

# Graph Grammars for Self Assembling Robotic Systems

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

Robert Ghrist     David Lipsky  
Department of Mathematics  
University of Illinois  
Urbana, IL 61801  
{ghrist, dlipsky}@math.uiuc.edu

**Abstract**—In this paper we define a class of graph grammars that can be used to model and direct distributed robotic assembly or formation forming processes. We focus on the problem of synthesizing a grammar so that it generates a given, prespecified assembly. In particular, to generate an acyclic graph we synthesize a binary grammar (rules involve at most two parts), and for a general graph we synthesize a ternary grammar (rules involve at most three parts). We then show a general result that implies that no binary grammar can generate a unique stable assembly. We conclude the paper with a discussion of how graph grammars can be used to direct the synthesis of parts floating in a fluid or for self-motive robotic parts.

## I. INTRODUCTION

Engineering in the realm of the very small presents us with the daunting problem of manipulating and coordinating vast numbers of objects so that they perform some global task. Nevertheless, there are examples of sophisticated machines, such as the ribosome or the mechanical motor in the bacterial flagellum, that seem to be built *in bulk* spontaneously. One hypothesis for how this occurs is that simple small components *self assemble* into more complex aggregates which, in turn, self assemble into larger aggregates.

Our starting point in understanding self assembly is the idea of *conformational switching* [16]: Each part (molecule, robot, etc.) exists in one of several *conformations* or shapes. When two parts come into close proximity, they attach or not based on whether their conformations are complimentary. If they do attach, their conformations change (mechanically for example), thereby determining in what future assembly interactions the parts may partake.

In this paper, as in other work [15], [9], we consider the conformation of a part as corresponding to 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 the fact they are attached. An *assembly rule*, then, is a pair of such labeled graphs interpreted as follows. If a subset of parts with their labels and edges matches the first part of an assembly rule, then it may be replaced by the second part of the rule to achieve a new state of the system. We are in particular interested in (1) the situation where the parts decide in a distributed fashion whether and how to execute an assembly rule and (2) how to synthesize a set of assembly rules, called a *graph grammar*, so that only copies of a desired (prespecified) assembly are obtained from the bottom-up execution of the grammar by a set of parts.

Specifically, the contributions of this paper are the introduction of a class of graph grammars suitable for describing distributed assembly; rule synthesis procedures for trees and graphs; a theorem about the impossibility of assembling a unique stable graph using rules consisting of acyclic graphs; a discussion of how assembly rules can be implemented in a distributed fashion by simple communication protocols, leading to a notion of the algorithmic communication complexity of an assembly sequence; and, finally, the introduction of a physical model based on programmable parts floating passively in fluids that can be used to implement the above grammars and protocols.

## II. PREVIOUS AND RELATED WORK

Conformational switching was first described as a symbolic process for self assembly by Saitou [15], who considered the assembly of strings in one dimension. Self assembly as a graph process has been described by the first author of this paper [9], [10], although the graph grammar formalism is new to this paper, and a rule synthesis procedure for trees was given that is somewhat more complex than the one described here. A method for using potential fields and deadlock avoidance to implement the rules on with a group of mobile robots was also described [10].

Graph grammars were introduced [5], [4] 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 and thus (incidentally), can perform arbitrary computation. The use of graph grammars to model distributed assembly, to the best of our knowledge, is new.

There are many other models of self assembly besides graph grammars, a complete list of which is beyond the scope of this paper. But, for example, several groups [20], [3] have explored self assembly using passive *tiles* floating in liquid. The tiles attach along complimentary edges (due, for example, to capillary forces or the assembly of complimentary strands of DNA) upon random collisions. A simple dynamical model of the *physics* of tile assembly has been described [11]. Somewhat similar to the *stable set* in this paper (Definition 3.7), the identification of “unique” assemblies has been explored [17]. There is also preliminary work on supplying tile systems with conformal-like state information [8]. Such systems can in fact be used to perform arbitrary computations [19] and are best

understood as two or three dimensional symbolic processes. Another approach uses geometrical constraints on part-part interactions to model, for example, the assembly of proteins into spherical shells called *capsids* [2]. 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 [18].

The proof of Theorem 3.1 is topological, utilizing tools from covering space theory (see e.g., [7]). Our notion of commutative assembly actions and parallelization is related to the second author's work state complexes for reconfigurable robots [1], [6].

### III. GRAPH GRAMMARS FOR ASSEMBLY

#### A. Definitions

A *simple labeled graph* over an alphabet  $\Sigma$  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  $l : V \rightarrow \Sigma$  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  $G$  or by  $V$ ,  $E$  and  $l$  when there is no danger of confusion. We will usually use the alphabet  $\Sigma = \{a, b, c, \dots\}$ .

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

- 1)  $h$  is injective,
- 2)  $\{x, y\} \in E_{G_1} \Rightarrow \{h(x), h(y)\} \in E_{G_2}$ ,
- 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 3.1:* 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.

We may refer to rules as being *constructive* ( $E_L \subset E_R$ ), *destructive* ( $E_L \supset E_R$ ) or *mixed* (neither constructive or destructive). A rule is *acyclic* if its right hand side contain no cycles (the left hand side may contain cycles). Several examples of rule sets are given in Examples 3.1-3.3.

*Definition 3.2:* A rule  $r$  is *applicable* to a graph  $G$  if there exists an embedding  $h : L \rightarrow G$ . 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 3.3:* 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

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

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

*Definition 3.4:* A *graph assembly system* is a pair  $(G_0, \Phi)$  where  $G_0$  is the *initial graph* and  $\Phi$  is a set of rules (called the *rule set*).

We often refer to a system simply by its rule set  $\Phi$  and assume that the initial graph is the infinite graph defined by

$$G_0 \triangleq (\mathbb{N}, \emptyset, \lambda x.a) \quad (1)$$

where  $a \in \Sigma$  is the *initial symbol* (here  $\lambda x.a$  is the function assigning the label  $a$  to all vertices).

