Proceedings of the 45th IEEE Conference on Decision & Control
Manchester Grand Hyatt Hotel
San Diego, CA, USA, December 13-15, 2006

Collective Sensing with Self-Organizing Robots

Joshua Bishop Eric Klavins

Department of Electrical Engineering
University of Washington
Seattle, WA 98195
{pushpin, klavins}@u.washington.edu

Abstract— **We introduce a way to compose two local rule sets to form an environmental sensor. Each set of local rules results in a different global behavior when interpreted by a set of interacting programmable particles. In the composed system, the particles choose which set of rules to use depending on whether or not a certain condition is true or false about the initial state of the system. The global behavior of the system eventually matches only one set of rules, signaling that the particles have collectively *recognized* the condition on the initial state. We demonstrate the composition method on our robotic testbed.**

I. Introduction

Self-organization is the phenomenon by which many locally interacting particles form global behaviors. It is ubiquitous in nature. For example, virus capsids, cell-membranes and tissues are all self-assembled from smaller components in a completely distributed fashion and the processes they form are the result of local interactions. Self-organization is beginning to find its way into engineering, harnessed using a variety of technologies such as DNA [1], [2], PDMS [3], MEMs [4] and robots [5], [6], [7].

We are particularly interested in self-organizing systems that are *programmed* by local interaction rules. The rules state what the result of a local interaction among particles should be in terms of the local internal states of the particles and the connection topology they form. The formal tool we use to describe an analyze these systems is the *graph grammar* [8], which we have used to describe robotic self-assembly [5], [6], [9], self-replication [10] and cooperative control [11].

Complex self-organized behaviors, described by graph grammars or otherwise, are difficult to engineer due to their massively concurrent nature and vast irregular state spaces. To attempt to harness this complexity, we are exploring methods with which to *compose* local-rule-based behaviors to form composite systems. The idea is that a complex rule set with easily determined properties can be formed from the composition of many simpler ones with known properties. This can be done in a variety of ways. One may form a system that first uses one set of local rules and then another. This is not necessarily straightforward since it is difficult for the participants to decide using only local interactions when to start using the second set of rules. Furthermore, the global properties resulting from the combination of two rule sets may be difficult to determine.

In this paper, we describe one particular composition, the *boolean recognizer*. The construction takes two *output grammars* and results in a system that behaves like one or the other depending on whether certain environmental factors are present in the initial condition. We describe the grammatical formalism, introduce the construction, give examples and prove the construction results in the expected global behavior. Finally, we demonstrate the construction with several examples designed for our self-organizing *programmable-parts* robotic testbed [5], [6].

II. Related Work

Environmental sensor networks based on systems of large numbers of autonomous robots have been demonstrated for many types of tasks, such as localization and navigation [12], exploration [13], and chemical plume sensing [14]. In addition, methods for the decentralized coordination of autonomous agents to achieve behavioral consensus have also been previously demonstrated [15], [16].

However, whereas the methods described in [15], [16] lead to an average behavior that is dependent on the initial states of the agents, the method described in this paper leads to one of two behaviors, which is determined by initial environmental conditions. In this sense, our method derives more from the notion of consensus as defined in computer science [17] than as defined in the control systems literature. The notion of composition as used here is also grounded in computer science, where provably correct methods for the composition of programs are of great interest [18].

III. Definitions

This section contains the basic definitions of graph grammars and introduces the idea of a boolean recognizer. The notion of graph grammar used here is almost identical to that defined in [8], except for the use of *unary rules* defined below.

A. Graphs

We use *labeled graphs* to describe the state of a self-assembling system. In this formalism, vertices represent the parts of a system which may be, for example, robots, molecules or particles. Edges represent a connection of some kind between parts formed by, for example, a communication link, a dynamic coupling or a chemical bond.

1-4244-0171-2/06/$20.00 ©2006 IEEE.

Formally, a *labeled graph* over an alphabet Σ is a triple $G = (V, E, l)$ where (V, E) is a graph and $l : V \to \Sigma$ is a labeling function. We refer to labeled graphs as simply *graphs* for the rest of this paper. We refer to the vertices, edges and labels of a graph by V_G, E_G and l_G. The label $l(v)$ of a part v corresponds to its *internal discrete state* and will be used to keep track of local information.

B. Grammars

A *graph grammar* consists of a set Φ of rules. Each rule r is of the form $L \rightharpoonup R$ where L and R are labeled graphs over some small vertex set $V_L = V_R$.

For example, the rule

$$a \quad a \rightharpoonup b - b$$

rewrites two unconnected parts labeled a as two connected parts labeled b. This is an example of a *binary* rule since $|V_L| = |V_R| = 2$. Notice that in the general definition of a rule, there can be any number of vertices and that the *context* does not matter (the parts involved can be connected to any number of other parts, for example).

