivable pair and accepts with an empty stack and in the final state only those strings in the language of Γ, $V(\Gamma)$. Therefore, since Γ was arbitrary, $VS \subseteq L2$. □

VS and *DL2* are shown incomparable by constructing a language for each which is not contained in the other. The techniques used are similar to those found in [10].

Lemma 3.7 *VS* $\not\subseteq$ *DL2*.

Proof: Consider the language $P = \{a^n b^n c^n \mid n > 0\}$. By the pumping lemma for context-free languages, P is not context-free. Since *DL2* is closed under complement [12] and is contained in *L2*, $\overline{P}$ is not in *DL2*. However, as demonstrated below, $\overline{P}$ *is* in *VS*.

Divide $\overline{P}$ into the 5 non-disjoint sets shown below.

$$\overline{P} = P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5,$$

$$P_1 = \{a,b,c\}^* - a^+b^+c^+,$$

$$P_2 = \{a^ib^jc^k \mid i < j,\ i,j,k > 0\} \cup \{a^n | n > 4\},$$

$$P_3 = \{a^ib^jc^k \mid i > j,\ i,j,k > 0\} \cup \{a^n | n > 4\},$$

$$P_4 = \{a^ib^jc^k \mid i < k,\ i,j,k > 0\} \cup \{a^n | n > 4\},$$

$$P_5 = \{a^ib^jc^k \mid i > k,\ i,j,k > 0\} \cup \{a^n | n > 4\}.$$

As shown below, each P_i is expressible as a type 3 edge grammar. The union of the P_i's (P) is also a type 3 edge grammar since *VS* is closed under union. (This closure is easy to show using the method in [12] to show that *LS* is closed under union.)

P_1 is in *LS*, by *LS*'s closure properties, and hence P_1 is in *VS* by theorem 3.2.

P_2 is in *VS* since the language for the type 3 edge grammar below is equal to P_2.

$$\begin{array}{lcllcl}
S & \rightarrow & A, & & & \\
A & \rightarrow & (a,\epsilon)A, & A & \rightarrow & (a,\epsilon)B, \\
B & \rightarrow & (b,aa)B, & B & \rightarrow & (b,aa)B', \\
B' & \rightarrow & (b,a)B', & B' & \rightarrow & (b,a)C, \\
C & \rightarrow & (c,a)C, & C & \rightarrow & (c,a).
\end{array}$$

Note that the B' productions guarantee that $i < j$. This edge grammar is similar to the one used in lemma 3.2. The type 3 edge grammars for P_3, P_4, and P_5 are straightforward to produce from the given edge grammar for P_2.

Note that though the sets P_2, P_3, P_4, and P_5 include extra subsets of $\{a\}^*$, these extra "a" strings have no influence on the union of the P_i's since they are already in P_1.

Hence, there is a type 3 edge grammar for $\overline{P}$. Therefore, since $\overline{P} \in VS$ and $\overline{P} \notin DL2$, VS is not contained in $DL2$. □

Lemma 3.8 $DL2 \not\subseteq VS$.

Proof: An iterated counter [10,11,12] is a PDA with only an "end-of-stack" marker and one other symbol. An iterated counter can accept on either final state or empty stack. Let IC be the class of languages that iterated counters can accept. The PDA constructed in lemma 3.6 is clearly an interated counter. Therefore, VS is also contained in IC.

Fischer states in [10] that $DL2$ and IC are incomparable. Therefore, there is a language Q that is in $DL2$ and not in IC. If $DL2$ were contained in VS, Q would be in VS and therefore in IC. This is a contradiction; hence $DL2$ is not contained in VS. □

Lemma 3.9 $LS \subset (DL2 \cap VS)$.

Proof: The language of the type 3 edge grammar Γ used in the proof of lemma 3.2 is in VS by definition. $V(\Gamma)$ is in $DL2$ since there are no ambiguities. By the pumping lemma for regular languages, $V(\Gamma)$ is not in LS. Therefore, since LS is contained in VS and $DL2$, LS is properly contained in $VS \cap DL2$. □

With the language class hierarchy firmly in hand, we proceed to investigate decidability questions in the next section. We find that the hierarchy established in this section (3) enables us to use decidability results known for Chomsky grammars to help answer decidability questions for edge grammars.

4 Decidability Results

So far we have investigated the structure of edge grammars in terms of the components of the graphs which they generate — edges and vertices. Now, we investigate the global structure produced by edge grammars — $G(\Gamma)$ and its member graphs. The theorem below presents questions about edge grammars which we found to be undecidable.

Undecidability Theorem *The following questions are undecidable for a graph H and type 1 edge grammars Γ and Θ.*

Size: *Is $G(\Gamma)$ empty[6], finite, or infinite[7]?*

Membership: *Is H isomorphic to a member of $G(\Gamma)$?*

Connectivity: *Are all of the graphs in $G(\Gamma)$ connected? planar? hamiltonian?*

Containment: *Is each graph in $G(\Gamma)$ isomorphic to some member of $G(\Theta)$?*

Intersection: *Does there exist at least one graph which is isomorphic to a member of $G(\Theta)$ and to a member of $G(\Gamma)$?*

The proof technique for this theorem is to first reduce the Post Correspondence Problem (PCP) to the question "is $V(\Gamma) = \emptyset$?" (Is $G(\Gamma)$ empty?) Then, we show how to reduce PCP or "is $V(\Gamma) = \emptyset$" to each of the other questions in the theorem. It follows immediately that each question is undecidable since PCP is undecidable.

