Universal Self-Replication Using Graph Grammars

Eric Klavins*
Electrical Engineering Department
University of Washington
Seattle, WA 98195

Abstract

A graph grammar is described that can be used to replicate arbitrarily labeled strands of particles (modeled as linear graphs). The rules of the grammar dictate how pairs of particles should attach or disassociate upon random collisions, and thus describes a parallel and completely distributed algorithm for replication. The correctness of the algorithm is proved and the rate at which replication occurs is characterized. The algorithm may be applied to systems of *programmable parts* (stirred or agitated in a fluid) that can encode the (very simple) rules of the grammar and that can attach and dis-attach to create strands and other structures.

1 Introduction

Engineering at the realm of the very small presents us with several fundamental challenges, chief among them is the construction of complicated structures from simple parts. Although we have the ability to manufacture large numbers of simple (for example *flat*) small components (using, for example, lithography), the prospect of arranging such components into a prespecified and complicated macro-scale object is daunting. One way to address this problem that is the subject of considerable attention in a variety of fields (MEMs, nanotechnology, biochemistry) is *self-assembly*.

Our starting point in understanding self assembly is the idea of *conformational switching* [14]: Each particle (molecule, robot, etc.) exists in one of several *conformations* or shapes. When two particles collide, they attach or not based on whether their conformations are complimentary, as illustrated in Figure 1. If they do attach, their conformations change (mechanically for example), thereby determining in what future assembly interactions the particless may partake.

As in other work [13, 12], we represent the conformation of a part by a discrete symbol, and we model an assembly as a simple graph labeled by such symbols. Vertices in these graphs represent parts, and the presence of an edge between two parts represents that they are somehow attached.

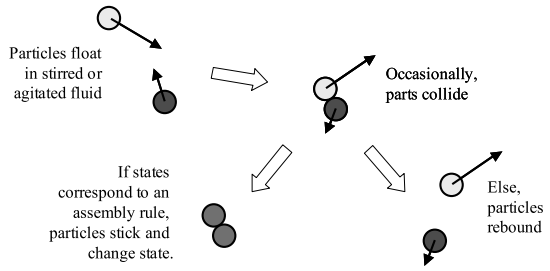

Fig. 1: A particle based embedding of self organization using graph grammars.

One problem then, is to define the rules so that random interactions (that follow the rules) always produce a given graph (the desired assembly). In contrast, the problem we focus on in this paper is the design of a rule set that can replicate *any* suitable labeled "seed" graph (see Figure 2). The products of the replication, being identical to the seed, should continue to replicate until all "raw materials" are used up. The difference between these problems is that the desired assembly is encoded in the rules of the system, whereas in the later problem, the desired assembly is encoded as a seed assembly.

The specific contributions of this paper are the definition of the replication rules, which we call the *replication environment*, and a proof that the rules indeed behave as claimed above when interpreted using the graph grammar approach.

2 Definitions

We begin with basic definitions. Much of this section appeared first elsewhere [12], we include it for completeness. Basic graph-theory definitions [3] are not numbered, but only recalled.

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

*This work was supported in part by NSF Grant number #0347955.

