Self-Assembly From the Point of View of Its Pieces

Eric Klavins
Electrical Engineering Department
University of Washington
Seattle, WA 98195
klavins@ee.washington.edu

***Abstract*— A graph grammar can be used to describe or direct the changing connection topology of a collection of, for example, self-organizing robots. Productions in a grammar describe the legal *local interactions* in which the robots may engage and the resulting global structures and processes that form can be analyzed by looking at the set of reachable graphs generated by the grammar. In this paper we discuss the relationship between a grammar and its *characteristic automata set,* which describes the trajectories of the grammar from the "point of view" of the components of the initial graph. The automata set of a grammar is essentially a *Petri-Net* description of these pathways and is related to the grammar as a metabolic network is related to a set of chemical reactions. It is useful for the analysis of the behavior of the grammar.**

I. Introduction

Building systems that self-organize into complex structures and processes presents us with the daunting problem of coordinating vast numbers of objects so that they perform global tasks. Because of the potentially enormous quantities of objects involved, uniquely addressing and manipulating each one is impossible. Nevertheless, there are examples of complex machines, such as the ribosome or the motor in a bacterial flagellum, that seem to assemble spontaneously out of large numbers of simple components. This seems to occur when simple components self-organize via local interactions into more complex aggregates which, in turn, self-organize into larger aggregates and processes. Thus, the primary means to *control or direct* these processes is to engineer the allowable local interactions in such a way that these interactions lead to the desired global properties.

To this end, we have introduced [7] the use of *graph grammars* [5], [4] to model local rules for the self-assembly and self-organization of robotic devices [1]. In this setting, a collection of robotic particles is modeled as a vertex-labeled graph, where vertices correspond to robots, labels correspond to the internal discrete states of the robots, and edges correspond to the existence of a physical connection between the robots. A graph grammar is a set of rules of the form $L \Rightarrow R$ where L and R are (small) graphs. Rules are interpreted as follows: If there exists a copy of L in whatever graph currently describes the structure of the system, it may be replaced by R. The idea is that, if the rules are small, the robots can decide via local interactions whether the rule applies, and they can apply the rule without knowledge of the global structure.

This work is partially supported by NSF Grant #0347955.

We can solve several controller-synthesis problems for graph grammars. For example, given any graph G_d, a grammar Φ can be synthesized with rules involving at most three vertices such that the unique stable assemblies of the grammar are isomorphic to G_d [7]. We can also define other processes such as self-replication and locomotion using graph grammars.

We have used graph grammars to control the assembly of robotic programmable parts [1], and are actively pursuing the application of these ideas to MEMs self-assembly and molecular self-organization using DNA. The present paper is particularly geared toward the molecular setting in that we show the relationship between graph grammars and the pathways that connected components may take in trajectories of the system. The relationship is similar to the relationship between chemical reactions and metabolic networks [2].

In particular, we introduce the *characteristic automata set* of a graph grammar, which contains one automaton for each component type in the initial graph of a given system. Each automaton describes the evolution of the corresponding component in terms of its state and local neighborhood. Actions shared by all of the automata describe the synchronization between processes. In this paper, we make these definitions precise and prove a simple theorem about the representation of the automaton. We also show that the problem of determining the characteristic automata set is co-**NP** complete. Then we describe an approximate algorithm that generates the characteristic automata set of a system and give several examples. Finally, we describe how to use automata sets as specifications and then find grammars that satisfy them, and give examples of how this is done.

II. Definitions

A. Graph Grammars

In this section we review the definitions related to our notion of a graph grammar. A more complete treatment of these definitions and their consequences can be found elsewhere [7].

A *simple labeled graph* over an alphabet Σ is a triple $G = (V, E, l)$ where V is a set of *vertices*, E is a set of unordered pairs or *edges* from V, and $l : V \to \Sigma$ is a labeling function. We restrict our discussion to simple labeled graphs and thus simply use the term *graph*. We denote an edge $\{u, v\} \in E$ by uv. We denote by V_G, E_G and l_G the vertex set, edge set and labeling function of the graph G.

We usually use the either the alphabet $\Sigma = \{a, b, c, ...\}$ or the alphabet $\Sigma = \{a_0, a_1, ..., b_0, b_1, ..., c_0, c_1, ...\}$ for labels. We interpret a vertex labeled by a_i as having "type" a and having state $i \in \mathbb{N}$.

Definition 2.1: A *rule* is a pair of graphs $r = (L, R)$ where $V_L = V_R$. The graphs L and R are called the *left hand side* and *right hand side* of r respectively. The *size* of r is $|V_L| = |V_R|$. Rules whose vertex sets have one, two and three vertices are called *unary*, *binary* and *ternary*, respectively.