Definition 4.1 *I is an instance of Post's Correspondence Problem (PCP) of size N over alphabet Σ if $I = \{ (a,b) \mid a,b \in \Sigma^*, (a,b) \neq (\epsilon,\epsilon) \}$ and $|I| = N$.*

We establish the following notation for an arbitrary PCP instance I of size N over Σ. Index each pair in I as (a_i, b_i) where i is between 1 and N. Let $n_i = |a_i|$ and $m_i = |b_i|$. Let a_{ik} be the k^{th} letter of a in the i^{th} (a,b) pair, $1 \leq k \leq n_i$; likewise for b_{ik} with respect to m_i. Let $\mathcal{A}$ be all of the a_{ik}'s and $\mathcal{B}$ be all of the b_{ik}'s.

A pair (x,y) is a solution to I iff $x = y$ and $(x,y) = (a_{s_1}a_{s_2}\ldots a_{s_M}, b_{s_1}b_{s_2}\ldots b_{s_M})$ such that $(a_{s_l}, b_{s_l}) \in I$ for $1 \leq s_l \leq N$ and $1 \leq l \leq M$. That is, a solution is a combination of pairs from I such that the two sides are equal.

[6] "Is $G(\Gamma)$ empty" is the same question as "is $G(\Gamma) = \{(\emptyset,\emptyset)\}$" which is the same as "is $V(\Gamma) = \emptyset$."

[7] Does $G(\Gamma)$ have an (in)finite number of pair-wise non-isomorphic graphs?

Lemma 4.1 *For each PCP instance I of size N over Σ there is a type 1 edge grammar Γ such that there is* no *solution to I iff $V(\Gamma) = \emptyset$.*

Proof: Construct a type 1 edge grammar $\Gamma = \langle N, T, S, P \rangle$ with respect to I as shown below. For each $x, x' \in \mathcal{A}$ and $y, y' \in \mathcal{B}$,

$$
\begin{aligned}
N &= \{S, S', M, H, L, R, A_x, B_y, L_x, R_y, X\} \\
T &= \{(1,\epsilon), (\epsilon,1)\} \\
P &= \{S \rightarrow MS'M, \\
&\quad S' \rightarrow S' A_{a_{i1}} \ldots A_{a_{in_i}} B_{b_{i1}} \ldots B_{b_{im_i}}, \text{ for } (a_i, b_i) \in I, \\
&\quad S' \rightarrow H, \\
&\quad MHA_x \rightarrow (1,\epsilon)ML_x, \qquad MHB_y \rightarrow (1,\epsilon)MR_y, \\
&\quad L_x A_{x'} \rightarrow A_{x'} L_x, \qquad R_y B_{y'} \rightarrow B_{y'} R_y, \\
&\quad L_x B_x \rightarrow LL, \qquad R_y A_y \rightarrow RR, \\
&\quad A_x LL \rightarrow LLA_x, \qquad B_y RR \rightarrow RRB_y, \\
&\quad MLL \rightarrow (\epsilon,1)MH, \qquad MRR \rightarrow (\epsilon,1)MH, \\
&\quad (\epsilon,1)MHM \rightarrow (\epsilon,1)XXX, \\
&\quad X \rightarrow (1,\epsilon)(\epsilon,1)\}
\end{aligned}
$$

The one S production provides two "end-of-tape" markers. The first N S' productions enable Γ to nondeterministically guess a solution to I. If for some i, n_i is zero, then there are no "A" terminals used; likewise for m_i. Note that at least one of n_i and m_i is nonzero for each i.

After the last S' production, $S' \rightarrow H$, the next productions first check to see that what is between the end-of-tape markers is a solution. That is, the right coordinate must equal the left coordinate. These productions also record matched elements of the coordinates as the terminals $(1,\epsilon)(\epsilon,1)$ on the left of the leftmost end-of-tape marker.

The last production is reached only if the solution guessed for I is correct. This production erases the end-of-tape markers and the head. Note that this production is used iff there is a solution to I.

It is easy to see that there is a solution to I of length n iff $S \Rightarrow^* (1^{n+3}, 1^{n+3})$. Therefore, Γ produces the empty language iff I has no solution. □

Now that there is a reduction from PCP to the emptiness of $G(\Gamma)$, we can proceed to prove that the other questions in the Undecidability theorem are reducible to PCP or the emptiness question.

Lemma 4.2 (Size) *For each PCP instance I of size N over Σ there is a type 1 edge grammar Γ such that there is a/no solution to I iff $G(\Gamma)$ is infinite/finite.*

Proof: Based on the edge grammar Γ constructed in the proof of lemma 4.1, construct a type 1 edge grammar $\Theta = \langle N, T', S, P' \rangle$ with respect to I as follows. Let P' contain all of the productions from Γ listed above plus the three discussed below

Add the production $X \rightarrow (1,\epsilon)(\epsilon,1)X$ to P'. This production enables Θ to derive arbitrarily long strings if a state is reached where the production with left-hand-side $(\epsilon,1)MHM$ in Γ could be used.

Add the productions $(1,\epsilon) \rightarrow (0,\epsilon)$ and $(\epsilon,1) \rightarrow (\epsilon,0)$ to P'. These two productions nondeterministicly change 1's to 0's in the final output of a derivation in Θ.