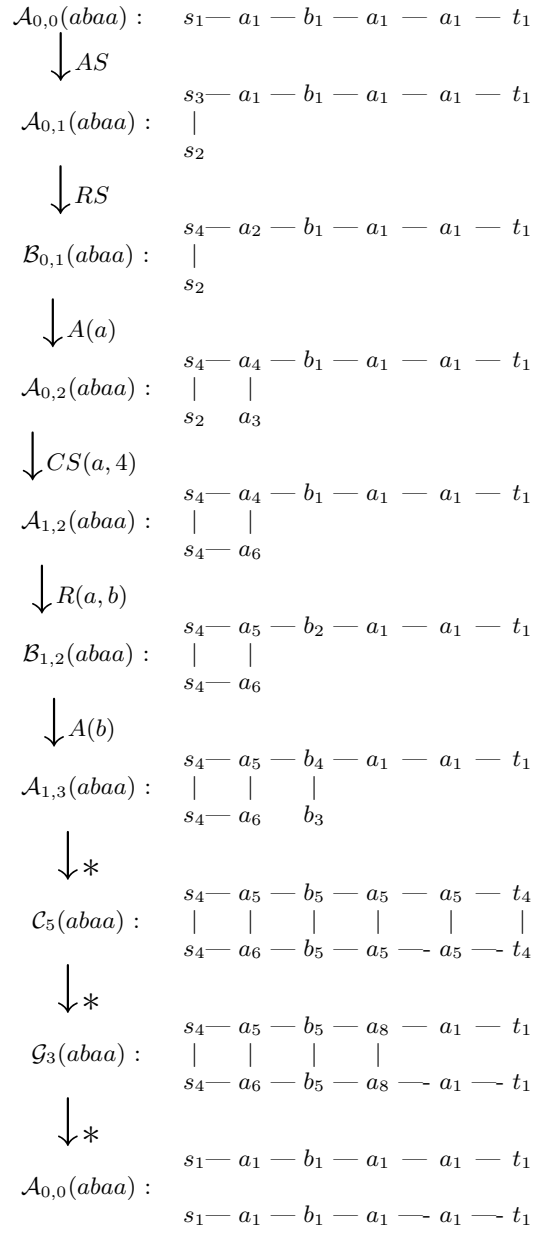

Fig. 2: The replication of the strand $\mathcal{A}_{0,0}(abaa)$ in two phases. In the first phase, each vertex of the strand attaches to a free vertex with the same basic symbol. In the second phase, the replicated strand separates from its parent. The notation used is explained in Sections 3-4.

G or by V, E and l when there is no danger of confusion.

Given graphs G_1 and G_2, we write $f : G_1 \to G_2$ and $f : V_{G_1} \to V_{G_2}$ equivalently to mean that f is a function from the vertex set of G_1 to the vertex set of G_2. A function $h : G_1 \to G_2$ is a label preserving *embedding* if

1. h is injective,
2. $\{x, y\} \in E_{G_1} \Leftrightarrow \{h(x), h(y)\} \in E_{G_2}$,
3. $l_{G_1} = l_{G_2} \circ h$.

If h is also surjective then it is called an *isomorphism*. The graphs G_1 and G_2 are said to be *isomorphic* (written $G_1 \simeq G_2$) if there exists an isomorphism relating them.

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 $|V_L| = |V_R|$.*

Definition 2.2 *A rule r is* applicable *to a graph G if there exists an embedding $h : L \to G$. 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*

$$\begin{aligned} V' &= V \\ E' &= (E - \{\{h(x), h(y)\} \mid \{x, y\} \in L\}) \\ &\quad \cup \{\{h(x), h(y)\} \mid \{x, y\} \in R\} \\ l'(x) &= \begin{cases} l(x) \text{ if } x \notin h(V_L) \\ l_R \circ h^{-1}(x) \text{ otherwise.} \end{cases} \end{aligned}$$

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

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

Definition 2.5 *An* assembly sequence *of a system (G_0, Φ) is a sequence $\{G_i\}_{i=0}^k$ such that there exists a sequence of actions $\{(r_i, h_i)\}_{i=1}^k$ where $r_i \in \Phi$ and*

$$G_i \xrightarrow{r_i,h_i} G_{i+1}$$

for $i \in \{0, ..., k-1\}$.

Thus, a system (G_0, Φ) defines a parallel, non-deterministic dynamical system whose states are labeled graphs over V_{G_0}. The system is non-deterministic since, at any step, many rules in Φ may be simultaneously applicable, each possibly via several witnesses. If the image of the witnesses of a set of actions is disjoint, the actions may be applied *in parallel*.

Two vertices in a graph G are *connected* if there is a path (sequence of edges) connecting them in G. The *connectivity* relation on V is an equivalence relation partitioning V into sets $\{V_i\}_{i \in I}$ where v_1 and v_2 are connected if and only if $v_1, v_2 \in V_i$ for some i. The sets V_i are called the *components* of G.

Definition 2.6 *A connected graph G is* reachable *in a system (G_0, Φ) if there exists a finite assembly sequence $\{G_i\}_{i=0}^k$ of (G_0, Φ) such that G is isomorphic to some component of G_k. The set of all such reachable graphs is denoted $\mathcal{R}(G_0, \Phi)$.*

3 An Environment for Replication

3.1 The Symbols

Let Σ be a set of symbols and let $\omega \in \Sigma^*$ be a finite string of symbols from Σ. Denote[1] by ω^n the n^{th} symbol in ω, where $n \in \{1, ..., |\omega|\}$. Also, let s and t be two distinct symbols *not* in Σ. In the sequel, the symbol s will denote the "start" of the strand to be replicated, the symbol t will represent the "terminal" end of the strand, and ω will denote the symbols that define the body of the strand.