Example 2.1: To illustrate the notions in this paper, we will consider a running example that models a *catalytic system*. The system has three rules, which we have labeled r_1, r_2 and r_3:

$$\Phi_{catalyst} = \left\{ \begin{array}{llll} a_0 \quad c_0 & \Rightarrow & a_1 - c_1 & (r_1) \\ a_0 \quad c_1 & \Rightarrow & a_1 - c_2 & (r_2) \\ \quad c_2 \quad\quad & & c_0 & \\ \quad / \quad \backslash & \Rightarrow & & \\ a_1 \quad\quad a_1 & & a_2 \text{---} a_2 & (r_3) \end{array} \right.$$

The relative locations of the vertices represent their identities. In the last of the three rules, for example, $V_L = V_R = \{1, 2, 3\}$ and vertex 1 is labeled by $l_L(1) = a_1$ in the left hand side and by $l_R(1) = a_2$ in the right hand side. We interpret the rules in this grammar as follows: We start with a collection of robots, some of which are labeled a_0 and some of which are labeled c_0. When a robot labeled a_0 bumps into another labeled c_0, the two "connect" and change their labels to a_1 and c_1 respectively. When a robot labeled a_0 bumps into another labeled c_1, the two connect and change to a_1 and c_2 respectively. Finally, if a robot labeled c_2 ever finds itself connected to two other robots both labeled a_1, it connects them, changes their labels to a_2 and then disconnects itself and changes its label back to c_0. We will return to this example throughout the paper. ▲

We denote the set of all (simple labeled) graphs over Σ by $\mathfrak{G}$ and the set of all finite rules over Σ by $\mathfrak{R}$. A *homomorphism* between graphs G_1 and G_2 is a function $h : V_{G_1} \to V_{G_2}$ that preserves edges: $xy \in E_{G_1} \Leftrightarrow h(x)h(y) \in E_{G_2}$. A *monomorphism* is an injective homomorphism. An *isomorphism* is a surjective monomorphism: in this case, G_1 and G_2 are said to be *isomorphic*. A homomorphism h is said to be *label preserving* if $l_{G_1} = l_{G_2} \circ h$.

Definition 2.2: A rule $r = (L, R)$ is *applicable* to a graph G if there exists a label-preserving monomorphism $h : V_L \to V_G$. In this case, the function h is called a *witness*. An *action* on a graph G is a pair (r, h) such that r is applicable to G with witness h.

Definition 2.3: Given a graph $G = (V, E, l)$ and an action (r, h) on G with $r = (L, R)$, the *application* of (r, h) to G yields a new graph $G' = (V, E', l')$ defined by

$$\begin{aligned} E' &= (E - \{h(x)h(y) \mid xy \in L\}) \\ &\quad \cup \{h(x)h(y) \mid xy \in R\} \end{aligned}$$

$$l'(x) = \left\{ \begin{array}{l} l(x) \text{ if } x \notin h(V_L) \\ l_R \circ h^{-1}(x) \text{ otherwise.} \end{array} \right.$$

We write $G \xrightarrow{r,h} G'$ to denote that G' was obtained from G by the application of (r, h).

Example 2.2: The first rule (r_1) in the grammar $\Phi_{catalyst}$ defined in Example 2.1 is applicable to any graph $G = (V, E, l)$ having the property that there exist distinct vertices u and v such that

1) $l(u) = a_0$
2) $l(v) = c_0$ and
3) $uv \notin E$.

The witness to this fact is the function $h : \{1, 2\} \to V$ defined by $h(1) = u$ and $h(2) = v$. Finally, the application of the action (r_1, h) to G yields a graph $G' = (V, E', l')$ identical to G except that

1) $l'(u) = a_1$
2) $l'(v) = c_1$ and
3) $uv \in E'$. ▲

Definition 2.4: A *system* is a pair (G_0, Φ) where G_0 is the *initial graph* of the system and Φ is a set of rules (called the *rule set* or *grammar*).

Example 2.3: An initial graph $G_0 = (V, E_0, l_0)$ for the grammar $\Phi_{catalyst}$ defined in Example 2.1 would have $l_0(v) \in \{a_0, c_0\}$ for each $v \in V$. ▲

In previous expositions on the basic definitions of graph grammars, we would now describe how to obtain *trajectories* from a system starting with G_0 and using the $\xrightarrow{a,h}$ relation. However, in this paper it will be more convenient to describe the operational semantics of a system using *automata*, which we do in the next sub-sections.

B. Automata

Let A be an alphabet different from Σ. We use the symbol A instead of Σ (the alphabet of labels for labeled graphs) to emphasize that elements in A represent *actions*.

By an *automaton*, over the alphabet A, we mean a tuple $M = (Q, q_0, \Delta, Q_F)$ where Q is a set of *states*, the state $q_0 \in Q$ is the *initial state*, the set $Q_F \subseteq Q$ is the set of *final states* and $\Delta \subseteq Q \times A \times Q$ is a transition relation. Trajectories of M are of the form

$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} q_n \xrightarrow{a_n} ...$$

where $(q_i, a_i, q_{i+1}) \in \Delta$ for all i. Trajectories may be either finite (when and only when they end with a state in Q_F) or infinite. The set of actions $a \in A$ for which there exists states $q, q' \in Q$ such that $(q, a, q') \in \Delta$ is called the *action set* of the automata and is denoted A_M.