Note that all graphs derived by Θ are isomorphic to a single node if strings can only consist of 1's. However, each Γ_n is isomorphic to the complete graph K_n if strings of length n can consist of all combinations of 1's and 0's.

Given the proof in lemma 4.1, Θ produces an infinite number of complete graphs of increasing size iff there is a solution to PCP. If there is no solution, then Θ produces no graphs. Therefore, $G(\Theta)$ is infinite iff I has a solution. □

Lemma 4.3 (Membership) *For each PCP instance I of size N over Σ, there is a graph $H = \langle V, E \rangle$ and a type 1 edge grammar Γ such that there is a solution to I iff H is isomorphic to some graph in $G(\Gamma)$.*

Proof: This follows almost directly from the constructed Γ in the proof of lemma 4.1. Let $H = \langle \{a\}, \emptyset \rangle$ be the graph consisting of a single node. Each $G_n(\Gamma)$ is defined by those edges and vertices derived by Γ where the vertex labels are of length n. All solutions to I of length m imply exactly one terminal edge in Γ, $(1^{m+3}, 1^{m+3})$. Therefore, each G_n is either isomorphic to H or G_n is the null graph. Therefore, H is isomorphic to a member of $G(\Gamma)$ iff I has a solution. □

Lemma 4.4 (Connectivity) *For each PCP instance I of size N over Σ there is a type 1 edge grammar Γ such that there is no solution to I iff all graphs in $G(\Gamma)$ are connected/planar/hamiltonian.*

Proof: For connectivity, modify the edge grammar Γ from the proof of lemma 4.1 to construct Δ with respect to I. Allow Δ to also produce a string of 0's in the right component the same way that a string of 1's is produced in Γ. Now, a graph of the form $\langle \{1^{n+3}, 0^{n+3}\}, \emptyset \rangle$ is derived using Δ iff there is a solution to I of length n. Since the null graph is vacuously connected, all graphs in $G(\Delta)$ are connected iff I has no solution.

PCP can be reduced to the question of planarity by using the modified edge grammar Θ from the proof of lemma 4.2. If I has one solution, it has infinitely many solutions. For $n > 4$, the complete graph K_n is not planar. Therefore, all graphs in Θ are planar iff I has no solution.

PCP can be reduced to the question of the hamiltonianness of a graph family by using the edge grammar Δ for reducing PCP to the connectivity question from above. It is easy to show that all graphs in Δ are hamiltonian iff I has no solution. □

Lemma 4.5 (Containment and Intersection) *For each graph H and type 1 edge grammar Γ, there is a type 1 edge grammar Θ such that 1) H is isomorphic to a member of $G(\Gamma)$ iff each graph in $G(\Theta)$ is isomorphic to some member of $G(\Gamma)$, and 2) H is isomorphic to a member of $G(\Gamma)$ iff there exists at least one graph which is isomorphic to a member of $G(\Theta)$ and to a member of $G(\Gamma)$,*

Proof: Construct type 1 edge grammar Θ which represents the finite structure of H. Uniquely label each node in H with equal length labels. Generate productions for Θ of the form $S \rightarrow$ "edge" where "edge" is a pair of H's node labels connected by an edge in H. Now, H is "in" $G(\Gamma)$ iff the graph in $G(\Theta)$ is isomorphic to some member of $G(\Gamma)$. This reduces the membership question to the containment question.

To reduce membership to the intersection question, let Θ be as above with respect to H. Now, H is "in" $G(\Gamma)$ iff H is isomorphic to a graph in $G(\Theta) = H$ and isomorphic to a graph in $G(\Gamma)$, and the reduction is complete. □

The combination of the reductions presented in lemmas 4.1 to 4.5 reduce PCP to each question in the Undecidability theorem. Since PCP is undecidable, each Undecidability

theorem question is undecidable.

Even though the membership question is in general undecidable, there still is motivation to determine when membership is decidable. We find that there are reasonable restrictions on edge grammars which make membership decidable. In particular, membership is decidable for many interesting regular graph families including the families of shuffle-exchange graphs, complete binary trees, and meshes.

Decidability Theorem *These questions are decidable for graph H and edge grammar Γ.*

Size: *If Γ is type 3, is $V(\Gamma)$ empty, finite, or infinite?*

Membership: *If Γ is type 1 and $|V_n(\Gamma)|$ is bounded by $f(n)$, a nondecreasing function with no upper bound, then is H isomorphic to a member of $G(\Gamma)$?*

Proof: The decidability of emptiness, finiteness, and infiniteness for type 3 edge grammar languages follows directly from the fact that $V3 \subset L2$ and the decidability of these questions for type 2 Chomsky languages [12].

Decidability for the new membership question is proved with a counting argument. Let H be the graph $\langle V_H, E_H \rangle$ where $|V_H| = m$. Since f is nondecreasing and has no upper bound, there exists an N such that $f(n) > m$ for all $n \geq N$. Test all $G_p(\Gamma)$, $p < N$, for isomorphism with H. We can generate each $G_p(\Gamma)$ since V_p is computable on a LBA (see section 3), and membership of word in a language in $L1$ is decidable [12]. If one of the G_p is isomorphic to f, then f is isomorphic to a member of $G(\Gamma)$; if not, then f is not isomorphic to any member of $G(\Gamma)$. □

In the next section, we apply edge grammars to problem of mapping parallel algorithms into parallel architectures.