We suppose that each symbol can take on one of a finite number of states $i \in \{0, 1, 2, ..., m\}$ and define

$$\Gamma_i = \{\sigma_i \mid \sigma \in \Sigma \cup \{s, t\}\}$$

to be a new set of symbols. For example s_3 represents the 3^{rd} state of the symbol s and ω_i^n represents the i^{th} state of the n^{th} symbol in the string ω. In the graph grammar presented below,

$$\Gamma = \bigcup_{0 \le i \le m} \Gamma_i$$

is the co-domain of the labeling functions for rules and for the initial graph.

3.2 The Initial Graph

The initial graph consists of two parts: The strand to be replicated and the "raw materials" needed to perform the replication. The strand is defined by

$$\mathcal{A}_{0,0}(\omega) = (V, \varnothing, l)$$

where $V = \{0, ..., |\omega| + 1\}$ and l is defined by

$$l(v) = \begin{cases} s_1 \text{ if } v = 0 \\ \omega_1^v \text{ if } 0 < v \le |\omega| \\ t_1 \text{ if } v = |\omega| + 1. \end{cases}$$

This graph is illustrated for $\omega = abaa$ at the top of Figure 2 and schematically in Figure 3. The purpose of the subscripts in $\mathcal{A}_{0,0}(\omega)$ will be explained later. The "raw materials" are defined by

$$G = (\mathbb{N}, \varnothing, p) \tag{1}$$

where p is subject to the following conditions

1. $p(\mathbb{N}) = \Gamma_0$
2. $|p^{-1}(\sigma_0)| = \infty$ for each $\sigma_0 \in \Gamma_0$.

Thus, G is a discrete graph all of whose vertices are labeled by symbols in state 0. Furthermore, each symbol occurs infinitely often.

The initial graph for the replication of a given strand $\mathcal{A}_{0,0}(\omega)$ is then $G_0(\omega) = \mathcal{A}_{0,0}(\omega) \amalg G$ where $\amalg$ represents the *disjoint union*.

[1] Using a superscript here is contrary to the standard convention, that ω^n denotes the n-fold concatenation of ω, but will help make the notation more concise.

3.3 The Rules

The rules for the replication environment are as follows. Many of them are parameterized by symbols in Σ. We use the variables x and y to represent arbitrary symbols in Σ.

ADDING PARTICLES For each symbol in Σ there is a rule $A(x)$ that matches a free vertex (one in state 0) to a vertex in the strand. We also have rules AS and AT for adding free start and termination vertices respectively.

$$\begin{array}{rcl} AS & : & s_0 \quad s_1 \Rightarrow s_2 - s_3 \\ A(x) & : & x_0 \quad x_2 \Rightarrow x_3 - x_4 \\ AT & : & t_0 \quad t_2 \Rightarrow t_3 - t_4 \end{array}$$

MOVING RIGHT Once a new vertex is matched at a position k along the initial strand, we make the position $k+1$ available to receive a free vertex using the following rules.

$$\begin{array}{rcl} RS(x) & : & s_3 - x_1 \Rightarrow s_4 - x_2 \\ R(x,y) & : & x_4 - y_1 \Rightarrow x_5 - y_2 \\ RT(x) & : & x_4 - t_1 \Rightarrow x_5 - t_2 \end{array}$$

CLOSING Once two adjacent positions are filled, they should be attached. The following rules accomplish this. Closing the new strand occurs sequentially from the beginning to the end of the strand and trails the addition of new vertices. Note that the rules for closing require that the two vertices involved be from the same strand. Because the upper right vertex involved in the first two rules can be in one of two states (4 or 5), an integer i appears as a parameter to those rules.

$$\begin{array}{rcl} CS(x,i) & : & \begin{array}{ccc} s_4 & - & x_i \\ | & & | \\ s_2 & & x_3 \end{array} \Rightarrow \begin{array}{ccc} s_4 & - & x_i \\ | & & | \\ s_4 & - & x_6 \end{array} \\ CS(x,y,i) & : & \begin{array}{ccc} x_5 & - & y_i \\ | & & | \\ x_6 & & y_3 \end{array} \Rightarrow \begin{array}{ccc} x_5 & - & y_i \\ | & & | \\ x_5 & - & y_6 \end{array} \\ CT(x) & : & \begin{array}{ccc} x_5 & - & t_4 \\ | & & | \\ x_6 & & t_3 \end{array} \Rightarrow \begin{array}{ccc} x_5 & - & t_4 \\ | & & | \\ x_5 & - & t_4 \end{array} \end{array}$$