We will find it useful to define a special type of context-dependent unary rule of the form $L \overset{\circ}{\rightharpoonup} R$ that rewrites the label of a single vertex v as long it has degree $d(v) = 0$. For example, the rule

$$\omega \overset{\circ}{\rightharpoonup} a$$

rewrites a part having label ω having no neighbors so that it has the label a. The use of a different arrow ($\overset{\circ}{\rightharpoonup}$) highlights the requirement that $d(v) = 0$.

We make use of the several standard concepts in graph theory to describe the way in which rules rewrite labeled graphs. If G and H are graphs, we write graph *isomorphism* $G \simeq H$, which implies a *witness* h that maps vertices in G to vertices in H while preserving edges and labels. If $U \subseteq V$, we write the graph induced by U as $G|_U$. We write a sequence of graphs $\{G_i\}_{i\in\mathbb{N}}$.

A *system* (G_0, Φ) consists of an initial graph G_0 and a set of rules Φ. We usually assume that $E_{G_0} = \varnothing$. Given a system (G_0, Φ), a sequence

$$\{G_i\}_{i\in\mathbb{N}}$$

is a *non-deterministic system trajectory* if for all i, there exists a rule $r \in \Phi$ such that:

i. If $r = L \rightharpoonup R$, there exists a set of distinct vertices $U \subseteq V_{G_0} = V$ such that
   a. $G_i|_U \simeq L$ and $G_{i+1}|_U \simeq R$ via the same witness h.
   b. $G_i|_{V-U} = G_{i+1}|_{V-U}$.
   c. If $uv \in E_{G_i}$ with $u \in U$ and $j \in V - U$ then $uv \in E_{G_{i+1}}$ and conversely.

ii. If $r = L \overset{\circ}{\rightharpoonup} R$, there exists a vertex $u \in V$ such that
   a. $d_i(u) = 0$
   b. $G_i|_u \simeq L$ and $G_{i+1}|_u \simeq R$
   c. $G_i|_{V-u} = G_{i+1}|_{V-u}$.

If the sequence is finite, then we require that there is no rule in Φ applicable to the terminal graph. If G' is obtained from G via the rule r and witness h, we write $G \xrightarrow{r,h} G'$.

The set of all graphs reachable from G_0 via some trajectory is called the *reachable set* $\mathcal{R}(G_0, \Phi)$. The set of all connected components of graphs in $\mathcal{R}(G_0, \Phi)$ up to isomorphism is denoted $\mathcal{C}(G_0, \Phi)$. If $C \in \mathcal{C}(G_0, \Phi)$, then C is called *stable* if its vertices can not participate in any rule application in any trajectory in which it appears. The set of all stable components is denoted $\mathcal{S}(G_0, \Phi)$.

C. Labelings

The construction of a boolean recognizer grammar is best expressed using labels coming from a cartesian product of alphabets. This allows us to use labels as a type of data structure, with each alphabet encoding a separate type of local information.

Given a graph G with a labeling function $l_G : V \to \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_r$, the function $l_{G,i} : V \to \Sigma_i$ returns the i^{th} part of the label of a vertex in G. We define the i^{th} *projection* of a graph G by

$$\pi_i(G) = (V_G, E_G, l_{G,i}).$$

Also, if $\mathcal{G}$ is a set of graphs, then $\pi_i(\mathcal{G})$ denotes the set $\{\pi_i(G) \mid G \in \mathcal{G}\}$.

In the rest of this paper, we use the alphabet $\Sigma_1 = \{a, b, c, ...\}$ of letters and the alphabet $\Sigma_2 = \mathbb{N}$ of natural numbers. The boolean recognizer grammar uses the alphabet $\Sigma_1 \times \Sigma_2$ and we write, for example, a_5 instead of $(a, 5)$, for convenience.

If $G = (V, E, l)$ is a graph and $f : V \to \mathbb{N}$, we define the graph $G[f] = (V, E, l')$ by $l'(v) = (l(v), f(v))$. If $r = L \rightharpoonup R$ then $r[f]$ is the rule $L[f] \rightharpoonup R[f]$.

D. Boolean Recognizers

We begin with two *output grammars* Φ_0 and Φ_1 over the alphabet Σ_1 with $\mathcal{R}(\Phi_1) \neq \mathcal{R}(\Phi_2)$. We suppose that these grammars each have a distinctive behavior starting from the discrete graph containing no edges and having all vertices labeled by a. For example, one grammar might form stable chains of length three while the other forms stable cycles of length four.

Next, we consider the second alphabet Σ_2 as input to a boolean function $Q : \mathbb{N} \to \{0, 1\}$. Assuming that $m \in \mathbb{N}$ is expressed as a binary number, we use the bits of m to represent the presence different "substances". For example, $(001)_2$ represents the presence of substance 1, and $(101)_2$ represents the presence of both substances 1 and 3.