5 Edge Grammars and Parallel Computation

Graphs are a natural abstraction for the interconnection architectures of many parallel computers. In addition, graph families can be used to represent problem instances of a parallel algorithm. In the parallel computation literature, graphs and graph families are often used to abstract the implementation of parallel algorithms on parallel architectures [15,23].

Edge grammars were originally introduced in this context as a formal tool for representing and embedding graph families commonly used in parallel computation [2,3]. In particular, Berman and Snyder studied the problem of developing uniform strategies for embedding and multiplexing large-sized parallel algorithms into fixed-size (smaller) parallel machines (the mapping problem) [5]. In studying the mapping problem, edge grammars were developed to define graphs and graph families. The formalism proved particularly fruitful because edge grammars can be used not only to represent the graph families of many commonly used parallel algorithms, but in many cases can also be used to produce an automatic embedding from larger members of a graph family into smaller members. The representation of parallel algorithms with edge grammars then functions as part of a uniform procedure for implementing large-size parallel algorithms on fixed-size parallel machines.

In the mapping strategy described in [3], it was desirable to be able to easily embed large graphs from a graph family into small graphs from the *same* family. This represents the mapping of a large-size parallel algorithm into a parallel machine of the same interconnection architecture. Differences in interconnection structure are then processed in a separate layout step. When such an embedding can be done in a uniform fashion over the whole graph family, we call the family *contractable*. There is a particularly useful subclass of contractable graph families, called *truncatable* graph families, which promote automatic embedding. In the following, we describe these classes and give sufficient conditions under which an edge grammar for a given graph family is truncatable.

Definition 5.1 *Let* $G(\Gamma) = \{G_n(\Gamma) | G_n = \langle V_n(\Gamma), E_n(\Gamma)\rangle\}$ *be the graph family of edge grammar* Γ *and let* k *be a fixed positive integer. Then* $G(\Gamma)$ *is* k-**contractable** *if for each* $n \geq 0$, *there is a mapping* $c : V_{n+k} \rightarrow V_n$ *such that* $\{(c(v), c(w)) \,|\, c(v) \neq c(w),$ *and* $(v, w) \in E_{n+k}\} \subseteq E_n$.

Definition 5.2 *Let* $G = \langle V, E\rangle$ *be a graph whose labels are strings in* Σ^* *and let* k *be a*

fixed integer. Let t_k be the mapping which assigns to each label xw ($x \in \Sigma^*, w \in \Sigma^k$) in V the label x in Σ^*. Then the graph $t_k(G)$ with the vertex set $\{t_k(v) \mid v \in V\}$ and edge set $\{(t_k(v), t_k(w)) \mid t_k(v) \neq t_k(w)$, and $(v,w) \in E\}$ is the k-**truncation** *of G.*

Definition 5.3 *Let k be a fixed positive integer. A graph family $G(\Gamma) = \{G_n | n \geq 0\}$ for edge grammar Γ is* k-**truncatable** *if for each $n \geq 0, t_k(G_{n+k}) \subseteq G_n$. That is, G_n is the k-truncation of G_{n+k}.*

Proposition *If a graph family is k-truncatable, it is k-contractable.*

The proposition follows directly from the definitions.

The converse is not true however; not all k-contractable graph families are k-truncatable. For example, the family of shuffle-exchange graphs $\{SE_n\}$ (see figures 4 and 5) with the usual labeling (i.e. vertex w is adjacent to vertex v if the label of v is a shuffle or the exchange of the label of w) is k-contractable with the trivial contraction which maps every vertex in SE_{n+k} onto a single vertex in SE_n. However, this family is not k-truncatable since $t_k(E_{n+k}) \neq E_n$, for any $k > 0$ and $n > 1$.

Truncation Theorem *Let Γ be a type 3 edge grammar with graph family $G(\Gamma) = \{G_n | n \geq 0\}$ and let k be a fixed integer. If for every terminal pair $(v,w) \in T$, either $|v| = |w| = k$ or $(v,w) = (\epsilon, \epsilon)$, and for each $A \in N$, there is a derivation $A \hookrightarrow^* (\epsilon, \epsilon)$ in Γ, then $G(\Gamma)$ is k-truncatable.*

Proof: To show that $G(\Gamma)$ is k-truncatable, we must show that for every $n \geq 0$, $t_k(G_{n+k}) \subseteq G_n$ for k-truncation t_k. Let $S \hookrightarrow^* (v,w)$ with $|v| = |w| = n + k$. Then v and w are labels in V_{n+k} and if $v \neq w$, (v,w) is an edge in E_{n+k}. There are two cases to this proof; each involve specifying how the last k symbols in the vertices of the edge (v,w) were derived.

For the first case, the last k symbols are derived by a production of the form $H \rightarrow (a,b)$. Since Γ is type 3 and by hypothesis all terminals have length k, there are strings x and y with $v = xa$, $w = yb$, $|x| = |y| = n$ such that $S \hookrightarrow^* (x,y)H \hookrightarrow (x,y)(a,b) = (v,w)$.