SEPARATION Once a strand is copied and closed, the following rules separate the new strand from the old one. The first set of separation rules breaks apart the links that join the strands.

$$\begin{array}{rcl} BT & : & t_4 - t_4 \Rightarrow t_1 \quad t_1 \\ B(x) & : & x_8 - x_8 \Rightarrow x_1 \quad x_1 \\ BS & : & s_6 - s_6 \Rightarrow s_1 \quad s_1 \end{array}$$

The second set of separation rules change the states of the vertices adjacent to the most recently separated pair, so that they may separate next.

$$\begin{array}{rcrcl} LT(x) & : & x_5 - t_1 & \Rightarrow & x_8 - t_1 \\ L(x,y) & : & x_5 - y_1 & \Rightarrow & x_8 - y_1 \\ LS(x) & : & s_4 - s_1 & \Rightarrow & s_6 - s_1 \end{array}$$

These rules taken together comprise the replication environment. In the sequel we use the symbol Φ to denote the set of all such rules. Note that there are $4|\Sigma|^2+9|\Sigma|+4$ rules in Φ. Also, symbols in Σ can be in one of nine states, the symbol t can be in one of five states and the symbol s can be in one of seven states. Thus, 21 symbols in all are needed. The action of the first few rules is illustrated in Figure 2.

4 Analysis

4.1 The Reachable Set

To prove that the system described above is correct, we must show that the reachable set contains only copies of, or partial copies of, $\mathcal{A}_{0,0}(\omega)$ and that some reachable graph yields, in one step, two copies of the original strand. To proceed, we catalog all the graphs that can possibly arise in any assembly sequence of $(G_0(\omega), \Phi)$. They are shown schematically in Figure 3. The relationship between the reachable graphs and the rules in shown in Figure 4, which shows illustrates the assembly of a length-three string.

There are seven varieties of reachable graph. The graphs $\mathcal{A}_{j,k}(\omega)$ are those wherein (1) the start symbol has been copied (except in $\mathcal{A}_{0,0}(\omega)$) via the rule AS; the first $k-1$ symbols have been copied via the rules $A(x)$, $RS(x)$ and $R(x,y)$; and the first j links in the new strand have been closed via the rules $CS(x,i)$ and $C(x,y,i)$. The graph $\mathcal{B}_{j,k}(\omega)$ is similar, except that k^{th} vertex on the original strand is in state 2 due to the rule $R(\omega^{k-1}, \omega^k)$, meaning it is ready to receive a matching free vertex via the rule $A(\omega^k)$.

The graphs $\mathcal{C}_j(\omega)$ are those wherein all symbols (including s and t) have been copied and the first j links have been closed. Thus, $\mathcal{C}_{n+1}(\omega)$ where $n = |\omega|$ represents two identical attached strands (see, for example, $\mathcal{C}_5(aaba)$ in Figure 2). The graphs $\mathcal{D}_j(\omega)$, $\mathcal{E}_j(\omega)$ and $\mathcal{F}_j(\omega)$ are intermediary graphs required to separate the two strands. The graph $\mathcal{D}_0(\omega)$ consists of the original strand and its completed and separated copy. That is

$$\mathcal{D}_0(\omega) \simeq \mathcal{A}_{0,0}(\omega) \amalg \mathcal{A}_{0,0}(\omega).$$

In Figure 4, we illustrate the structure of the reachable set and its relation to the rules in Φ for a system based on a string ω of size $|\omega| = 3$. We summarize the structure of the reachable set in the following theorem.