The boolean recognizer construction operates on initial graphs G_0 having no edges and in which each vertex is labeled $(a, i) = a_i$ for some $i \in \mathrm{dom}(Q)$. The goal is to define a composition of Φ_0 and Φ_1 so that the system ultimately behaves like Φ_i if and only if

$$Q\left(\bigvee_{v\in V_{G_0}} l_2(v)\right) = i,$$

where $\bigvee$ represents *bitwise or*. In particular, we will construct a grammar $\Phi_Q(\Phi_0, \Phi_1)$ such that if the above is true, then

$$\pi_1(\mathcal{S}(G_0, \Phi_Q)) = \mathcal{S}(\pi_1(G_0), \Phi_i).$$

If Φ_Q has the above property, then it is called a *boolean recognizer* for Q with output grammars Φ_0 and Φ_1.

IV. Synthesis of Boolean Recognizer Grammars

For a given boolean function Q and output grammars Φ_0 and Φ_1 we supply the construction of a boolean recognizer grammar Φ_Q and demonstrate it with a simple example.

A. Construction of a Boolean Recognize Grammar

Let Q be a boolean function and Φ_0 and Φ_1 be output grammars designed to operate on the initial graph $G_0 = (V, \emptyset, x \mapsto a)$. First define four sub-grammars

$$\begin{aligned}
\Phi_0' &= \{\ r[f] \ \mid\ r \in \Phi_0,\ Q(\text{range}(f)) = 0\ \}, \\
\Phi_1' &= \{\ r[f] \ \mid\ r \in \Phi_1,\ Q(\text{range}(f)) = 1\ \}, \\
\Psi &= \{\ x_n\ e\ y_m \ \rightharpoonup\ X \ \mid\ n \neq m\ \}, \\
\text{and} & \\
\Omega &= \left\{ \begin{array}{llll} x_n & -\omega_n & \rightharpoonup\ \omega_n & \omega_n \\ \omega_n & \stackrel{\circ}{\rightharpoonup} & a_n & \end{array} \right\}
\end{aligned}$$

where

$$X = \begin{cases} x_{n|m}\ e\ y_{n|m} & \text{if } Q(n) = Q(m) = Q(n|m), \\ \omega_{n|m} \quad \omega_{n|m} & \text{otherwise.} \end{cases}$$

Note that the symbol e is a wildcard standing for either an edge or the absence of an edge. It is intended to remain consistent within a rule definition. Finally, the notation $n|m$ represents the *bitwise or* operation.

The roles of the four sub-grammars defined above are as follows. Grammars Φ_i' contain copies of the output grammars for each subscript n such that $Q(n) = i$. The grammar Ψ updates subscripts between any two (interacting) vertices to reflect the union of the information gathered by each particle. It also introduces the symbol ω to indicate that an inconsistency in information was encountered, implying the need to change from one output grammar to the other. We assume that ω does not occur in any of the rules in either of the output grammars. Finally, the grammar Ω removed edges between vertices labeled by ω_n until they all have degree 0. These vertices are then relabeled a_n, allowing them to be assembled into new components by the correct output grammar, the one corresponding to the subscript $Q(n)$.

The boolean recognizer grammar is the union

$$\Phi_Q = \Phi_0' \cup \Phi_1' \cup \Psi \cup \Omega. \tag{1}$$

B. An Example

Consider the boolean function Q_1 defined by the truth table

n	$Q_1(n)$
0	1
1	0

and output grammars

$$\Phi_0 = \{\ a\ \rightharpoonup\ f\ \} \text{ and } \Phi_1 = \{\ a\ \rightharpoonup\ t\ \}.$$

The corresponding boolean recognizer grammar is

$$\Phi_{Q_1} = \left\{ \begin{array}{lllll} a_1 & \rightharpoonup & f_1 & & (\Phi_0') \\ a_0 & \rightharpoonup & t_0 & & (\Phi_1') \\ a_0 & a_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ a_0 & f_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ a_0 & w_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ t_0 & a_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ t_0 & f_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ t_0 & w_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ w_0 & a_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ w_0 & f_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ w_0 & w_1 & \rightharpoonup & w_1 \quad w_1 & (\Psi) \\ w_0 & \stackrel{\circ}{\rightharpoonup} & a_0 & & (\Omega) \\ w_1 & \stackrel{\circ}{\rightharpoonup} & a_1. & & (\Omega) \end{array} \right.$$

Neither of the control grammars in this example contain rules that create edges. Therefore, rules in the construction Φ_{Q_1} that contain edges have been left out.