*Definition 3.5:* An *assembly sequence* of a system  $(G_0, \Phi)$  is a finite 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  $i \in \{0, \dots, k-1\}$ .

Thus, a system  $(G_0, \Phi)$  defines a non-deterministic dynamical system whose states are the labeled graph over  $V_{G_0}$ . The system is non-deterministic since, at any step, many rules in  $\Phi$  may be simultaneously applicable, each possibly via several witnesses. Figure 1 illustrates several assembly sequences arising from Examples 3.1-3.3.

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  $G$ . A graph  $G$  is connected if it has exactly one component.

*Definition 3.6:* A connected graph  $G$  is *reachable* in a system  $(G_0, \Phi)$  if there exists an assembly sequence  $\{G_i\}_{i=0}^k$  of  $(G_0, \Phi)$  such that  $G$  is isomorphic to some component of  $G_k$ . The set of all such reachable graphs is denoted  $\mathcal{R}(G_0, \Phi)$ , or just  $\mathcal{R}(\Phi)$  if  $G_0$  defined by (1).

*Definition 3.7:* A graph  $G \in \mathcal{R}(G_0, \Phi)$  is *stable* if for all  $G'$  there does not exist an action  $(r, h)$  on the disjoint union  $G \amalg G'$  such that  $r = (L, R) \in \Phi$  and  $h(L) \cap V_G$  is nonempty. The set of all such stable graphs is denoted  $\mathcal{S}(G_0, \Phi)$ , or just  $\mathcal{S}(\Phi)$  if  $G_0$  defined by (1).

Our focus in this paper is the problem of constructing a rule set so that a prespecified graph or graphs are the only stable graphs of the system:

#### B. Examples

*Example 3.1:* Define a constructive rule set by

$$\Phi_1 = \begin{cases} a \ a \Rightarrow b - b, \\ a \ b \Rightarrow b - c, \\ b \ b \Rightarrow c - c \end{cases}$$

We use the position of the nodes in the presentation of the rules to denote the re-labeling. For example, the first rule in  $\Phi_1$  is given by

$$\begin{aligned} L &= (\{1, 2\}, \emptyset, \lambda x.a) \\ R &= (\{1, 2\}, \{\{1, 2\}\}, \lambda x.b) \end{aligned}$$

An example assembly sequence for  $\Phi_1$  set is shown in Figure 1(a). The reachable set  $\mathcal{R}(\Phi_1)$  consists of all cycles and chains. Only cycles are stable however. This is because, once a chain closes into a cycle (using the last rule), all of its nodes are

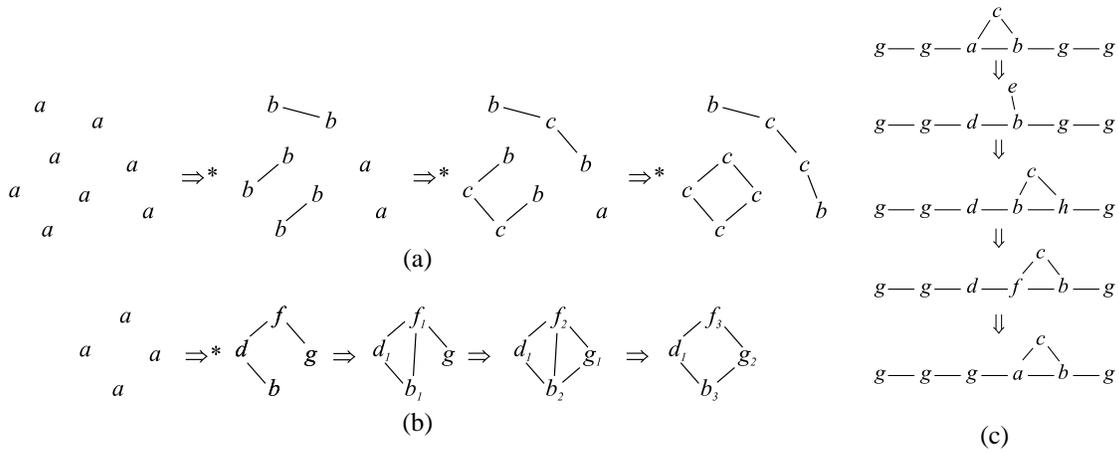


Fig. 1. An example assembly sequences arising from the rule sets defined in Examples 3.1-3.3. The symbol  $\Rightarrow^*$  denotes the application of more than one rule and, in the first transition of (a), for example, demonstrates the concurrent application of rules. (a)  $\Phi_1$  produces unstable chains and stable cycles. (b) The stable set of  $\Phi_2$  is only the four-cycle. (c)  $\Phi_3$  defines a dynamical system, wherein a part ‘ratchets’ along a substrate.

labeled by  $c$ , which does not appear in the left hand side of any rule. Thus, the stable set  $\mathcal{S}(\Phi_1)$  consists only of cycles.

*Example 3.2:* Define a mixed rule set with three binary constructive rules, two ternary constructive rules and one ternary destructive rule by

$$\Phi_2 = \begin{cases} a \ a \Rightarrow b - c, & b \begin{array}{c} d \\ \diagdown \quad \diagup \\ f \end{array} \Rightarrow \begin{array}{c} d_1 \\ \diagdown \quad \diagup \\ b_1 \end{array} f_1, \\ a \ c \Rightarrow d - e, & b_1 \begin{array}{c} f_1 \\ \diagdown \quad \diagup \\ g \end{array} \Rightarrow \begin{array}{c} f_2 \\ \diagdown \quad \diagup \\ b_2 \end{array} g_1, \\ a \ e \Rightarrow f - g, & b_2 - f_2 \Rightarrow b_3 \ f_3 \end{cases}$$

An example assembly sequence for  $\Phi_2$  is shown in Figure 1(b). The three binary constructive rules yield chains of length 4. The two constructive and cyclic ternary rules ‘‘triangulate’’ the cycle. The last rule, removes one the first triangulating edge to yield a length 4 cycle, which is the unique stable graph of the system. Theorem 3.1 implies that binary rules are insufficient to construct a unique stable cycle. In Section III-D, we describe the triangulating procedure in for general graphs.