For the second case, the last k symbols are derived by a production of the form $H \rightarrow (a,b)J$, where $J \hookrightarrow^* (\epsilon, \epsilon)$. Since Γ is type 3 and by hypothesis all terminals have length k, there are strings x and y with $v = xa$, $w = yb$, $|x| = |y| = n$ such that $S \hookrightarrow^* (x,y)H \hookrightarrow (x,y)(a,b)J \hookrightarrow^* (x,y)(a,b)(\epsilon,\epsilon) = (v,w)$.

Since $H \hookrightarrow^* (\epsilon, \epsilon)$ by hypothesis, we know that $S \hookrightarrow^* (x,y)H \hookrightarrow^* (x,y)(\epsilon,\epsilon) = (x,y)$. If $x \neq y$, then $(x,y) = (t_k(v), t_k(w))$ is in E_n. In any case, x and y are in V_n. Hence, $t_k(G_{n+k})$ is a k-truncation of G_n, and $G(\Gamma)$ is k-truncatable. □

Graph families which are k-truncatable by the truncation theorem include complete binary trees, cube-connected cycles, butterfly networks, square meshes, hypercubes, finite element graphs, toruses, linear arrays, complete graphs and others. (See [2,3] for the examples of some of these edge grammars.) Also note that the theorem provides one effective procedure to determine if a graph family is k-truncatable.

For a parallel algorithm whose graph family $G = \{G_n | n \geq 0\}$ is k-truncatable, there is a uniform algorithm for mapping any large graph G_n in the family into a fixed-size parallel architecture H. The mapping algorithm is given below.

Mapping Algorithm

To implement graph G_n on architecture H:

1. Choose the largest interconnection graph G_m from the family G, $m \leq n$, which can be laid out efficiently on the interconnection architecture H.

2. Truncate G_n to G_m using the edge grammar derivation of G_n. This mapping will specify which processes in G_n must be simulated at each single processor in H during the multiplexing phase.

3. Lay out the contracted graph G_m on H using a good layout heuristic [3] or library routines.

4. Multiplex G_n on H using the truncation specified in step 2 and the layout specified in step 3.

An important feature of this strategy is that the layout problem can be separated from the contraction problem. In particular, solutions to both problems may be independently optimized.

As an example, suppose we wish to execute a parallel algorithm whose interconnection graph is a binary tree with 255 processes on a mesh-connected computer with 49 processors

as in figure 8. An example algorithm might be Schwartz's parallel max-finding algorithm [21] or one for Browning's parallel tree machine [7].

By the Truncation theorem, the family of complete binary trees is k-truncatable for any $k > 0$ using the edge grammar given in figure 1. Hence, we can map the 255 node tree onto a 31 node tree using a 3-truncation t_3. In particular, the 3-truncation maps nodes with labels $a_1 \ldots a_7$ onto nodes with labels $t_3(a_1 \ldots a_7) = a_1 \ldots a_4$ in the 31 node tree. For example, nodes with labels $2222 * **$ are mapped to the node 2222. Therefore, 2222222 and 2222110 are mapped to the same truncated node.

In the next phase of the algorithm, the 31 node tree can be laid out on the 49 node mesh using an H-tree layout [15] (figure 9). Using the assignment of processes in the 255 node tree to nodes in the (contracted) 31 node tree (and hence to processors in the 49 node mesh), the original algorithm can be multiplexed on the fixed-size target architecture.

The result of this procedure is that the large-size parallel algorithm can be run on a fixed-size architecture with the same results as if the architecture was "big enough" to accommodate the algorithm. This mapping procedure using edge-grammar generated contractions appears to generate optimal or near-optimal mappings for many commonly used parallel algorithms and architectures [5].

6 Conclusion and Open Problems

In this paper, we have introduced edge grammars as a formal language mechanism for easily and efficiently describing, producing and manipulating families of graphs.

We have treated edge grammars as a formal language with respect to the definition and derivation of graphs. Viewing the "output" from grammars as a language, we have compared their power to that of the Chomsky grammars. This lead us to the Hierarchy theorem.

Another hierarchical view that would be useful, but that we did not pursue, is that of looking only at the structure of the graph families produced by edge grammars. For example, are there interesting graph family structures which a type 2 edge grammar can produce but not a type 3 grammar? Since all of the graphs that we are interested in for applications are produced by type 3 edge grammars, a structural hierarchy might provide a class of interconnection structures desirable for parallel computation. For example, "type 3" graph structures may have good or easy to find separators, bifurcators, or embeddings [6] whereas

strictly "type 2" structures may not.

In the Undecidability and Decidability theorems we addressed the question of membership. Though in general the decidability news is bad, for interesting applications membership is decidable. The decidability of membership for type 2 and "non-even" type 3 edge grammars is an open question.

A question to which we often thought we had an answer is: is there a pumping-type lemma for type 3 and/or type 2 edge grammars? We believe that useful pumping lemmas for edge grammars must not only show that the length of the derivable strings can be arbitrarily long, but that the size of $V_n(\Gamma)$ is unbounded as n increases. If there were any pumping lemmas of this type, then membership for the "pumpable" grammars would be decidable. (The proof would be similar to the counting proof in the Decidability theorem). A pumping lemma for type 2 edge grammars could also surely be used to show that *V2* is properly contained in *V1*.

The implication of the Undecidability theorem is that questions about the structure of a graph family produced by an edge grammar are at best possibly decidable only for types 2 and 3 edge grammars. The obvious question is, "What structural properties of graph families produced by type 3 (or 2) edge grammars *are* decidable?" Intesting structural qualities to look for might be connectivity, bounded degree, bipartiteness, etc.