Remarks: This definition is equivalent to the standard definition of a *Bücci Automaton* found in the literature [10]. Bücci Automata are usually studied with respect to the languages (strings of actions) they accept. We think of automata as representing discrete event systems [9] with actions in A representing events that change a system from one state to another. Thus the term *trajectory* to refer to a string of states and actions. Finally, we consider sets of automata over the same set of actions. In this setting, the full state of the set is a mapping from the automata set to the set of states (that

is, a vector of states). If we suppose that actions are taken concurrently we arrive at the definition of a *marked graph* [3], a simple kind of Petri Net. In fact, Petri Nets have been used to represent metabolic processes in biology [6] in a way that resembles characteristic automata sets.

If $\sim$ is an equivalence relation on Q with equivalence classes $[q]$ for all $q \in Q$, then one may obtain a new automaton $M/\sim = (Q', q_0', \Delta', Q_F')$ (called the automaton induced by $\sim$) defined by

1) $Q' = \{[q] \mid q \in Q\}$
2) $q_0' = [q_0]$
3) $([q_1], a, [q_2]) \in \Delta'$ if and only if there exist $q_1' \in [q_1]$ and $q_2' \in [q_2]$ such that $(q_1', a, q_2') \in \Delta$
4) $Q_F' = \{[q] \mid q \in Q_F\}$.

There are many ways to compare automata, such as with simulation and bisimulation. For the purposes of this paper, we will need to know when two automata M and M' are *structurally equivalent*, by which we mean that there exists a bijection $f : Q_1 \to Q_2$ an injection $g : A_M \to A_{M'}$ such that

1) $(q_1, a, q_2)\Delta \Leftrightarrow (f(q_1), g(a), f(q_2)) \in \Delta'$;
2) $f(q_0) = q_0'$;
3) $f(Q_F) = Q_F'$.

When two automata are structurally equivalent, we write $M \cong M'$

We can (somewhat indirectly) express the semantics of a graph grammar (G_0, Φ) in terms of an automaton.

Definition 2.5: Let (G_0, Φ) be a graph grammar. Define the *associated automaton* $M(G_0, \Phi) = (Q, q_0, \Delta, Q_F)$ by

1) $Q = \mathfrak{G}$
2) $A_M = \Phi$;
3) $(G, r, G') \in \Delta$ if and only if there exists an h such that $G \xrightarrow{r,h} G'$;
4) $q_0 = G_0$;
5) $Q_F =$ graphs G to which no rule in Φ is applicable.

Note that many graphs in $\mathfrak{G}$ may not be reachable from G_0 via applications of rules in Φ. We will not be concerned with these graphs in general even though they are technically part of the automaton.

Example 2.4: Suppose that G_0 is the first graph (on the left) in Figure 1. The rest of the figure shows the beginning of one possible trajectory through $M(G_0, \Phi_{catalyst})$. The entire automaton has many more states and actions than shown. Note: Graphs in this paper are often drawn in such a way as to suggest a particular geometry, but the reader is cautioned to remember that vertices in the graphs we consider have no "locations" – they simply have labels and connections to other vertices. Any physical implementation of the grammars we describe here will have to interpret "label" and "connection" appropriately (as in, for example, out robotic implementation of graph grammars [1]). ▲

C. Point of View

Definition 2.6: Given a simple labeled graph $G = (V, E, l)$, the *k-neighborhood* of a vertex set $U \subseteq V$ is defined to be the graph $N_G(U, k)$ induced by the vertices

$$\{v \in V \mid \text{there exists some } u \in U \text{ such that } d(u, v) \leq k\}$$

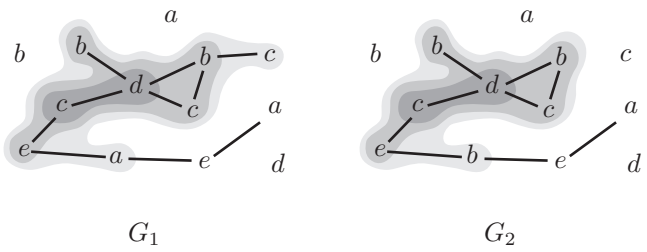

Fig. 2. The k neighborhoods of a set U with respect to two graphs G_1 and G_2 for $k \in \{0, 1, 2\}$. In this figure, U consists of two vertices labeled by c and d respectively. The graphs shown are U–0, U–1 equivalent, but not U–2 equivalent.

that are distance k or less away from some vertex in U.

Definition 2.7: Let G_1 and G_2 be two graphs over the same vertex set V. Let $U \subseteq V$ and k be a natural number. We say that G_1 and G_2 are *U–k equivalent*, written

$$G_1 \overset{U}{\underset{k}{\sim}} G_2,$$

if and only if there exists a map h such that

1) h is a label-preserving isomorphism between $N_{G_1}(U, k)$ and $N_{G_2}(U, k)$ and
2) $h|_U = id_U$.