This boolean recognizer grammar results in all vertices labeled t_0, if all vertices in the initial graph are labeled a_0, or all vertices labeled f_1, if one or more vertices in the initial graph are labeled a_1. Figure 1 shows a sample trajectory of the system (G_0, Φ_{Q_1}) when G_0 contains a single part labeled a_1 and a number of parts labeled a_0.

It is possible to construct a grammar that describes the same behavior as Φ_Q using fewer rules. While we have no automatic method to guarantee a minimum-sized Φ_Q, it is certainly possible to use the construction here as a starting point and then modify the resulting boolean recognizer grammar by hand, especially for simple output grammars.

For example, consider the boolean recognizer grammar Φ_{Q_1} constructed above. We can use the facts that no edges are defined in any of the rules in Φ_{Q_1}, that the symbols w_0 never appears on the right-hand side of any rule, and that the symbol w_1 can be replaced by f_1 on the right-hand side of rules. Removing rules that contain w_0 and w_1 on their left-hand side and then substituting f_1 for w_1 where it still appears yields the grammar

$$\Phi_{Q_1}' = \left\{ \begin{array}{lllll} a_1 & \rightharpoonup & f_1 & & \\ a_0 & \rightharpoonup & t_0 & & \\ a_0 & a_1 & \rightharpoonup & f_1 & f_1 \\ a_0 & f_1 & \rightharpoonup & f_1 & f_1 \\ t_0 & a_1 & \rightharpoonup & f_1 & f_1 \\ t_0 & f_1 & \rightharpoonup & f_1 & f_1, \end{array} \right.$$

which has half as many rules as Φ_{Q_1}. Inspection reveals that it does indeed describe the same behavior as Φ_{Q_1}, modulo w_1 turning into f_1 indirectly.

A slight variation on the boolean recognizer grammar can be used if not all the vertices in the initial graph represent programmable elements. For example, one of the vertices might represent a stationary target or a static environmental feature [19], [11]. In this case it is possible to modify the

$$\begin{matrix} a_0 & a_0 \\ a_0 & a_1 \ a_0 \end{matrix} \Rightarrow \begin{matrix} t_0 & t_0 \\ t_0 & w_1 \ w_1 \end{matrix} \Rightarrow \begin{matrix} t_0 & w_1 \\ t_0 & w_1 \ f_1 \end{matrix} \Rightarrow \begin{matrix} w_1 & w_1 \\ w_1 & w_1 \ f_1 \end{matrix} \Rightarrow \begin{matrix} f_1 & f_1 \\ f_1 & f_1 \ f_1 \end{matrix}$$

Fig. 1. A sample trajectory of the system (G_0, Φ_{Q_1}). The symbol $\Rightarrow$ indicates that more than one rule was applied between each graph.

boolean recognizer grammar as in

$$\Phi''_{Q_1} = \left\{ \begin{array}{lllll} a_0 & \rightharpoonup & t_0 & & \\ a_0 & a_1 & \rightharpoonup & f_1 & a_1 \\ a_0 & f_1 & \rightharpoonup & f_1 & f_1 \\ t_0 & a_1 & \rightharpoonup & f_1 & a_1 \\ t_0 & f_1 & \rightharpoonup & f_1 & f_1 \end{array} \right.$$

where the initial label a_1 represents some vertex or vertices that cannot be programmed.

C. Correctness of the Construction

Theorem 4.1: Each component C of $\Phi_Q(\Phi_0, \Phi_1)$ satisfies exactly one of the following conditions:

i. $\pi_1(C) \in \mathcal{C}(\pi_1(G_0), \Phi_0)$ and $\forall v \in V_C$, $Q(l_2(v)) = 0$,
ii. $\pi_1(C) \in \mathcal{C}(\pi_1(G_0), \Phi_1)$ and $\forall v \in V_C$, $Q(l_2(v)) = 1$, or
iii. $\exists v \in V_C$ such that $(l_C)_1(v) = \omega$.

Proof: The proof is by induction on the number of steps taken in a trajectory $\{G_i\}_{i\in\mathbb{N}}$. The condition is certainly true of the initial graph: Each component consists of a discrete vertex labeled a_i. Furthermore, a is the only component of the initial graphs for the output grammars (so is reachable) and $Q(i)$ is either 0 or 1 so that exactly one of conditions (i) or (ii) apply.

Now, suppose the condition is true of the graph G_i in the trajectory and suppose that $G_i \xrightarrow{r,h} G_{i+1}$ where $r = L \rightharpoonup R$. There are four cases.