The Truncation theorem shows how the edge grammar formal language mechanism is useful in automating a part of one solution to the mapping problem. Given powerful pumping lemmas and more information about truncation and contractability, it might be possible to construct for certain pairs of interesting edge grammars (different pairs of graph families) efficient mappings from one graph family to the other. In addition, effective types of contraction other than truncation could be found.

Acknowledgments

We would like to thank Larry Synder and Mike Atallah for their interest and encouragement in this work.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] F. Berman. Edge grammars and parallel computation. In *Proceedings of the 21 st Annual Allerton Conference on Communication, Control, and Computing*, pages 214–223, 1983.

[3] F. Berman, M. Goodrich, C. Koelbel, W. Robison, and K. Showell. *A guide to the Poker Mapping Preprocessor.* Technical Report CSD-TR-488, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, 1984.

[4] F. Berman and G. Shannon. Edge grammars: decidability results and formal language issues. In *Proceedings of the 22nd Annual Allerton Conference on Communication, Control, and Computing*, pages 921–930, 1984.

[5] F. Berman and L. Snyder. On mapping parallel algorithms into parallel architectures. In *Proceedings of the 1984 International Conference on Parallel Processing*, pages 307–309, 1984.

[6] B. Bhatt and F. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984.

[7] S. Browning. *The tree machine: A highly concurrent programming environment.* PhD thesis, California Institute of Technology, 1980.

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

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

[10] P. Fischer. Turing machines with restricted memory access. *Information and Control*, 9:364–370, 1966.

[11] M. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.

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

[13] D. Janssens and G. Rozenberg. On the structure of node-label controlled graph languages. *Information Sciences*, 20:191–216, 1980.

[14] D. Janssens and G. Rozenberg. Restrictions extensions and variations of NLC grammars. *Information Sciences*, 20:217–244, 1980.

[15] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[16] T. Pratt. Pair grammars graph languages and string-to-graph translations. *Journal of Computer and System Sciences*, 5(6):560–595, 1971.

[17] F. Preparata. *Algorithm design and VLSI architecture.* Technical Report , Coordinated Science Lab, University of Illinois, Urbana, Illinois, 1982.

[18] A. Rosenberg and L. Snyder. Bounds on the costs of data encodings. *Mathematical Systems Theory*, 12:9–39, 1978.

[19] A. Rosenfeld and D. Milgram. Web automata and web grammars. *Machine Intelligence*, 7:307–324, 1972.

[20] A. Salomaa. *Formal Languages.* Academic Press, 1973.

[21] J. Schwartz. Ultracomputers. *ACM Transactions on Programming Languages and Systems*, 2(4):484–521, 1980.

[22] H. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computing*, C-20(2):153–161, February 1971.

[23] J. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[24] L. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computing*, C-30(2):135–140, February 1981.

Figures

$$
\begin{aligned}
\Gamma &= \langle N, T, S, P \rangle \\
N &= \{S, A\} \\
T &= \{(2,2), (2,0), (2,1), (1,1), (0,0), (\epsilon, \epsilon)\} \\
P &= \{S \to (2,2)S, \quad S \to (2,0)A, \quad S \to (2,1)A, \quad S \to A, \\
&\quad\ A \to (0,0)A, \quad A \to (1,1)A, \quad A \to (\epsilon, \epsilon)\}
\end{aligned}
$$

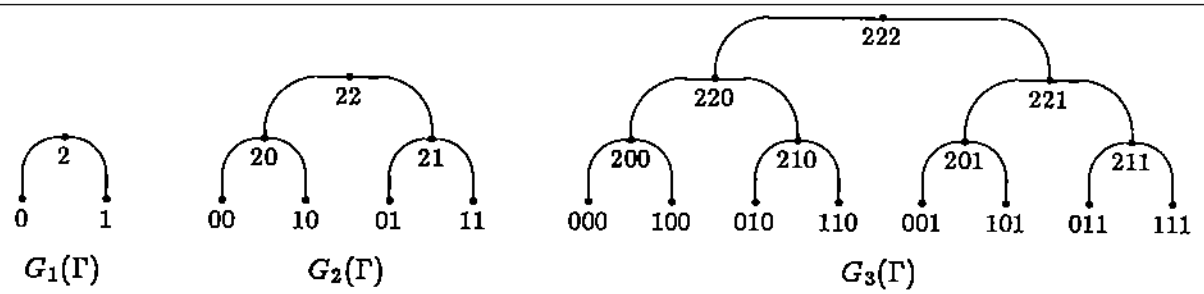
Figure 1: Γ is an edge grammar for the family of complete binary trees.

$$
\begin{aligned}
S &\Rightarrow (2,2)S \\
&\Rightarrow (2,2)(2,0)A \\
&\Rightarrow (2,2)(2,0)(1,1)A \\
&\Rightarrow (2,2)(2,0)(1,1)(\epsilon, \epsilon)
\end{aligned}
$$

Therefore, $S \Rightarrow^* (221, 201)$.

Since, $|221| = |201|$, $S \hookrightarrow^* (221, 201)$.

Figure 2: A derivation of $(221, 201)$ using the edge grammar from figure 1.