Remarks: If $k = 0$, then $N_G(U, k)$ is simply the graph induced by U. If the vertices in U are part of a finite component C and the maximum distance from a vertex in U to any other vertex in the component is k, then $N_G(U, j) = N_G(U, k) = C$ for all $j \geq k$. If G and G' are U–k equivalent, then they are U–$(k-1)$ equivalent. Figure 2 shows the 0, 1 and 2-neighborhoods of a set U for two graphs G_1 and G_2 over the same vertex set V. In the figure, $U \subseteq V$ consists of two vertices labeled by c and d respectively. The graphs shown are U–0 and U–1 equivalent, but not U–2 equivalent.

Given a collection $\mathcal{G}$ of graphs over the same vertex set V, we denote the equivalence class of $G \in \mathcal{G}$ with respect to $\overset{U}{\underset{k}{\sim}}$ by

$$[G]_U^k = \{H \in \mathcal{G} \mid G \overset{U}{\underset{k}{\sim}} H\}.$$

Furthermore, the k-neighborhoods of U with respect to all of the graphs in a given equivalence class $[G]_U^k$ are all isomorphic. Thus, we may safely represent $[G]_U^k$ by $N_G(U, k)$.

Example 2.5: The graphs G_1 and G_2 shown in Figure 2 are both elements of $[G_1]_U^0$, where U contains the vertices labeled c and d located in the darkly shaded region of the figure. *Every* graph in this class has the property that the vertices in U induce the subgraph $c - d$, thus we use this graph as the representation of $[G_1]_U^0$. ▲

The main phenomenon we are concerned with in this paper is how the local neighborhood of a collection U of vertices changes as graph grammar rules are applied. This notion can be captured by an appropriately instantiated automaton.

Definition 2.8: Let (G_0, Φ) be a graph grammar over a set of vertices V, let $U \subseteq V$. We define the *k-neighborhood automaton for U* to be the automaton $M/\overset{U}{\underset{k}{\sim}}$ induced by the

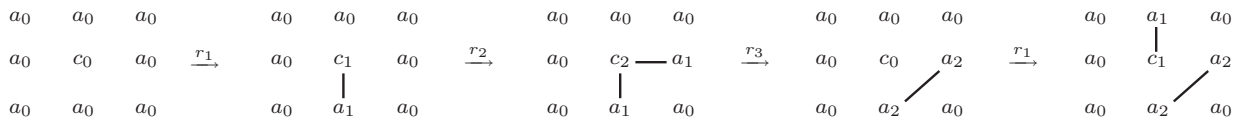

Fig. 1. The initial part of one possible trajectory arising from the system $(G_0, \Phi_{catalyst})$ where G_0 is the graph shown in the upper left of the figure.

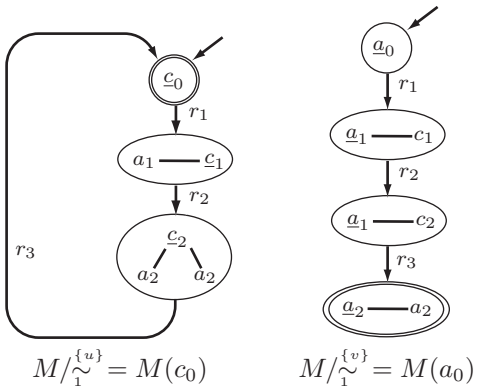

Fig. 3. The 1-neighborhood automata with respect to $\Phi_{catalyst}$ for any vertex sets $\{u\}$ and $\{v\}$ where $l_0(u) = c_0$ and $l_0(v) = a_0$. These automata are structurally equivalent to the automata corresponding to any other vertices initially labeled c_0 and a_0, respectively. Therefore, we may denote them by $M(c_0)$ and $M(a_0)$ as well. Together, these two automata form the characteristic automata set $\mathcal{M}(G_0, \Phi_{catalyst})$.

equivalence classes of $\overset{U}{\underset{k}{\sim}}$, where $M = M(G_0, \Phi)$ is the automaton associated with (G_0, Φ) (Def. 2.5). In presentations of $M/\overset{U}{\underset{k}{\sim}}$, we represent $[G]^k_U$ by $N_G(U, k)$.

Example 2.6: Figure 3(left) shows the 1-neighborhood automaton for $U = \{u\}$ for the grammar $\Phi_{catalyst}$, where $u \in V$ is any vertex with the initial label $l_0(u) = c_0$. Figure 3(right) shows the 1-neighborhood automata for any single vertex initially labeled by a_0. In the figures, the vertex in U is shown underlined, the initial states are indicated by incoming arrows and the final states are indicated by double circles around the states. According to Definition 2.8, each state is instantiated by an equivalence class of graphs. The representation of the equivalence classes is via the graph $N_G(U, 1)$. ▲

Remarks: The identification of final states is not entirely straightforward. If we assume that the initial graph is finite and disconnected, that there is one vertex initially labeled c_0 and that there are an even number of vertices labeled a_0, then the final states are as shown in the figure. However, if this is not the case, then other states may be final as well. We will not stress this difficulty in this paper, mainly because if there are an enormous number of vertices, then the automata in the figure describe the behavior of the system at least until most of the "reactants" are "used up".