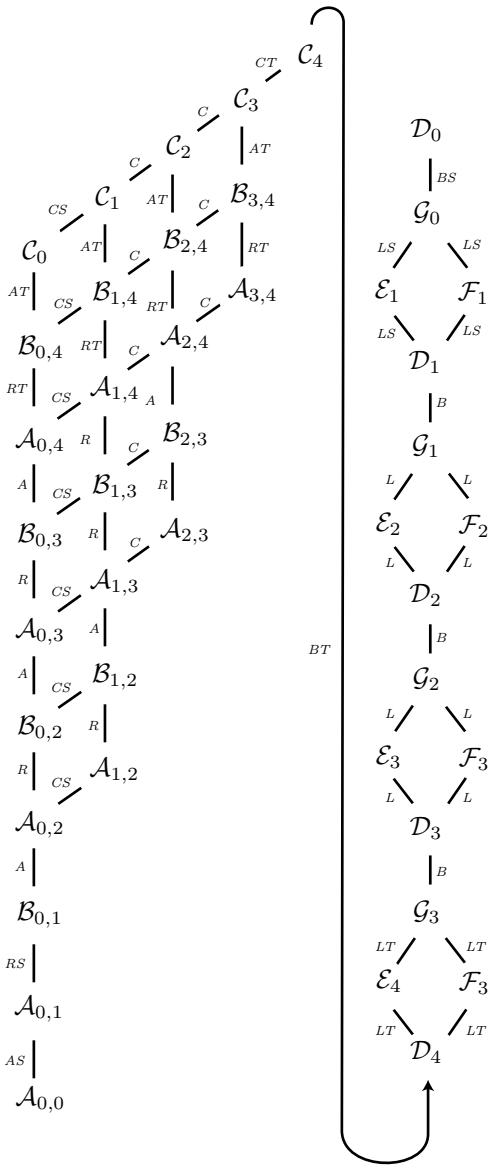

Fig. 4: The assembly graph for $(G_0(\omega), \Phi)$ when $|\omega| = 3$. The dependence of the reachable graphs on ω and the specific form of the rules used have been omitted for the sake clarity.

Theorem 4.1 *For all $\omega \in \Sigma^*$ with $|\omega| = n$, the reachable set $\mathcal{R}(G_0(\omega), \Phi)$ consists of exactly*

i. $\mathcal{A}_{j,k}(\omega)$ *for* $0 \le j \le n$, $0 \le k \le n+1$ *and* $j \le k$

ii. $\mathcal{B}_{j,k}(\omega)$ *for* $0 \le j \le n$, $1 \le k \le n+1$ *and* $j \le k$