$G(\Gamma) = \{G_1(\Gamma), G_2(\Gamma), G_3(\Gamma), \ldots\}$
$V(\Gamma) = \{2, 0, 1, 22, 20, 21, 00, 01, 10, 11, 222, 220, 221, 200, \ldots\}$
$E(\Gamma) = \{(2,0), (2,1), (22,20), (22,21), (20,00), (20,10), (21,01), (21,11), (222,220), \ldots\}$

Figure 3: The graph, vertex and edge sets derived by the edge grammar from figure 1.

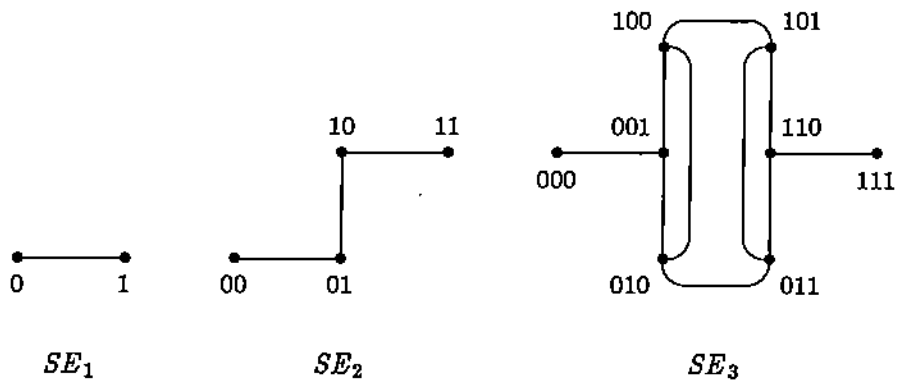


Figure 4: SE_1, SE_2 and SE_3 in the family of shuffle-exchange graphs.

$$\Theta = \langle N, T, S, P \rangle$$

$$N = \{S, E, S_0, S_1\}$$

$$T = \{(0,0),(0,1),(1,1),(\epsilon,0),(0,\epsilon),(\epsilon,1),(1,\epsilon)\}$$

$$\begin{aligned} P = \{ & S \to E, & S \to (0,\epsilon)S_0, & & S \to (1,\epsilon)S_1, \\ & E \to (0,0)E, & E \to (1,1)E, & & E \to (0,1), \\ & S_0 \to (0,0)S_0, & S_0 \to (1,1)S_0, & & S_0 \to (\epsilon,0), \\ & S_1 \to (0,0)S_1, & S_1 \to (1,1)S_1, & & S_1 \to (\epsilon,1)\} \end{aligned}$$

E derives exchange edges.
S_0 derives shuffle edges with a 0 shuffled.
S_1 derives shuffle edges with a 1 shuffled.

Figure 5: Θ is an edge grammar for the family of shuffle-exchange graphs.

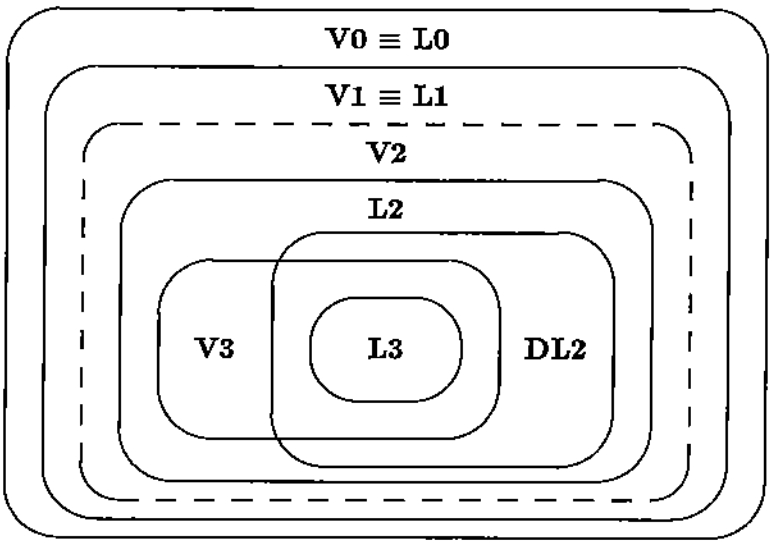


Figure 6: Hierarchical structure of the Chomsky and edge grammar language classes as indicated by the Hierarchy Theorem. The border between *V2* and *V1* is dashed since it is not known if *V2* is properly included in *V1*.

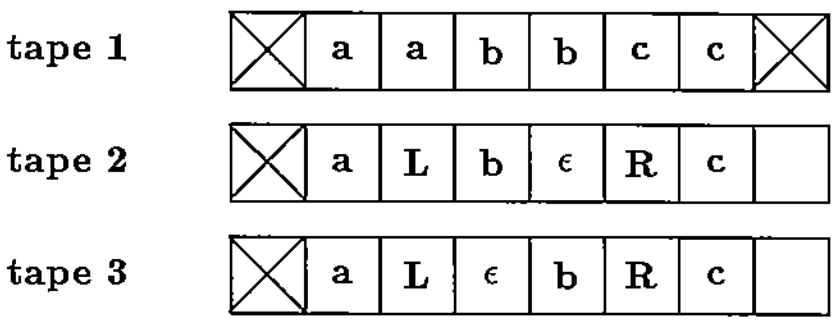


Figure 7: This is a tape state from machine M simulating the derivation $S \Rightarrow^* (a,a)L(b,\epsilon)(\epsilon,b)R(c,c) \Rightarrow^* (aabbcc, aabbcc)$. M is an LBA based on the construction in lemma 3.4 and the edge grammar Γ in lemma 3.3.