III. Characteristic Automata

We are particularly interested in the paths that the components in the initial graph G_0 take in trajectories of (G_0, Φ). We have the following result.

Theorem 3.1: Let (G_0, Φ) be a graph grammar over vertices V with associated automaton M. Suppose that C_1 and C_2 are components of G_0 with vertex sets V_1 and V_2 respectively. If C_1 and C_2 are isomorphic then

$$M/\overset{V_1}{\underset{k}{\sim}} \cong M/\overset{V_2}{\underset{k}{\sim}} .$$

Proof: Suppose that C_1 and C_2 are distinct so that V_1 and V_2 are disjoint. Let h be a witness to the fact that C_1 and C_2 are isomorphic and let $\tilde{h}$ be the extension of h to V defined by

$$\begin{aligned} \tilde{h}|_{V_1} &= h \\ \tilde{h}|_{V_2} &= h^{-1} \\ \tilde{h}|_{V-(V_1-V_2)} &= \mathrm{id}_V. \end{aligned}$$

Let G' be the graph defined by re-indexing each vertex v of G to $\tilde{h}(v)$ transforming the edges accordingly (so that the re-indexing is an isomorphism) . Finally, define $f([G]^k_{V_1}) = [G']^k_{V_2}$ and $g = \mathrm{id}_{\mathfrak{R}}$ to demonstrate structural equivalence. ■

This result suggests that the automata associated with the various component *types* in G_0 are sufficient to capture the k-neighborhood behaviors, leading to the following definition. In the definition, we denote by $\mathcal{C}(G_0) = \{C_1, C_2, ...\}$ the set of components of G up to isomorphism.

Definition 3.1: Suppose that $C_j \in \mathcal{C}(G_0)$ has vertex set $V_j \subseteq V_{G_0}$. Let $M = M(G_0, \Phi)$ be the automaton associated with (G_0, Φ). The *k-th characteristic automata set* of the grammar (G_0, Φ) is the set

$$\mathcal{M}(G_0, \Phi) = \{M/\overset{V_1}{\underset{k}{\sim}},\ M/\overset{V_2}{\underset{k}{\sim}},\ ...\}.$$

By Theorem 3.1, the choice of which components of G_0 we use to represent their types does not matter. So we are justified in calling $\mathcal{M}(G_0, \Phi)$ "the" characteristic automata set. Furthermore, we will write $M(C)$ for $M/\overset{V_i}{\underset{k}{\sim}}$ whenever C is label-preserving isomorphic to C_i.

Remark: As noted above, a characteristic automata set is essentially a Petri Net or marked graph. We think of the full state of a characteristic automata set $\mathcal{M}$ as a "marking" $m : \mathcal{M} \to \bigcup_{M \in \mathcal{M}} Q_M$ that assigns each automaton to its current state. A new marking m' is determined when a rule r is applied by unmarking the preset of r and marking the postset of r.

Example 3.1: An initial graph of $\Phi_{catalyst}$ (see 2.3) has two component types: A single vertex labeled c_0 and a single vertex labeled a_0. The characteristic automata set is, thus,

$$\mathcal{M}(G_0, \Phi_{catalyst}) = \{M(c_0), M(a_0)\},$$

where $M(c_0)$ and $M(a_0)$ are as shown in Figure 3. ▲

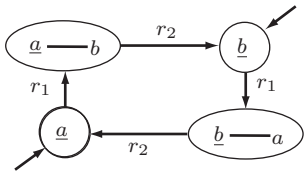

Fig. 4. The 1-characteristic automata set for the grammar in example 3.2 when G_0 is the discrete graph containing exactly one vertex labeled a and one vertex labeled b. The set can be written as a single automaton with different start states for different components.

Example 3.2: Two automata in $\mathcal{M}(G_0, \Phi)$ may be structurally equivalent if one ignores the requirement that their initial states should correspond. As an example, consider the grammar

$$\Phi = \left\{ \begin{array}{lllll} a & b & \Rightarrow & a-b \\ a-b & & \Rightarrow & b & a. \end{array} \right.$$

A system with these rules whose initial graph contains a's and b's, will have trajectories wherein particles whose 1-neighborhoods are initially the singleton graph a will eventually change to having 1-neighborhoods equal to the singleton graph b. The characteristic automata set therefore has two automata that are structurally equivalent, except with different start states, as shown in Figure 4 for the case when G_0 contains exactly two vertices, one labeled a and one labeled b. This also shows the dependence of $\mathcal{M}(G_0, \Phi)$ on G_0: For this example, the number of states in $\mathcal{M}(G_0, \Phi)$ grows exponentially with the number of vertices in G_0. ▲

IV. Construction of the Characteristic Automata Set

The characteristic automata set $\mathcal{M}(G_0, \Phi)$ is a useful tool for analyzing the behavior of a graph grammar system. It can be used to identify whether a grammar has cyclic behaviors; how trajectories in a system can deadlock; and in general, what the possible fates of a component in G_0 are. The set can be constructed by hand for small examples, but for larger examples it is useful to have an automated method, which we now describe. Fix a grammar Φ and an initial graph G_0.