Case $r \in \Phi'_0$: Suppose $C_1, ..., C_k$ are the components of $h(V_L)$ and assume that none of these components contain a vertex labeled ω. Then $\pi_1(C_j) \in \mathcal{C}(\pi_1(G_0), \Phi_0)$ for each j and $Q(l_{G_i,2}(v)) = 0$ for all $v \in h(V_L)$, by the inductive hypothesis. Suppose r is derived from the rule $\tilde{r} \in \Phi_0$. Then

$$\pi_1 \circ h(V_L) \xrightarrow{\tilde{r},h} H$$

and the components of H are all contained in $\mathcal{C}(\pi_1(G_0), \Phi_0)$ by definition of the reachable component set. Also, since r does not change the subscripts of any labels, it follows that $Q((l_{G_{i+1}})_2(v)) = 0$ for all $v \in h(V_L)$. Thus the components of G_{i+1} meet condition (i) above. Assume now that at least one of the components $C_1, ..., C_k$ do contain a vertex or vertices labeled ω. By definition, any $r \in \Phi'_0$ cannot rewrite the symbol ω. Therefore any component that contains an ω after the application of r meets condition (iii) and any component that does not meets condition (i) by the previous argument.

Case $r \in \Phi'_1$: This case is similar to case 1.

Case $r \in \Psi$: Suppose that r operates on vertices u and v which are labeled x_m and y_n respectively. If $Q(n|m) = Q(n) = Q(m)$, then, after the rule application, the labels become $x_{n|m}$ and $y_{n|m}$ and the component(s) containing u and v do not change (since no edges are added or deleted by r). Thus, if the component(s) of G_i containing u and v satisfy (i) (resp. (ii)) above, then so do the component(s) of G_{i+1} containing u and v. On the other hand, if $Q(n|m) \neq Q(n)$ or $Q(n|m) \neq Q(m)$, then the labels of u and v are re-labeled by $\omega_{n|m}$ and condition (iii) applies, above.

Case $r \in \Omega$: In this case, r either operates on a component of size greater than 1, in which case it splits it into two components having vertices labeled ω (condition (iii)), or r operates on a discrete vertex and re-labels it a_n, which satisfies either condition (i) or (ii) depending on $Q(n)$. □

Theorem 4.2: In any trajectory of (G_0, Φ_Q), the subscripts eventually stop changing and arrive at the value $Q_{max} = \bigvee_{v\in V_{G_0}} l_2(v)$.

Proof: Only the rules in Ψ change the subscripts. They do so by computing, in a pairwise fashion, the bitwise or of all of the subscripts in the system, resulting in each subscript being equal to Q_{max}. □

Theorem 4.3: in any trajectory of (G_0, Φ_Q) it is eventually the case that labels of the form ω_n stop appearing.

Proof: Once the subscripts all become Q_{max} there may be labels of the form $\omega_{Q_{max}}$. Via the rules in Ω, these eventually become completely disconnected and then relabeled $a_{Q_{max}}$. □

The above arguments together imply that the system defined by Φ_Q eventually behaves like either Φ_1 or Φ_2, depending on whether $Q(Q_{max})$ is 0 or 1 – meaning that components that do not correspond to the correct output grammar are not stable. We thus have the main result.

Theorem 4.4: The construction given for $\Phi_Q(\Phi_0, \Phi_1)$ results in a boolean recognizer grammar.

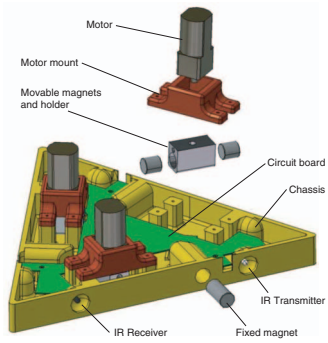
V. Robotic Testbed

A. Hardware

The methods in this paper were designed to be implemented on a set of robots called programmable parts [5]. A *programmable part* (Figure 2(a)) consists of an equilateral triangular chassis that supports three controllable latching mechanisms, three IR transceivers, and control circuitry.

The parts float randomly on a custom-made air table (Figure 2(b)). To maximize useful collisions, we use various methods for stirring the parts on the air table.

Each latch consists of two permanent magnets: one fixed and the other mounted on the end of a small geared DC motor. The default position of the magnets is such that the north pole of the fixed magnet and the south pole of the movable magnet are pointing out. When two latches from different parts come into contact, they temporarily bind – the fixed magnet of one attaching to the movable magnet of the other. At that point, a contact switch on each part is pressed



(a) A programmable part

(b) The air-table environment

Fig. 2. (a) The components of the programmable part include low power magnetic latches, infrared communications, and an on-board microcontroller. (b) The components of the testbed environment include a custom-made air table and fans for mixing.

and the parts may communicate. If at any point the parts mutually decide to detach from each other, each temporarily rotates its movable magnet 180°, forcing the parts apart. The movable magnets then return to their default positions.