*Example 3.3:* A rule set need not define an assembly process, as in the following set, containing constructive and destructive rules as well as mixed rules that simply re-label nodes.

$$\Phi_3 = \begin{cases} e \ g \Rightarrow c - h, \\ a - c \Rightarrow d \ e, \\ b - h \Rightarrow f - b, \\ d - f \Rightarrow g - a \end{cases}$$

An ‘‘assembly sequence’’ of  $\Phi_3$  is shown in Figure 1(c). The sequence starts with a cycle labeled with  $a$ ,  $b$  and  $c$  attached to substrate of parts labeled  $g$ . As the figure shows, the rule set ‘‘ratchets’’ the cycle along the substrate.

### C. Properties

Given a system  $(\Phi, G_0)$ , we bound the size of the reachable and stable sets using a basic topological tool: covering space

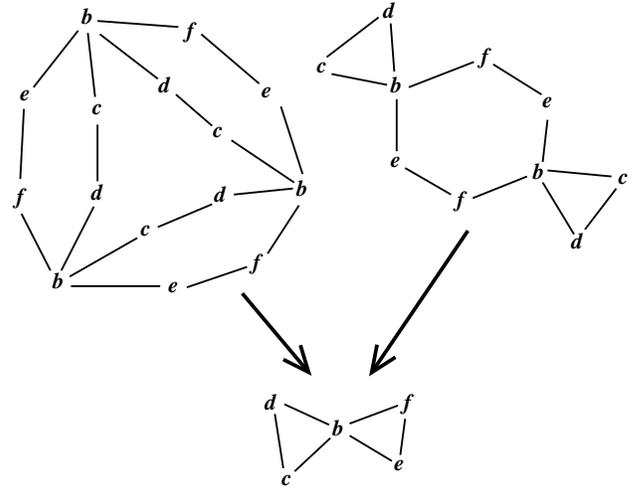


Fig. 2. Two examples (above) of covers of a simple labeled graph (below). The cover on the left is a 3-fold cover; that on the right is 2-fold.

theory.<sup>1</sup> A clear and comprehensive introduction to these classical techniques can be found in [7, pp. 60-68].

*Definition 3.8:* Given a labeled graph  $G$ , a *cover* of  $G$  is a labeled graph  $\tilde{G}$  together with a *covering projection* — a continuous map  $p : \tilde{G} \rightarrow G$  which is a local isomorphism. That is,  $p : (\tilde{V}, \tilde{E}) \rightarrow (V, E)$  preserves all labels and preserves the index of each vertex (the number of incident edges).

Examples appear in Figure 2. Note that  $p$  is not injective except in the case of a *trivial* cover in which  $\tilde{G}$  is isomorphic to  $G$ .

*Theorem 3.1:* Let  $\Phi$  denote an acyclic rule set. Then any cover of a graph in the reachable set  $\mathcal{R}(\Phi)$  is also in  $\mathcal{R}(\Phi)$ . In particular,  $\mathcal{R}(\Phi)$  has infinitely many isomorphism types of graphs if it contains any graph with a cycle.

*Proof.* Assume that  $G_K$  is the terminal element of an assembly sequence consisting of graphs  $\{G_i\}_{i=0}^K$  with corresponding actions  $\{(r_i, h_i)\}_{i=1}^K$  from the rule set  $\Phi$ . Consider any finite

<sup>1</sup>We have modified the definitions and suppressed most of the explicit topological terminology for the sake of clarity.

$$b - e - f - b - e - f$$

Fig. 3. A rule with this left hand side can de-stabilize the covers in Figure 2 even though they do not apply to the (stable) graph being covered.

covering projection  $p_K : \widetilde{G}_K \rightarrow G_K$  of the terminal graph. We will reverse the assembly sequence on  $G_K$  and lift this disassembly procedure to a sequence of disassembly steps for the cover  $\widetilde{G}_K$ . This yields an assembly sequence for  $\widetilde{G}_K$ .

For each  $i$ , the set  $h_i(R_i)$  is the subtree of  $G_i$  which is the image of the right hand side of the  $i^{\text{th}}$  rule. Since the rule set is acyclic, the *lifting criterion* [7, pp. 61-62] is automatically satisfied, and it follows that the inverse image  $\widetilde{R}_K := p_K^{-1}(R_K)$  is a disjoint union of isomorphic copies of  $R_K$ .

Replace each copy of  $R_K$  in  $\widetilde{R}_K$  with its left hand side  $L_K$ , as per Definition 3.1. Call this new graph  $\widetilde{G}_{K-1}$  and denote by  $\widetilde{L}_K$  the disjoint union of copies of  $L_K$  within  $\widetilde{G}_{K-1}$ . Define a new projection map  $p_{K-1} : \widetilde{G}_{K-1} \rightarrow G_{K-1}$  to be (1)  $p_K$  on the complement of  $\widetilde{L}_K$ ; and (2) the natural projection  $L_K \rightarrow G_{K-1}$  identifying the disjoint copies. This graph  $\widetilde{G}_{K-1}$  is clearly a cover of  $G_{K-1}$  via  $p_{K-1}$  since labels and indices are preserved.

Continue this procedure inductively, lifting  $R_i$  to  $\widetilde{R}_i$ , replacing it in parallel with copies of  $L_i$ , and then defining the projection map  $p_{i-1} : \widetilde{G}_{i-1} \rightarrow G_{i-1}$ . This terminates in a covering projection  $p_0 : \widetilde{G}_0 \rightarrow G_0$ . Since the lift of any discrete set is again a discrete set, we have that  $\widetilde{G}_0$  is the correct initial set for an assembly sequence for  $\widetilde{G}$ .