iii. $\mathcal{C}_j(\omega)$ *for* $0 \le j \le n+1$

iv. $\mathcal{D}_j(\omega)$ *for* $1 \le j \le n+1$

v. $\mathcal{E}_j(\omega)$ *for* $1 \le j \le n+1$

vi. $\mathcal{F}_j(\omega)$ *for* $1 \le j \le n+1$

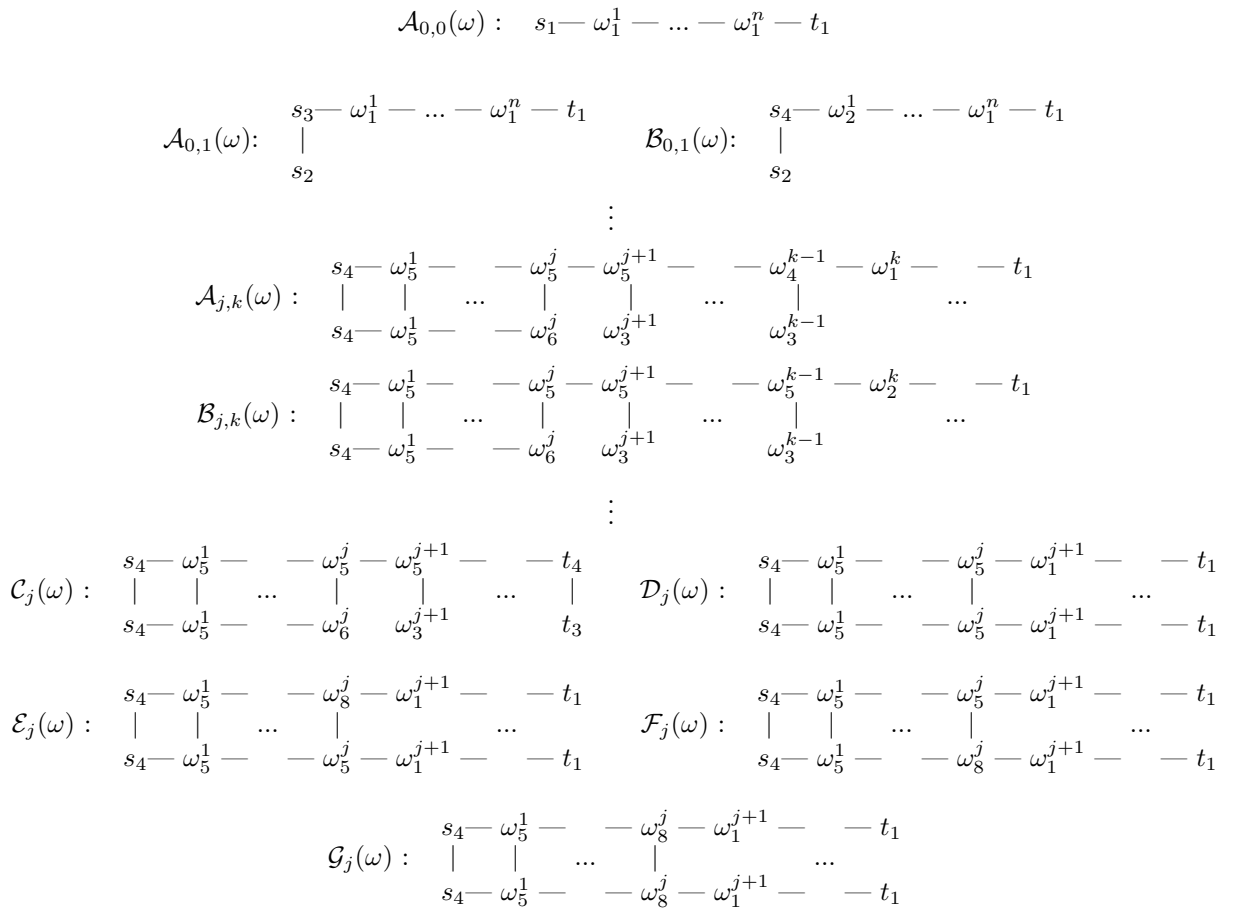

Fig. 3: A complete catalog of $\mathcal{R}(\Phi, G_0(\omega))$, excluding singletons.

vii. $\mathcal{G}_j(\omega)$ *for* $0 \leq j \leq n$

as defined in Figure 3, and the singletons $(\{1\}, \varnothing, \lambda x.z)$ *for* $z \in \Gamma_0$.

Proof: Clearly $\mathcal{A}_{0,0}(\omega)$ is reachable since it is a component of $G_0(\omega)$. The only rule that applies to $G_0(\omega)$ is AS. Write $AS = (L, R)$ where

$$\begin{aligned} V_L &= V_R = \{0,1\} \\ E_L &= \varnothing \\ E_R &= \{\{0,1\}\} \\ l_L(v) &= \begin{cases} s_0 \text{ if } v = 0 \\ s_1 \text{ if } v = 1 \end{cases} \\ l_R(v) &= \begin{cases} s_2 \text{ if } v = 0 \\ s_3 \text{ if } v = 1 \end{cases} \end{aligned}$$

Then AS is applicable via a witness $h : L \to G_0(\omega)$ where $h(1)$ equals vertex 0 of $\mathcal{A}_{0,0}(\omega)$ and $h(0)$ equals some vertex k of G (from Equation (1)) such that $p(k) = s_0$. The result of the action (AS, h) is a new graph

$$G_1 = \mathcal{A}_{0,1}(\omega) \amalg G$$

Note that $G \simeq G - \{k\}$ since there are an infinite number of vertices with each label in Γ_0. In a similar manner, we can show that every graph listed above (1) has at most two rules in Φ applicable to it and (2) yields another of the graphs listed above. For example, there exist h_1 and h_2 such that

$$\mathcal{B}_{j,k}(\omega) \amalg G \xrightarrow{A(\omega^k), h_1} \mathcal{A}_{j,k+1}(\omega) \amalg G$$

and

$$\mathcal{G}_j(\omega) \amalg G \xrightarrow{B(\omega^j), h_2} \mathcal{D}_j(\omega) \amalg G.$$

Furthermore, each graph listed can be produced via a sequence of actions applied to $G_0(\omega)$. The relationship between the reachable graphs is illustrated in Figure 4.