This mechanism allows us to *program* the outcomes of each interaction and we use the graph grammar formalism to write down these programs. This in turn allows us to direct the programmable parts to self-assemble into arbitrary assemblies [8] (e.g hexagons [5]) or perform tasks like self-replication [10], depending on the graph grammar we use.

B. Simulation

We also explore the programmable parts system in a simulation environment that allows us to "experiment" with a greater number of parts [5]. The simulation uses the *Open Dynamics Engine* [20] library, which can routinely compute trajectories of hundreds of parts and determine the result of collisions and contact situations quickly. In the simulation environment the parts are stirred by a grid of downward-facing fans. The main simplifications in the simulation are that the magnets are modeled as pairs of point attractors to avoid explicitly modeling their magnetic fields; the effect of the fans is modeled as a time varying force field acting on only the centers of mass of the parts; and communication is not explicitly modeled.

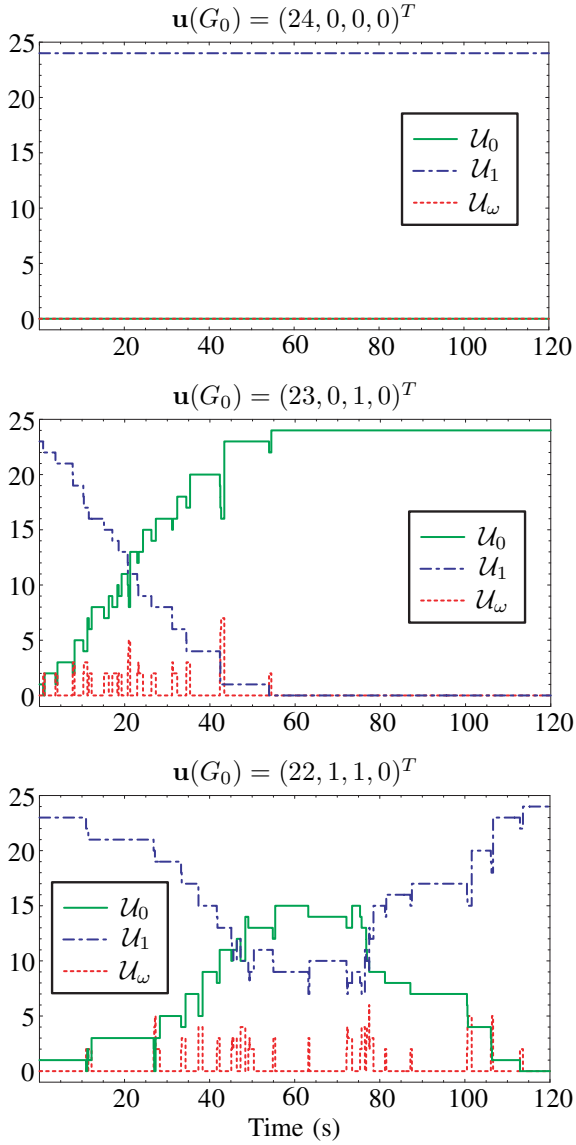
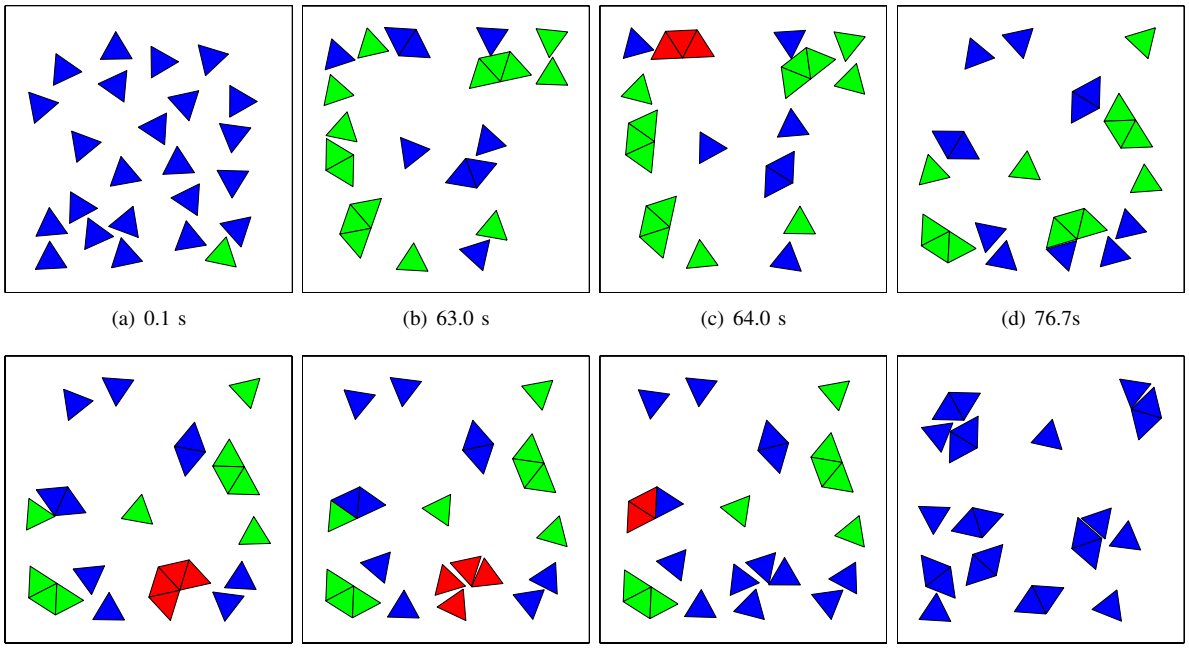