1) Let
$$G_0 \xrightarrow{r_{i_0}} G_1 \xrightarrow{r_{i_1}} ... \xrightarrow{r_{i_{n-1}}} G_n$$
be a finite trajectory of $M(G_0, \Phi)$ (or a finite prefix of an infinite trajectory). This can be found by simulating the system for n steps, either probabilistically (in which case we can only approximate the actual characteristic automata set) or by enumerating all possible trajectories (feasible only for small systems).
2) For each component C with vertices V_C of G_0, define an automaton $M(V_C) = (Q(V_C), q_0(V_C), \Delta(V_C), Q_F(V_C))$. The states of $Q(V_C)$ are defined by the following procedure:

$Q(V_C) = \varnothing$
for $i = 0$ to n **do**
  if $\neg\exists H \in Q$ such that $G_i \overset{V_C}{\underset{k}{\sim}} H$
    then $Q(V_C) \leftarrow Q(V_C) \cup N_{G_i}(V_C, k)$
  endif
endfor

The rest of the automaton is defined by
  a) $q_0(V_C) = N_{G_0}(V_C, k)$
  b) $(H, r, H') \in \Delta(V_C) \Leftrightarrow$ there exists an j such that $G_j \overset{V_C}{\underset{k}{\sim}} H$ and $G_{j+1} \overset{V_C}{\underset{k}{\sim}} H'$ and $r_k = r$.
  c) $Q_F(V_C) = \{H \in Q(V_C) \mid H \overset{V_C}{\underset{k}{\sim}} G_n\}$ if no rules in Φ are applicable to G_n and $Q_F(V_C) = \varnothing$ otherwise.
3) Next, suppose that $C_1, ..., C_j$ are all the components of G_0 isomorphic to a given component $C = C_1$. From the set $M(C_1), ..., M(C_j)$ we can build an approximation (in that it may not contain all possible states) of the characteristic automaton for components of this type. First, re-index the vertices V_{C_k} to V_{C_1} by using any isomorphism between C_k and C_1. Second, let $M_i(V_1)$ be the automaton obtained from $M_i(V_{C_i})$ by this re-indexing. Third, define the characteristic automaton for components of type C by $M(C) = (Q(C), q_0(C), \Delta(C), Q_F(C))$ where
  a) $Q(C) = \bigcup Q_i(V_1)$
  b) $q_0(C) = q_0(V_1)$
  c) $\Delta(C) = \bigcup \Delta_i(V_1)$
  d) $Q_F(C) = \bigcup Q_{F,i}(V_1)$.

We have implemented this algorithm in *Mathematica* along with a number of other graph grammar utilities. Each of the characteristic automata sets presented in this paper was either generated by the above algorithm or it was generated by hand and checked with the algorithm (and a number of corrections were made!).

Example 4.1: Consider the grammar

$$\Phi = \left\{ \begin{array}{llll} a_0 & a_0 & \Rightarrow & b_0 - b_1 & (r_1) \\ b_1 & a_0 & \Rightarrow & c_0 - b_2 & (r_2) \\ b_0 & b_2 & \Rightarrow & c_0 - c_0 & (r_3) \end{array} \right.$$

that constructs cycles of three vertices, all labeled by c_0 starting from a soup of vertices all labeled a_0. The characteristic automata set contains one automaton (because the initial graph contains one component type), which we generated using the above algorithm and which is shown in Figure 5. It is clear from that figure that a vertex may either end up as part of a three-cycle of c_0 vertices, or it may end up as a c_0 with two unconnected c_0 vertices — which occurs when two $b_0 - c_0 - b_2$ components join to make a six-cycle via two applications of the rule r_3. This a result of the fact that the reachable set of a binary rule set is *closed under covers* [7]. ▲

Remark: In all of the examples we have so far considered, the components of G_0 have been single vertices, perhaps with different labels. In general, the characteristic automata set "makes sense" for components whose vertices remain connected in any trajectory, as, for example, proteins do when forming supermolecular aggregates. If the vertices in a component ever become disconnected, then they must experience essentially separate evolutions. We plan to report on a generalization of the ideas in this paper that can handle this difficulty in a future paper.

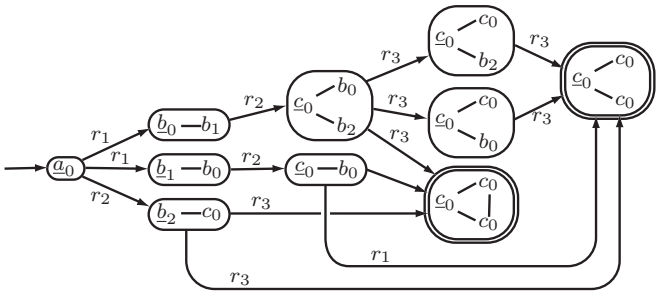

Fig. 5. The characteristic automaton for the grammar described in Example 4.1. In it, a vertex may either end up in a three-cycle or in part of a larger cycle. The Figure was redrawn from a figure generated by the *Mathematica* implementation of the algorithm described in Section IV.