In the case where  $G$  possesses a cycle, one has that there are infinitely many non-isomorphic covers, corresponding to subgroups of the fundamental group of  $G$ : see [7, Thm. 1.38].  $\square$

The stable set,  $\mathcal{S}(\Phi) \subset \mathcal{R}(\Phi)$ , does not enjoy this property. Indeed, there may be additional rules in  $\Phi$  which have “large” connected regions of  $\widetilde{G}$  in their left hand sides. If the left hand sides are sufficiently large, then these rules can apply to covers even though  $G$  itself is inert. See Figure 3.

The following is a sample of the type of result that can be obtained using covering space theory.

**Theorem 3.2:** Assume that  $\Phi$  is an acyclic rule set, and that the stable set contains a graph  $G \in \mathcal{S}(\Phi)$  but not any of its covers. Then for each edge  $e \in E(G)$ , there exists a rule in  $\Phi$  whose left hand side contains a copy of every edge of some cycle in  $G$  passing through  $e$ .

*Proof.* Consider the 2-fold cover of  $G$  lifted along the edge  $e$ ; that is, take two copies of  $G$ , snip each copy of  $e$ , and chain them end-to-end to form the connected graph  $\widetilde{G}$ , as in Figure 2[right] using the edge  $e \text{---} f$ . One checks that this is indeed a cover of  $G$ . (The more rigorous definition is to use the correspondence theorem for covers via the fundamental group.)

Since the rule set is acyclic, Theorem 3.1 implies that  $\widetilde{G} \in \mathcal{R}(\Phi)$ . By hypothesis, this graph is not stable; hence there is some rule  $(r, h)$  applicable to  $\widetilde{G}$  but not to  $G$ . Consequently,  $p_*(h(L))$  must not be isomorphic to  $h(L)$ . From the choice

---

### Algorithm 1 *MakeTree*( $V, E$ )

---

**Require:**  $T = (V, E)$  is an unlabeled tree

```

1: if  $V = \{x\}$  then
2:   return  $(\emptyset, \lambda z . \text{if } z = x \text{ then } a \text{ else } \perp \text{ endif})$ 
3: else
4:   let  $(x, y)$  be an edge of  $(V, E)$ 
5:   let  $(V_1, E_1)$  be the component of  $(V, E - \{x, y\})$ 
      containing  $x$ 
6:   let  $(V_2, E_2)$  be the component of  $(V, E - \{x, y\})$ 
      containing  $y$ 
7:   let  $(\Phi_i, l_i) = \text{MakeTree}(V_i, E_i)$  for  $i = 1, 2$ 
8:   let  $u, v$  be new labels
9:   let  $\Phi = \Phi_1 \cup \Phi_2 \cup \{l_1(x) \ l_2(y) \Rightarrow u - v\}$ 
10:  let  $l = \lambda z .$ 
11:    if  $z = x$  then  $u$ 
12:    else if  $z = y$  then  $v$ 
13:    else if  $l_1(z) \neq \perp$  then  $l_1(z)$ 
14:    else  $l_2(z)$ 
15:  return  $(\Phi, l)$ 
16: end if

```

---

of cover  $\widetilde{G}$ , we conclude that  $p_*(h(L))$  contains a loop in  $G$  passing through the edge  $e$ .  $\square$

**Corollary 3.1:** If  $\Phi$  is an acyclic rule set all of whose left hand sides are vertex sets, then the stable set is closed under covers.

Additional results and extensions to non-acyclic rules are possible if one carefully tracks cycles. A more detailed analysis will appear in future work. In particular, since rules with large vertex sets require high levels of communication to assemble (see Section IV), one could hope for topological bounds on the maximal amount of communication required to build a unique stable graph.

#### D. Synthesis Algorithms

In this section we consider the problem: Given a graph  $G$ , find  $\Phi$  such that  $\mathcal{R}(\Phi) = \{G'\}$  and  $G' \simeq G$ .

#### E. Trees

We define in Algorithm 1 a recursive function *MakeTree* that, given any tree  $T$ , produces a set of binary rules  $\Phi_T$  so that  $\mathcal{S}(\Phi_T) = \{T'\}$  where  $T' \simeq T$ , not considering labels. *MakeTree* takes as an argument an unlabeled tree  $(V, E)$  and returns a pair  $(\Phi, l)$  where  $\Phi$  is a rule set and  $l$  is a labeling function on  $V$ . The base case (lines 1-2) labels the single point with the label  $a$ . For the recursive step, an edge  $(x, y)$  is chosen and *MakeTree* is called recursively on the two tree components that result from removing  $(x, y)$  from  $E$ . The recursive calls return rule sets  $\Phi_1$  and  $\Phi_2$  and labeling functions  $l_1$  and  $l_2$ . The rule set  $\Phi$  for  $(V, E)$  is constructed from these in line 9 and the labeling function  $l$  is constructed in lines 10-13. The construction uses two new labels  $u$  and  $v$  that we suppose have not been used before by any recursive call to *MakeTree*.

**Theorem 3.3:** Let  $(\Phi, l) = \text{MakeTree}(V, E)$  where  $(V, E)$  is an unlabeled tree. Then  $\mathcal{S}(\Phi) = \{T\}$ .

*Proof:* By general induction on the definition of *MakeTree*.  
□

The algorithm *MakeTree*( $V, E$ ) produces exactly  $|V| - 1$  rules and uses  $2|V| - 2$  labels.

#### F. Arbitrary Graphs

Given a graph  $G$ , Algorithm 2 defines a function *MakeGraph* that produces a rule set  $\Phi_G$  such that  $\mathcal{S}(\Phi_G) = \{G\}$  (with a new label scheme) using rules which are *at most* ternary. A priori, this would appear to be difficult, given the constraints of Theorem 3.2. The ingredients which make this low-communication synthesis possible include using rules which may be both cyclic and deconstructive: see the discussion of Section IV.

The function *MakeGraph* takes as input an unlabeled graph  $(V, E)$  and produces a pair  $(\Phi, l)$  where,  $\Phi$  is a rule set and  $l$  a labeling function on  $V$ . We begin in lines 1-2 by finding a maximal spanning tree  $T = (V, E_T)$  for  $(V, E)$ , and constructing a rule set  $\Phi$  for this tree using *MakeTree*. We let  $r$  denote the last vertex of  $G$  visited while executing *MakeTree*.