Fig. 3. Results of simulating the boolean recognizer grammar Φ_{Q_2} on 24 programmable parts for three different initial conditions. The lines in each plot represent the number of parts executing Φ_0, Φ_1, or are labeled with an ω during assembly, respectively.

VI. Simulation Analysis

We use the simulation engine here to demonstrate the behavior of an example system. We do so by showing trajectories generated by an example boolean recognizer grammar, starting from a representative set of initial conditions. For this example, consider the boolean function Q_2 defined by the truth table

n	$Q_2(n)$
$(00)_2$	1
$(01)_2$	1
$(10)_2$	0
$(11)_2$	1



(a) 0.1 s (b) 63.0 s (c) 64.0 s (d) 76.7s

(e) 76.9 s (f) 77.6 s (g) 77.7 s (h) 120.0 s

Fig. 4. Frames from simulation shown in Figure 3(bottom), labeled by total time elapsed. The green and blue parts are using the output grammars Φ_0 and Φ_1, respectively. The red parts have the label ω.

and output grammars

$$\Phi_0 = \left\{ \begin{array}{cccc} a & a & \rightharpoonup & b-b \\ a & b & \rightharpoonup & c-d \\ b-c & & \rightharpoonup & b-b \\ b-d & & \rightharpoonup & c-d \\ d-d & & \rightharpoonup & b \quad b, \end{array} \right.$$

and

$$\Phi_1 = \{ \; a \quad a \quad \rightharpoonup \quad b-b \; \}.$$

The output grammar Φ_0 results in stable components that are connected triples, or *trimers*, while the output grammar Φ_1 results in stable connected pairs, or *dimers*. The boolean function $Q_2(n)$ can be thought of as the proposition

$$b(n,1) \wedge \neg b(n,2)$$

where $b(n,i)$ denotes the i^{th} bit of n. Therefore, the corresponding boolean recognizer grammar Φ_{Q_2} creates either stable trimers if G_0 contains any parts labeled a_{10} or stable dimers otherwise.

Note that while the construction of Φ_{Q_2} relies upon output grammars that model the programmable parts simply as single vertices, it is also possible to use grammars that account for the geometry of the robots more explicitly (e.g. see [5]). Without any loss of generality, we omit a detailed discussion of this for the sake of simplicity.

The boolean function Q_2 tells us the default system behavior is that of the output grammar Φ_1. If the initial condition consists only of parts labeled a_{00} or a_{01}, only Φ_1 is ever executed. However, if the initial condition contains any parts labeled a_{01}, the output grammar will switch between Φ_1 and Φ_0. In this case, the final result depends on the presence of parts labeled a_{10} or a_{11}. Only three types of transient behaviors are therefore expected in this example, and we simulate three representative initial conditions to examine them.

We represent initial conditions with a vector $\mathbf{u} : \mathcal{G} \rightarrow \mathcal{R}^n$ (where $n = |\mathrm{dom}(Q)|$), defined as

$$\mathbf{u}(G) = (u_0(G), ..., u_M(G))^T \text{ with } u_i(G) = \left| l_{G,2}^{-1}(i) \right|.$$

For a given graph this vector contains the number of vertices labeled with each type of subscript. For example, the initial condition $\mathbf{u}(G_0) = (22,1,1,0)^T$ indicates that the initial graph G_0 contains 22 parts labeled a_{00}, one part labeled a_{01}, one part labeled a_{10}, and no parts labeled a_{11}.

In order to track the evolution of labels in a trajectory, we also define

$$\begin{aligned} \mathcal{U}_i(G) &= \left| \{ \, v \in V_G \mid Q(l_{G,2}(v)) = i \wedge l_{G,1}(v) \neq \omega \, \} \right|, \\ \text{and} \\ \mathcal{U}_\omega(G) &= \left| \{ \, v \in V_G \mid l_{G,1}(v) \neq \omega \, \} \right|. \end{aligned}$$

These equations measure the number of parts that are using the output grammar Φ_0 or Φ_1, or are labeled with an ω, respectively. In Figure 3 we plot these quantities over time in three simulations, which demonstrate the three different transient behaviors expected for this example.