Note that all rule applications in any assembly sequence are between a vertex on one of the reachable graphs and a singleton labeled by a symbol in state 0; or between two vertices on the same graph. This is because: all of the "ADDING" rules use symbols in state 0 and none of the reachable graphs (except singletons) have vertices labeled with symbols in state 0; all of the "CLOSING" rules require that the vertices to be linked be in the

same component; none of the other rules add edges. The result is that, even when the state of the system contains multiple copies of any of the graphs described in Figure 3, none of them can combine via any rule to give a graph not in Figure 3. □

Notice that $\mathcal{D}_0(\omega)$ does not appear in the statement of the theorem. This is because it is not connected, consisting as it does of two copies of $\mathcal{A}_{0,0}(\omega)$. However, notice that

$$\mathcal{G}_0(\omega) \xrightarrow{BS,h} \mathcal{D}_0(\omega)$$

for exactly one h. Furthermore, BS is the *only* rule that is applicable to $\mathcal{G}_0(\omega)$. This yields the following corollary, which is the main result of the paper:

Corollary 4.1 *For any infinitely long assembly sequence*

$$G_0(\omega) \xrightarrow{r_0,h_0} G_1 \xrightarrow{r_1,h_1} ...$$

of $(G_0(\omega), \Phi)$*, the rule* BS *is used infinitely often.*

4.2 Exponential Growth

Suppose that at some point k in an assembly sequence of $(G_0(\omega), \Phi)$, the graph G_k contains n non-trivial components. By Theorem 4.1, each such component is isomorphic to one of those described in Figure 3. Also, to each component at least one rule is applicable. Since non-trivial components do not interact, these rules can all be applied in parallel. The point is that when there are n non-trivial components in an assembly sequence, then after N parallel steps there are at least $2n$ components, where N is the number of steps required to assemble from $\mathcal{A}_{0,0}(\omega)$ to $\mathcal{D}_0(\omega)$ in the worst case. (If we had not considered the rules in parallel, it would take nN sequential steps to achieve the same number of components.) We have just argued that

Theorem 4.2 *If*

$$G_0(\omega) \xrightarrow{A_0} G_1 \xrightarrow{A_1} \dot{\to} .. \xrightarrow{A_{Nk-1}} G_{Nk}$$

where $A_i = \{(r_{i,1}, , h_{i,1}), ...\}$ *is a maximal set of actions that can be applied in parallel to* G_i *and* $N(\omega)$ *is the number of steps to replicate a strand in the worst case, then the number of non-trivial components in* G_{N_k} *is* $O(2^k)$*.*

5 Related Work

The replication environment described here is clearly inspired by DNA replication [18], although we make absolutely no claims that we have modeled this considerably more complex phenomenon at any useful level of detail. John von Neumann, in the late 1940s, was the first to describe self-reproducing automata [19], which he did in the context of *Cellular Automata* (CAs). Since then, many researchers have pursued similar ideas, usually also in the context of CAs, but sometimes in other formalisms as well [15]. We believe that using the particle model described in Figure 1, instead of CAs as the basis for our replicator, makes the present work potentially applicable to a broad array of engineered systems based on, for example, tile assembly [4].

Conformational switching was first described as a symbolic process for self assembly by Saitou [13], who considered the assembly of strings in one dimension. Self assembly as a graph process is described by the author of this paper [12]). A method for using potential fields and deadlock avoidance to implement graph grammar rules with a group of mobile robots was also described [9].

Graph grammars were introduced [6, 5] at least two decades ago and have been used to describe a broad array of systems, from data structure maintenance to mechanical system synthesis. Graph grammars are, of course, a generalization of the standard "linear" grammars used in automata theory and linguistics [7] and thus (incidentally), can perform arbitrary computation.

There are other models of self assembly besides graph grammars. For example, several groups [20, 1, 2] have explored self assembly using passive *tiles*. The tiles attach along complimentary edges upon random collisions. In this setting, a form of replication has been described that creates "barcodes" from a given seed string [21]. As described elsewhere [11], graph grammars are somewhat more general and are possibly better suited to describing the *topology* of assemblies.

A simple dynamical model of the *physics* of tile assembly has been described [10]. Somewhat similar to the *reachable set* in this paper, the identification of "unique" assemblies has been explored [16]. There is also other work on tile systems with conformal-like state information [8]. The addition of simple processing to each part, similar in capability to that assumed in the present paper, is considered in models of the assembly of the T4 bacteriophage [17].