The loop in lines 6-14 adds rules to  $\Phi$  that, in effect, triangulate the spanning tree until the resulting graph contains a subgraph isomorphic to  $G$ . At each step,  $E_S$  denotes the edges that have been used in this triangulation. In other words, at all times we have  $\mathcal{S}(\Phi) = \{(V, E_S)\}$ .

The triangulation proceeds as follows. For each edge  $\{v, v'\}$  in  $E - E_T$ , we find a minimal path in  $(V, E_S)$  from  $r$  to  $v$ , and append rules that add edges from  $r$  to each successive vertex along this path. We then update  $E_S$  to include these edges, and then repeat the procedure for a minimal path from  $r$  to  $v'$ . Finally, we add an edge from  $v$  to  $v'$ . At each step, we change the label on the root node, ensuring that the rules can only be applied in the order given. The triangulation for one edge cannot begin until the triangulation for the previous one is complete. This is especially important for the implementation of the final step of the algorithm. After the loop on lines 6-14 has finished, the resulting rule set will have a single graph in its stable set, and this graph will contain a subgraph isomorphic  $G$ . The rules added by the loop on lines 15-20 serve to remove the excess edges between the root node  $r$  and the other vertices. The left side of each of these rules contains a label that only appears on the root node after the entire triangulation process is complete. Thus the removal of excess edges can begin only after the triangulation is complete. In effect, the label on the root node tracks the progress towards completion of the triangulation. After all the rules given in lines 15-20 have been applied, the resulting graph will be isomorphic to  $G$ . As shown above, this graph will be the only element of the stable set for  $\Phi$ .

This discussion constitutes the following:

*Theorem 3.4:* Let  $(\Phi, l) = \text{MakeGraph}(V, E)$  where  $G = (V, E)$  is an unlabeled graph. Then  $\mathcal{S}(\Phi) = \{G\}$ .

## IV. COMMUNICATION

In this section we consider the amount of communication required to implement an assembly rule or a set of assembly rules. We consider as a basic unit of communication, the

---

### Algorithm 2 *MakeGraph*( $V, E$ )

---

**Require:**  $G = (V, E)$  is an unlabeled connected graph

```

1: let  $T = (V, E_T)$  be a maximal spanning tree of  $(V, E)$ 
2: let  $(\Phi, l) = \text{MakeTree}(V, E_T)$ 
3: let  $r$  be the last vertex visited while executing MakeTree
4: let  $a = l(r)$ 
5: let  $E_S = E_T$ 
6: for all  $e = \{v, v'\} \in E - E_T$  do
7:   let  $(\Psi, E_S, a) = \text{Triangulate}(V, E_S, r, v, l, a)$ 
8:   let  $(\Psi', E_S, a) = \text{Triangulate}(V, E_S, r, v', l, a)$ 
9:   let  $b$  be a previously unused label
10:  let  $\psi$  be the rule denoted in Figure 4(a)
11:  let  $\Phi = \Phi \cup \Psi \cup \Psi' \cup \psi$ 
12:  let  $E_S = E_S \cup \{\{v, v'\}\}$ 
13:  let  $a = b$ 
14: end for
15: for all  $v \in V$  do
16:   if  $\{r, v\} \in E_S - E$  then
17:     let  $\psi$  be the rule  $a - l(v) \Rightarrow a \ l(v)$ 
18:     let  $\Phi = \Phi \cup \psi$ 
19:   end if
20: end for
21: return  $(\Phi, \lambda z. \text{if } z = r \text{ then } a \text{ else } l(z) \text{ endif})$ 

```

**Subroutine:** *Triangulate*( $V, E, r, v, l, a$ )

```

1: let  $\Psi = \emptyset$ 
2: let  $(v_1, \dots, v_n) = \text{MinimalPath}(V, E, r, v)$ 
3: for  $i = 1$  to  $n - 1$  do
4:   let  $b$  be a previously unused label
5:   let  $\psi$  be the rule denoted in Figure 4(b)
6:   let  $\Psi = \Psi \cup \{\psi\}$ 
7:   let  $E = E \cup \{\{r, v_{i+1}\}\}$ 
8:   let  $a = b$ 
9: end for
10: return  $(\Psi, E, a)$ 

```

---

cost of executing a binary rule (of any type) in a distributed environment. Of course, in certain circumstances the cost of a binary rule could be further subdivided into, for example, the time spent transmitting a request to execute a rule and the time spent waiting for an acknowledgment. We do not do this here, but suppose that binary rules are the fundamental building block of bigger rules.

To reduce a larger rule to a set of binary rules within the graph grammar system we have proposed and expect to get the same stable set is not possible, as Corollary 3.1 implies. We can, however, augment the graph grammar system by supposing that each part is able to store an *unique id* number and possibly other information.

#### A. Binary Rules for Ternary Rules

Using an infinite set of symbols and an infinite set of rules, we can, for example, implement ternary rules with a set of binary rules.