The procedure outlined above for generating $\mathcal{M}(G_0, \Phi)$ is clearly time consuming, which we can show formally. Let *CHARAUT* be the following problem: Given a grammar Φ, a *finite* initial grammar G_0 and a set of automata of $\mathcal{M}$, decide whether $\mathcal{M}(G_0, \Phi) = \mathcal{M}$. We suspect the following.

Theorem 4.1: The decision problem *CHARAUT* is **coNP**-complete.

Proof: Begin with an instance of TAUTOLOGY [8, p.219] consisting of clauses $C_1, ..., C_k$ over variables $x_1, ..., x_n$. Suppose that

$$C_i = (v_{i,1} \vee v_{i,2} \vee v_{i,3})$$

where $v_{i,j}$ is either x_l or $\overline{x}_l$ for some $l \in \{1, ..., n\}$. From this instance, we first define $G_0 = (V, E_0, l_0)$ by

$$\begin{aligned} V &= \{1, ..., n+k+2\} \\ E_0 &= \varnothing \\ l_0(i) &= \begin{cases} (x_i, \perp) \text{ if } 1 \le i \le n \\ (C_{i-n}, \perp, \perp, \perp) \text{ if } n+1 \le i \le n+k \\ T \text{ if } i = n+k+1 \\ F \text{ if } i = n+k+2. \end{cases} \end{aligned}$$

Thus, there is one vertex for each variable, one for each clause, and two more vertices labeled T and F, respectively.

Next, we define the rule set Φ. The vertex labeled $(x_i, \perp)$ corresponds to the truth assignment for x_i. To non-deterministically explore all truth assignments, we use the rule sets

$$\Psi_i = \begin{cases} (x_i, \perp) \;\; T \;\Rightarrow\; (x_i, T) \;\; T \\ (x_i, \perp) \;\; F \;\Rightarrow\; (x_i, F) \;\; F \end{cases}$$

The vertex labeled T and the vertex labeled $(x_i, \perp)$ can interact to change the label $(x_i, \perp)$ to the label (x_i, T), leaving the vertex labeled T unchanged.

The vertex labeled $(C_j, \perp, \perp\perp)$ corresponds to the truth value of the clause C_j. To determine the value of the clauses, we use rules that allow the clause-vertices to interact with the variable-vertices. Suppose it happens that $v_{1,1} = \overline{x}_7$. We introduce the corresponding rules

$$\Upsilon_{1,1} = \begin{cases} (C_1, \perp, \#, *) \;\; (x_7, T) \;\Rightarrow\; (C_1, F, \#, *) \;\; (x_7, T) \\ (C_1, \perp, \#, *) \;\; (x_7, F) \;\Rightarrow\; (C_1, T, \#, *) \;\; (x_7, F) \end{cases}$$

where "$\#$" and "$*$" range over the set $\{\perp, T, F\}$. Thus, there are 18 such rules: one for each of the three literals in the clause.

The resulting rule set Φ is then

$$\Phi = \left(\bigcup_{i=1}^{n} \Psi_i \right) \cup \left(\bigcup_{i=1}^{k} \bigcup_{j=1}^{3} \Upsilon_{i,j} \right).$$

In all, there are $2n + 54k$ rules (required for the reduction of TAUTOLOGY to be polynomial). As long as there are unassigned variables and unevaluated clauses, some rule in Φ will apply. Each trajectory of (G_0, Φ) explores one possible truth assignment (and resulting clause evaluation), and every truth assignments is represented by some trajectory of the system.

We next describe the candidate automata set $\mathcal{N}$. There is one automata for each label in the initial graph. The automata for the vertices labeled T and F are trivial. The automata for the vertices initially labeled $(x_i, \perp)$ branch from this state to either (x_i, T) or (x_i, F).

The candidate automaton for $(C_i, \perp, \perp, \perp)$ is defined to have 26 states, one for each possible label assigning literals to values, except that we leave out the state labeled (C_i, F, F, F). If the instance of TAUTOLOGY is in fact true for all assignments, then this state of the automaton for C_i should never be reached. The transitions of the automaton correspond to the rules $\Upsilon(i, j)$ for each i and j. Thus, each clause-automaton has 26 states and 54 transitions.

The total number of states in the automata set $\mathcal{N}$ is then $2 + 3n + 26k$ and the number of transitions is $2 + 2n + 54k$. The reduction is, therefore, computable in polynomial time.

Furthermore, the instances $C_1, ..., C_k$ is a tautology if and only if $\mathcal{N}$ is the characteristic automata set for (G_0, Φ). ■

V. Grammar Synthesis from the Characteristic Automata Set

The synthesis problem for graph grammars and characteristic automata is stated as follows: Given a set $\mathcal{M}$ of automata, a graph G_0 and a one-to-one correspondence between the components of G_0 and the automata in $\mathcal{M}$, find a grammar Φ such that the characteristic automata set $\mathcal{M}(G_0, \Phi)$ corresponds to $\mathcal{M}$. The set $\mathcal{M}$ is called the *specification* and the system (G_0, Φ) is called the *solution*. An automatic method for this problem likely requires an extensive search, and may be computationally tractable only for small systems. In this section, we describe via examples how a solution to a specification can be found and when they cannot be found.