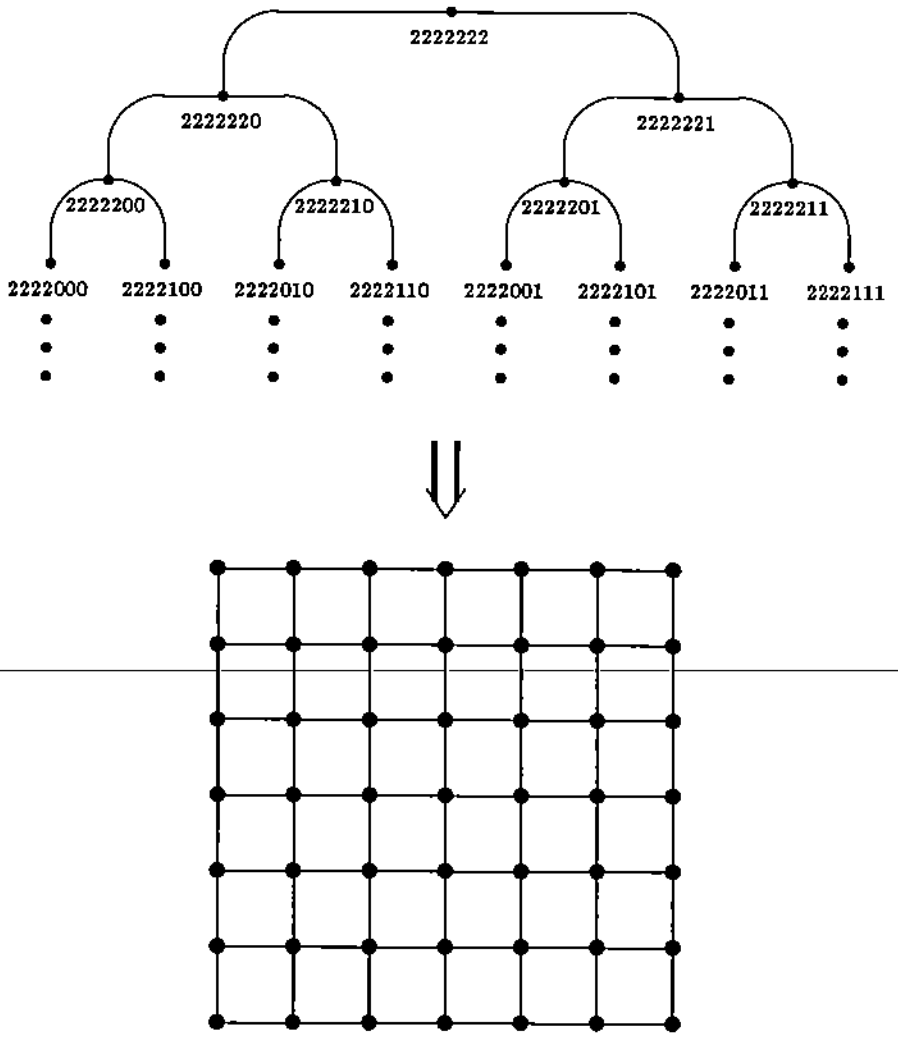


Figure 8: Interconnection structures for a 255 node complete binary tree algorithm and a 49 node square mesh architecture.

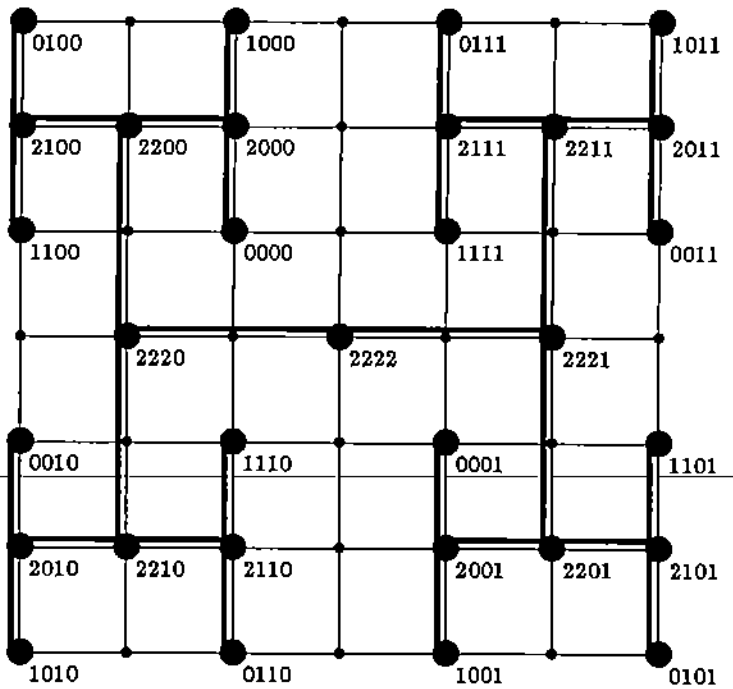


Figure 9: Layout of the 31 node contracted complete binary tree on a 49 node square mesh.