Several details of the boolean recognition process are illuminated by Figure 4, which shows snapshots in time of the simulation whose data we plot in Figure 3(bottom). The first snapshot (Fig. 4(a)) shows the initial condition of the simulation. Figures 4(b), 4(c) show the creation of a trimer under the output grammar Φ_0 (middle left). Figures 4(d)-4(g) illustrate the approach towards resolving inconsistent

information. Here, a trimer – with parts using Φ_0 – interacts with a single part using Φ_1. First, all four take the label ω. They then break apart and take new labels, using the output grammar consistent with their updated subscripts – in this case Φ_1. Fig. 4(h) shows that all 24 parts are eventually using Φ_1.

VII. Discussion

We have introduced a way to compose two local rule sets to program a self-organizing system. The collective recognition of a condition on the initial state is performed locally, resulting in one of two distinct global outcomes.

We are presently exploring several questions that address the fundamental questions of robotic self-assembly. For example, in a stochastic setting (such as with the programmable parts) what is a tractable model of the dynamics and can we argue about, e.g., the *expected behavior* of the system or the *rate* at which recognition occurs? The rate at which components are formed by the programmable parts is generally known [6] and so determining the rate at which recognition occurs is a justifiable next step. Also, can the output of the boolean recognizer be used as the input to another to produce arbitrary, and even feedback, interconnections? How else can grammars be composed? Can we map the construction in this paper onto a novel physical system beyond the robotic testbed, such as bacterial quorum sensing [21] or DNA self-assembly?

References

[1] 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.

[2] R. C. Mucic, J. J. Storhoff, C. A. Mirkin, and R. L. Letsinger. DNA-directed synthesis of binary nanoparticle network materials. *Journal of the American Chemical Society*, 120:12674–12675, 1998.

[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.

[4] A. Greiner, J. Lienemann, J. G. Korvink, X. Xiong, Y. Hanein, and K. F. Böhringer. Capillary forces in micro-fluidic self-assembly. In *Fifth International Conference on Modeling and Simulation of Microsystems (MSM'02)*, pages 22–25, April 2002.

[5] 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.

[6] N. Napp, S. Burden, and E. Klavins. The statistical dynamics of programmed robotic self-assembly. In *International Conference on Robotics and Automation*, 2006. Submitted.

[7] P. White, V. Zykov, J. Bongard, and H. Lipson. Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of Robotics Science and Systems*, Boston, MA, June 2005.

[8] E. Klavins, R. Ghrist, and D. Lipsky. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 51(6):949–962, June 2006.

[9] Eric Klavins, Samuel Burden, and Nils Napp. Optimal rules for programmed stochastic self-assembly. In *Robotics: Science and Systems*, Philadelphia, PA, 2006.

[10] Eric Klavins. Universal self-replication using graph grammars. In *The 2004 International Conference on MEMs, NANO and Smart Systems*, Banff, Canada, 2004.

[11] J-M. McNew and E. Klavins. Locally interacting hybrid systems with embedded graph grammars. In *Conference on Decision and Control*, 2006. To Appear.

[12] P. Corke, R. Peterson, and D. Rus. Localization and navigation assisted by cooperating networked sensors and robots. *Int. J. Robotics Research*, 24(9):771, 2005.

[13] M. Batalin and G. S. Sukhatme. The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Transactions on Robotics*, 2006.

[14] R. A. Russell, D. Thiel, R. Deveza, and A. Mackay-Sim. A robotic system to locate hazardous chemical leaks. *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, 1, 1995.

[15] J. A. Fax and R. M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transaction on Automatic Control*, 49(9):1465–1476, 2004.

[16] A. Jadbabaie, Jie Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transaction on Automatic Control*, 48(6):988–1001, 2003.

[17] N. A. Lynch, M. Fischer, and R. J. Fowler. A simple and efficient byzantine generals algorithm. In *IEEE Symp. on Reliability in Distributed Software and Database Systems, IEEE CS 2, Wiederhold(ed), Pittsburgh PA*, 1982.

[18] M. Abadi and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993.

[19] J-M. McNew and Eric Klavins. A grammatical approach to cooperative control. In O. Prokopyev, R. Murphey, and P. Pardalos, editors, *Cooperative Control and Optimization*. Kluwer Academic Publishers, 2005.

[20] ODE: The open dynamics engine. http://www.ode.org/.

[21] D. Karig and R. Weiss. Signal-amplifying genetic circuit enables in vivo observation of weak promoter activation in the rhl quorum sensing system. *Biotechnology and Bioengineering*, 89(6):709–718, 2005.