Example 5.1: Consider the two automata, shown in Figure 6, for an initial graph G_0 with $l_0(V_{G_0}) = \{a, b\}$. We suppose that these represent the 1-neighborhood automata for some graph grammar. This example has many solutions. We construct one of them here.

First, we choose an instantiation of rule r_1. Since an arc labeled by r_1 exits the initial states of both automata, the left hand side of r_1 must have at least one a and one b in it. Furthermore, either the labels or the 1-neighborhoods of these vertices must change, otherwise one of the arcs

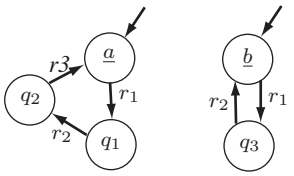

Fig. 6. The specification discussed in Example 5.1.

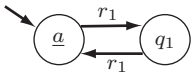

Fig. 7. An example specification with no solution, as discussed in example 5.2.

labeled r_1 would be a self-loop. There are several choices. For example,

$$a \quad b \quad \Rightarrow \quad c - d. \qquad (r_1)$$

Note that had we instead chosen $a \quad b \quad \Rightarrow \quad a - b$, we would end up with transitions not shown in Figure 6 because, for example, the graph $a - b - a$ would be reachable as well via rule r_1

Continuing along the same lines, we choose the following instantiations for r_2 and r_3:

$$\begin{array}{lll} c - d & \Rightarrow & e \quad b \qquad (r_2) \\ e \quad e & \Rightarrow & a - a \qquad (r_3) \end{array}$$

Once again, there are several choices for these rules; the reader can imagine others. ▲

The argument outlined in the previous example hints at the complexity of constructing a solution for a given automata set: We must instantiate rules that satisfy the constraints given by the presence of transitions labeled by the rule; We must ensure that a rule instantiation does not generate new transitions (which seems to involve solving the *CHARAUT* problem); We must make sure that a particular instantiation does not make later instantiations impossible. We speculate the problem of finding a solution is provably intractable.

Example 5.2: Not all specifications have solutions. A simple example is shown in Figure 7, which shows a single automaton for an initial graph G_0 with $l_0(V_{G_0}) = \{a\}$. The rule r_1 must apply to graphs in both states. Since it applies only to the initial state of the automaton for the component a, it must have only copies of a in its left hand side. Also, it must change the 1-neighborhood of components labeled a to some other structure, say H. Finally, since r_1 is applicable to graphs in state q_1, its left hand side must contain copies of H in it, which is a contradiction. ▲

VI. Discussion

We have demonstrated the relationship between a graph grammar and its characteristic automata set, which relates the components in the graph to automata describing how the local neighborhoods of those automata change as a result of rule applications. We also described how to find one object given the other, either by a formal procedure, or through examples.

The two types of object discussed here are separated by significant computational complexity. We *do not* find this particularly discouraging, rather we hope that insights gleaned from one description may lead to insights into the other. Furthermore, the method for generating the characteristic automata set is one that produces a series of better and better approximations of the actual result, each of which is useful for analyzing possible behaviors of the system.

One of the goals of our research is to specify, at a high level of abstraction, the behaviors of a system and then produce a grammar that implements it (as in Section V). Thus we plan to spend considerable effort in the future devising efficient methods for approaching this problem. Also, in practice, real systems that implement grammars (such as our self-organizing robot platform [1]) apply rules at random based on statistical-mechanical considerations. Thus, in a real-world setting, the automata described here would be labeled with *rates* that can be used to infer the most likely pathways taken by components, and these rates would have to be determined from properties of a probabilistic implementation of a grammar. However, rates are not preserved by the equivalence of graphs used to form the characteristic automata set, except, perhaps, in certain situations. We plan to report on this complication in a future paper.

References

[1] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Self-organizing programmable parts. In *International Conference on Intelligent Robots and Systems*. IEEE/RSJ Robotics and Automation Society, 2005.

[2] T. Chevalier, I. Schreiber, and J. Ross. Toward a systematic determination of complex reaction mechanisms. *Journal of Physical Chemistry*, 97(26):6776–6787, 1993.

[3] F. Commoner, A. W. Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer and System Science*, 5:511–523, 1971.

[4] B. Courcelle. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, chapter on Graph Rewriting: An Algebraic and Logic Approach, pages 193–242. MIT Press, 1990.

[5] H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69, 1979.

[6] R. Hofestädt. A petri net application to model metabolic processes. *Systems Analysis Modelling Simulation*, 16(2):113–122, October 1994.

[7] Eric Klavins, Robert Ghrist, and David Lipsky. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 2005. To Appear.

[8] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[9] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, January 1987.

[10] W. Thomas. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter Automata on Infinite Objects, pages 193–242. MIT Press, 1990.