To proceed, we start with an alphabet  $\Sigma$ , and introduce a new, infinite set of labels of the form  $x : i : S$  where  $x \in \Sigma$ ,  $i \in \mathbb{N}$  and  $S \subset \mathbb{N}$ . The symbol  $x$  denotes the label of the part with unique “address”  $i$ . The set  $S$  consists of a set of other

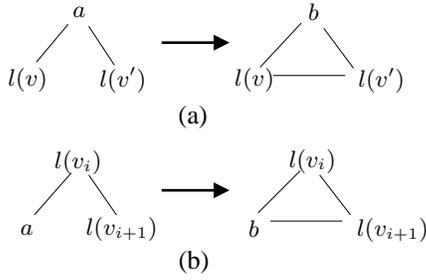


Fig. 4. The two ternary rules used in Algorithm 2.

part addresses which have committed to executing a given ternary rule<sup>2</sup>. For example, suppose we wish to implement, with binary (and unary) rules, the ternary rule  $r$  defined by

$$\begin{array}{c} a \\ \swarrow \quad \searrow \\ c \quad b \end{array} \Rightarrow \begin{array}{c} a' \\ \swarrow \quad \searrow \\ c' \quad b' \end{array} . \quad (2)$$

We can do so with the following rules:

$$\Phi_r = \left\{ \begin{array}{l} b : k : \emptyset - a : j : \emptyset \Rightarrow b : k : j - a : j : k \\ c : i : \emptyset - a : j : \emptyset \Rightarrow c : i : j - a : j : i \\ b : k : \emptyset - a : j : i \Rightarrow b : k : j - a : j : ik \\ c : i : \emptyset - a : j : k \Rightarrow c : i : j - a : j : ik \\ c : i : j \quad b : k : j \Rightarrow c : i : jk - b : k : ij \\ a : j : ik \Rightarrow a' : j : \emptyset \\ b : k : ij \Rightarrow b' : k : \emptyset \\ c : i : jk \Rightarrow c' : i : \emptyset \end{array} \right.$$

We have, in fact, specified an infinite set of rules: Eight for each triple  $(i, j, k) \in \mathbb{N}^3$ , with  $i, j, k$  distinct.

The set  $S$  in the symbols denotes the state of a protocol for executing  $r$ . For example, the symbol  $b : k : \emptyset$  is the state of a part  $k$  labeled by  $b$  that has not started to execute  $r$ . The symbol  $b : k : j$  is the state of a part  $k$  that has initiated execution of  $r$  by communicating with a part  $j$  labeled by  $a$ . The symbol  $b : k : ij$  denotes a part that has proceeded through the execution of  $r$  by communicating with parts  $i$  and  $j$  labeled by  $a$  and  $c$  respectively. Notice that in the fifth rule, where the edge between  $c$  and  $b$  is actually added, we require that  $c$  and  $b$  are connected to the same part  $j$  labeled with  $a$ . This restriction is possible since the parts have unique addresses. In particular, it follows that

$$\mathcal{R}(\Phi_r) = \mathcal{R}(\{r\}) \quad \text{and} \quad \mathcal{S}(\Phi_r) = \mathcal{S}(\{r\}).$$

This does not contradict Theorem 3.1 since, by assigning unique addresses to parts, we violate the hypotheses of having an infinite supply of identically labeled vertices as the start graph  $G_0$ .

### B. Parallelism and Communication Cost

The number of binary rules in a reduced rule set does not correspond directly to the communication cost of the set, however. One must take into account the *parallelizability* of the set as well. In this section we define the cost of an assembly  $G \in \mathcal{R}(\Phi)$  as the longest parallel assembly sequence of  $G$ .

<sup>2</sup>We use the notation  $ijk$  (or  $jki$ , etc) to represent the set  $\{i, j, k\}$ .

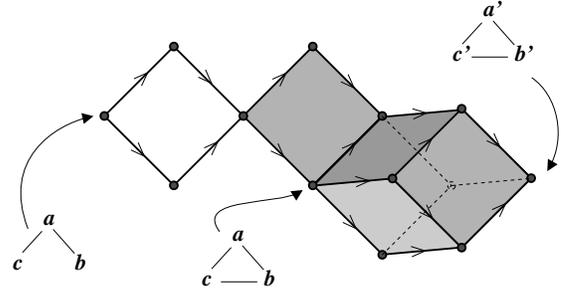


Fig. 5. The *state complex* [1], [6] coordinates all possible assembly sequences of (2) using the binary rules in  $\Phi_r$ . The 2-dimensional squares correspond to commutative pairs of rules; the 3-dimensional cube corresponds to a commutative triple.

**Definition 4.1:** A set  $\{(r_i, h_i)\}_{i=1}^k$  of actions is *commutative* if

$$h_i(V_{L_i}) \cap h_j(V_{L_j}) = \emptyset \quad \forall i \neq j.$$

**Definition 4.2:** Given an assembly sequence  $\sigma = \{G_i\}_{i=0}^k$  produced by  $\{(r_i, h_i)\}_{i=1}^k$ , a *parallelization* of  $\sigma$  is a finite increasing sequence  $j_0, \dots, j_{N-1}$ , where  $j_0 = 1$  and  $j_{N-1} = k$ , such that the sets

$$A_i \triangleq \{(r_m, h_m) \mid j_i \leq m < j_{i+1}\}$$

are commutative. A shortest parallelization is called *minimal* and the *communication cost* of  $\sigma$  is defined to be the length of a minimal parallelization of  $\sigma$ .

In the next definition, we consider all assembly sequences **for** a given graph  $G \in \mathcal{R}(\Phi)$ , in the sense that  $V_{G_i} = V_G$  for each  $G_i$  in the assembly sequence<sup>3</sup>.

**Definition 4.3:** Given a rule set  $\Phi$  and an assembly  $G \in \mathcal{R}(\Phi)$ , the worst case communication cost of  $G$  with respect to  $\Phi$  is the maximum cost of all assembly sequences **for**  $G$ .

One can construct a transition graph representing all possible assembly sequences for a given graph: The nodes are sub-assemblies and the edges correspond to single actions taking one subassembly to the next. In [14] it was noted that such a transition graph is the 1-d skeleton of a *cubical complex*, where higher dimensional cubes correspond to parallel assembly steps. The geometry of such complexes has recently been explored in [6], [1]: geodesics correspond to time-optimal solutions. In Figure 5 we illustrate this construction with the right hand side of the rule  $r$  in (2) with respect to the set of binary rules  $\Phi_r$ . Vertices represent assembly states, edges represent the application of a single rule, and higher-dimensional cubical cells identify all commutative sets of actions. This diagram shows that the communication cost of that particular graph is four, since each possible assembly sequence using the eight rules in  $\Phi_r$  can be compressed into four parallel steps.