6 Conclusions and Future Work

We have presented a very basic replication environment and proved that it behaves as claimed. We believe that such a system can be used to describe the parallel assembly and self-organization algorithms that bottom-up manufacturing techniques seem to require. Certainly, more sophisticated replicators and other such systems could be designed. However, the underlying methodology of examining the structure of the reachable set will be the same for more complicated systems.

Several modifications to our system are immediately obvious. For example, one could add a rule $q_0 \quad s_0 \quad \Rightarrow q_1 - s_2$ that "blocks" that the initial step in the replication of a strand by combining a blocking particle q with the start particle. The "release" of enough particles labeled q into the system could stop replication. Other variations

are readily imagined.

Many open issues remain. First, the physical realization of graph grammar systems has not been adequately addressed. In our lab we are building macroscale (3cm diameter) programmable parts that can execute graph grammar programs. The robots float on an air table and attach or not upon random collisions. We are also designing MEMs scale parts that operate similarly in a stirred fluid. Power, of course, remains a problem. We believe these problems are purely technical, however. Second, many theoretical issues remain. For example, what happens when two rules sets are combined that interact in a non-trivial way? That is, if we want a system that performs two tasks (replication and assembly, for example) — possibly with feedback loops between the tasks — how can this reliably be achieved? Other issues involve the interplay between graph grammars (which are essentially topological) and their geometrical and physical implementations.

Acknowledgements

Many of the ideas in this paper were inspired by conversations with Robert Ghrist and Dave Lipsky. The work was supported in part by NSF Grant number #0347955.

References

[1] L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computin*, pages 740–748, Hersonissos, Greece, 2001.

[2] L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. Wilhelm P. Moisset de Espanés, and K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 23–32, Montreal, Canada, May 2002.

[3] B. Bollobás. *Modern Graph Theory*. Springer, 1991.

[4] N. Bowden, A. Terfort, J. Carbeck, and G. M. Whitesides. Self-assembly of mesoscale objects into ordered two-dimensional arrays. *Science*, 276(11):233–235, April 1997. http://www.sciencemag.org/.

[5] 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.

[6] 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.

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

[8] C. Jones and M. J. Matarić. From local to global behavior in intelligent self-assembly. In *International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[9] E. Klavins. Automatically synthesized controllers for distributed assembly: Partial correctness. In S. Butenko, R. Murphey, and P. M. Pardalos, editors, *Cooperative Control: Models, Applications and Algorithms*, pages 111–127. Kluwer, 2002.

[10] E. Klavins. Toward the control of self-assembling systems. In A. Bicchi, H. Christensen, and D. Prattichizzo, editors, *Control Problems in Robotics*, pages 153–168. Springer Verlag, 2002.

[11] E. Klavins. Directed self-assembly using graph grammars. In *Foundations of Nanoscience: Self Assembled Architectures and Devices*, Snowbird, UT, 2004. Invited Paper.

[12] E. Klavins, R. Ghrist, and D. Lipsky. Graph grammars for self-assembling robotic systems. In *Proceedings of the International Conference on Robotics and Automation*, 2004. To Appear.

[13] K. Saitou. Conformational switching in self-assembling mechanical systems. *IEEE Transactions on Robotics and Automation*, 15(3):510–520, 1999.

[14] K. Saitou and M. Jakiela. Automated optimal design of mechanical conformational switches. *Artificial Life*, 2(2):129–156, 1995.

[15] M. Sipper. Fifty years of research on self-replication: An overview. *Artificial Life*, 4(3):237–257, 1998.

[16] Y. S. Smentanich, Y. B. Kazanovich, and V. V. Kornilov. A combinitorial approach to the problem of self assembly. *Discrete Applied Mathematics*, 57:45–65, 1995.

[17] R. L. Thompson and N. S. Goel. Movable finite automata (MFA) models for biological systems I: Bacteriophage assembly and operation. *Journal of Theoretical Biology*, 131:152–385, 1988.

[18] D. Voet and J. G. Voet. *Biochemistry*. John Wiley and Sons, 3 edition, 2003.

[19] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[20] E. Winfree. Algorithmic self-assembly of DNA: Theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structure and Dynamics*, 11(2):263–270, May 2000.

[21] H. Yam, T. H. LaBean, L. Feng, and J. H. Reif. Directed nucleation assembly of dna tile complexes for barcode-patterned lattices. *Proceedings of the National Academy of Science*, 100(14):8103–8108, July 2003.