We plan to use Definitions 4.2 and 4.3 in future work, examining the communication cost of synthesis algorithms (such as those in Section III-D) and characterizing a graph in terms of the communication cost of the minimal rule set that uniquely produces it.

<sup>3</sup>If  $G$  is obtained by building a “scaffolding” and then deleting certain parts, we consider those parts as part of  $G$ , dropping the requirement that  $G$  be connected in Definition 3.6.

## V. PHYSICAL MODELS OF ASSEMBLY

### A. Self-Motive Robots

In previous work [10] we showed how to use graph grammars as a basis for multi-robot formation forming (although we did not recognize our formalism as grammatical at the time). To each robot  $i$  we associate two sets: The set  $\mathcal{A}_i$  of robots  $j$  such that either  $i$  is “attached” to  $j$  or  $l(i)$   $l(j)$  matches a rule in  $\Phi$ , and the set  $\mathcal{R}_i$  of the robots where no rule matches. We then define an *artificial potential function*  $U_i$  of the form

$$U_i = \sum_{j \in \mathcal{A}_i} U_{\text{attract}}(x_i, x_j) + \sum_{j \in \mathcal{R}_i} U_{\text{repel}}(x_i, x_j)$$

where  $x_k$  is the position of robot  $k$ . The function  $U_{\text{attract}}$  is defined so that  $-\nabla_{x_i} U(x_i, x_j)$  has the set  $\|x_i - x_j\| = R$  as a global attractor. The  $U_{\text{repel}}$  is defined so that  $-\nabla_{x_i} U(x_i, x_j)$  has the set  $\|x_i - x_j\| = R$  as a global repeller. Here  $R$  is the desired distance between “attached” robots. Each robot  $i$  simply follows the negative gradient of  $U_i$  to move toward robots in  $\mathcal{A}_i$  and away from robots in  $\mathcal{R}_i$ . When two attracting robots come in close proximity, they execute the appropriate rule and change their states. This has the effect of changing  $\mathcal{A}_k$  and  $\mathcal{R}_k$ , and therefore  $U_k$ , for each  $k$ .

If the number of robots is finite, then deadlock can occur in the above system due to groups forming incompatible subassemblies. We thus add the rule

$$(V, E, l) \Rightarrow (V, \emptyset, \lambda x.a)$$

for each non-final subassembly of the goal assembly. Robots randomly, with low probability, choose to disassociate using this rule. Note that this rule may be large. We do not at this time have a general construction for turning large rules into a set of infinite binary rules, as we did for ternary rules in Section IV.

### B. Stirred Robotic Parts

We now introduce a *new* model of “robotic” assembly that requires less capable robots, in that they do not need to be able to move. Instead we suppose that large number of robotic parts *float* in a stirred fluid. Upon colliding (by chance), two parts will latch onto each other or not based on whether their current labels match the left hand side of a rule in  $\Phi$ . If they do latch together, then they change their labels according to the rule. A similar scheme works for mixed or destructive rules.

To model this system, we suppose that each part  $i$  with position  $x_i$  has a latch variable  $L_{i,j} \in \{0, 1\}$  associated with every other robot  $j$ . When the parts come in close proximity, they communicate their current labels to each other in an attempt to find an applicable rule<sup>4</sup>. If one is found, they both set their latch variable for the other to 1 and change their labels. Otherwise the latch variable is set to 0 and the parts bounce off each other. The dynamics of robot  $i$  are

$$m\ddot{x} = F_1(x_i, t) - c\dot{x}_i + \sum_{j \neq i} L_{i,j} L_{j,i} F_2(x_i, x_j) \quad (3)$$

where  $F_1$  is a time varying force field that models the effect of the fluid on the part and

$$F_2(x_i, x_j) \triangleq -\nabla_{x_i} U(x_i, x_j) - b \frac{\dot{x}_i(x_j - x_i)}{\|x_j - x_i\|} (x_j - x_i)$$

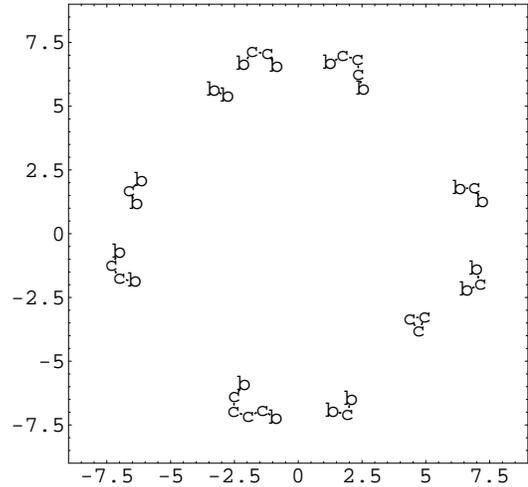
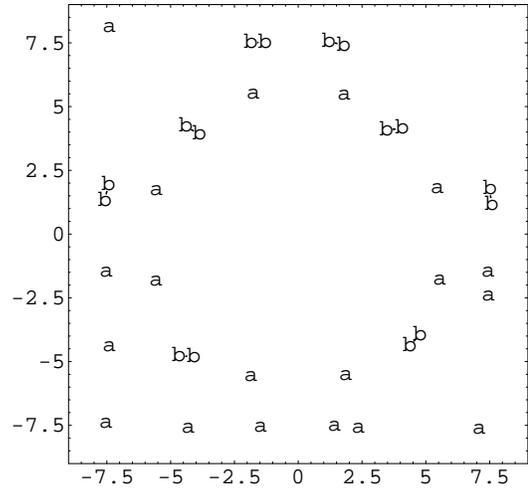
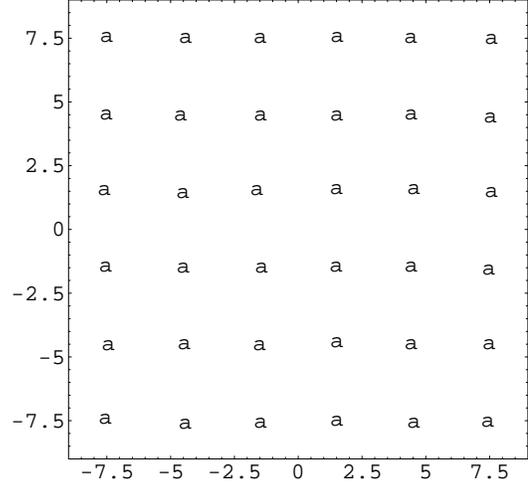


Fig. 6. Snapshots of a simulation of Equation 3 with 36 disk-shaped parts and a complete communications protocol implementing the rules in Example 3.1. The location of the part and its label is shown. The force due to stirring is modeled as the sum of five divergent force fields focused in different locations on the plane (which may result from jets perpendicular to the plane of the figures). The magnitudes of the components of the field oscillate out of phase. The parts start out even distributed and eventually form chains and cycles. After enough time, only cycles will be present (not shown).

is a damped spring, with spring potential  $U$ , modeling the latching mechanism. We suppose that  $U$  has a minimum at  $\|x_i - x_j\| = R$  as before. In other work [11] we describe, for example, how such a mechanism would work using capillary forces.

We have explored the behavior of the above model in simulation using the rule systems we explored in this paper. Figure V-B shows the evolution of a system of parts using the rules in Example 3.1. We plan to report on quantitative properties of this model in future work.

## VI. DISCUSSION

We have defined a class of graph grammars that describe self assembling graph systems. We focused on the properties of the reachable and stable graphs that rule sets produce. We noted in Section III-C that some rules (those without cycles in their right hand sides) are in a sense less powerful than cyclic rules, because acyclic rules cannot produce a unique stable cyclic graph while cyclic rules can. We then described two algorithms that can be used to synthesize rule sets for arbitrary trees and graphs. Indeed our algorithm for arbitrary graphs uses cyclic rules. In Section IV we introduced infinite rule sets that can, with binary rules, implement ternary cyclic rules. To work around Corollary 3.1, we added a unique identifier to each node, in a similar fashion to symmetry breaking methods in found in distributed algorithms [13]. Finally, we discussed how graph grammar rules can be used as a basis for robot assembly by describing two physical models appropriate to the task.

The methods we propose show that algorithms for collective tasks, here distributed self-assembly of a prespecified structure, can be *engineered*. The model is one of controlling the pairwise (or subgroup-wise) local interactions of a system so that global properties result. We believe that similar methods will be applicable to other systems, as may possibly someday provide a predictive model for distributed tasks observed in nature.

There are many aspects of the self-assembly model described here that we plan to explore more completely in the future: We showed how to produce binary infinite rules from ternary additive rules, but have not devised a procedure for this in general. We have described algorithms for producing rule sets that produce a desired graph as uniquely stable, but have not shown how to produce rule sets that have minimal communication cost (in the sense defined in Section IV). We have not defined an effective method for determining the communication cost (or the state complex) of a given reachable graph.

Finally, there are many aspects of natural and artificial self-assembly that the graph assembly model described here does not capture. For example, our model is not geometrical. But geometry is crucial in most self-assembly processes from meso-scale tile assembly to the formation of supra-molecular aggregates. Also, even if it may somehow be possible to encode a finite set of labels in the *conformation* of a part, the

infinite set of identification symbols we propose to recover binary rules is unreasonable. However, a set of appropriately shaped parts (using a graph grammar or not) can easily form a uniquely stable, cyclic graph. The interplay between symbolic descriptions of self-assembly and geometric descriptions will be a main focus of or future research in this area.

## Acknowledgments

Klavins is supported in part by AFOSR grant number F49620-01-1-0361. Ghrist is supported in part by NSF CAREER grant number DMS-0337713. Lipsky is supported in part by NSF VIGRE grant number DMS-9983160.

## REFERENCES

- [1] A. Abrams and R. Ghrist. State complexes for metamorphic robot systems. To appear, *Intl. J. Robotics Research*.
- [2] B. Berger, P.W. Shor, L. Tucker-Kellogg, and J. King. Local rule-based theory of virus shell assembly. *Proceedings of the National Academy of Science, USA*, 91(6):7732–7736, August 1994.
- [3] 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/>.
- [4] B. Courcelle. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, 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. Ghrist. Shape complexes for metamorphic robot systems. In *Workshop on the Algorithmic Foundations of Robotics*, December 2002.
- [7] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2001.
- [8] C. Jones and M. J. Mataric. From local to global behavior in intelligent self-assembly. In *International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.
- [9] E. Klavins. Automatic synthesis of controllers for distributed assembly and formation forming. In *Proceedings of the IEEE Conference on Robotics and Automation*, Washington DC, May 2002.
- [10] 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.
- [11] 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.
- [12] E. Klavins. A formal model of a multi-robot control and communication task. In *42nd IEEE Conference on Decision and Control*, Maui, HI, December 2003.
- [13] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
- [14] V. Pratt. Modeling concurrency with geometry. In *Proc. 18th ACM Symposium on Principles of Programming Languages*, 1991.
- [15] K. Saitou. Conformational switching in self-assembling mechanical systems. *IEEE Transactions on Robotics and Automation*, 15(3):510–520, 1999.
- [16] K. Saitou and M. Jakiela. Automated optimal design of mechanical conformational switches. *Artificial Life*, 2(2):129–156, 1995.
- [17] Y. S. Smentanich, Y. B. Kazanovich, and V. V. Kornilov. A combinatorial approach to the problem of self assembly. *Discrete Applied Mathematics*, 57:45–65, 1995.
- [18] 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.
- [19] H. Wang. Notes on a class of tiling problems. *Fundamenta Mathematicae*, pages 295–305, 1975.
- [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.

<sup>4</sup>A complete communications protocol for implementing binary rule sets (finite or infinite) can be formally defined using tools for modeling concurrency [12] for example. A complete description of the protocol we have in mind is beyond the scope of this paper, however.