The Statistical Dynamics of Programmed Self-Assembly

Nils Napp Samuel Burden Eric Klavins*

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

***Abstract*— We describe how a graph grammar program for robotic self-assembly, together with measurements of kinetic rate data yield a Markov Process model of the dynamics of programmed self-assembly. We demonstrate and evalidate the method by applying it to a physical testbed consisting of a number of "programmable parts", which are able to control their local interactions according to their on-board programs. We describe a technique for obtaining kinetic rate constants from simulation and a comparison of the behavior predicted by the Markov model with the behavior predicted by a low-level simulation of the system.**

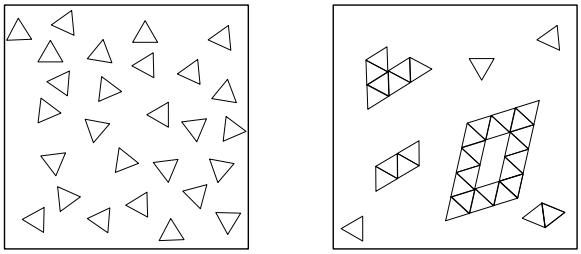
I. Introduction

This paper is about modeling a class of robotic systems that can be *programmed* to self-assemble into specific structures. We believe this work to be in the same vein as the initial work in modeling, for example, traditional robot manipulators using classical mechanics. Although the language of classical mechanics was well established at that point, adapting it to the application of feedback-controlled manipulators in a natural way required substantial effort. The analogous theory for programmed self-assembly is statistical mechanics, or more specifically, statistical dynamics. Although the theory of programmable self-assembly is not by any means unexplored (e.g. [5] is also in this vein), much work needs to be done before self-assembling systems can be reliably engineered.

The class of systems we consider consist of a number of robotic parts that are *randomly stirred* in some container. Upon chance collisions, the parts may bind, communicate and update their internal states. At any time they may detach from each other. The engineering goal is to program the particles so that a desired structure emerges in a predictable and reliable fashion (see Figure 1). We believe that this will lead to a variety of applications from macro-scale multi-robot coordination to novel manufacturing techniques of micro-scale devices.

We consider both an abstract theoretical model of this process based on graph grammars and Markov Processes, and a physical testbed that has been constructed to validate the theory [2].

*Corresponding author. This work is partially supported by NSF Grant #0347955: *CAREER: Programmed Robotic Self-Assembly.*



(a) (b)

Fig. 1. (a) An initial configuration of programmable robotic parts (See Figure 4). The parts are stirred randomly so as to produce collisions. (b) An assembly parts after the assembly process has completed. A complete final assembly and several partial assemblies are shown.

Our main contribution is a method by which to define a discrete state continuous time Markov Process model that can be directly obtained from (1) a description of the program used by the parts and (2) values of the *kinetic rate constants* defining the rates at which sub-assemblies form and decay. Such a model is more useful than a low-level simulation (based on rigid body dynamics, friction, collisions, etc.) of the system for a number of reasons. First, statistical sampling of a fast implementation of Gillespie's method [6] can be used to quickly evaluate the performance of a given program, whereas a low-level simulation might require considerable computation time. Second, such a model can be used in analysis of the system dynamics using, for example, statistical *model checking* [14], [9].

In this paper, we first define a model of self-assembly based on graph grammars [8] and statistical mechanics [4]. In particular, we describe how a graph grammar induces a Markov Process model of the system. We then show how to use the model to describe the dynamics of a particular robotic self-assembly system [2]. We describe how to obtain *kinetic rate constants* from a low-level simulation of the system and discuss for what values of the parameters (average kinetic energy, density) the assumptions of our model hold. Finally, we compare the two modeling methods to show that they predict similar results.

II. Related Work

The present paper owes much to the work of Hosokawa, Shimoyama and Miura who described a self-assembling system using chemical kinetics [7]. Their system consisted of passive triangular parts in a vertical shaker. Using an approximation for the kinetic rate constant based on (an oversimplification of) the geometry of sub-assemblies, they obtained mass-action-kinetics equations for their system. Furthermore, they showed that the kinetics model roughly matched experiment. The approach in the present paper is similar, with the following differences: (1) The parts we consider are programmable, meaning that the system dynamics are programmable as well. This leads to questions of design and performance; (2) We measure the kinetic rate constants from experiments and use a Markov-Process model instead of a continuum model; (3) We explore the parameter space of our system to identify where it is appropriate.

The modeling techniques in the present paper are applicable to robots, built by several groups, that either float on an air table [12], [2] or even that float in oil [11]. We also believe that these ideas will be applicable to micro- and nano-scale self-assembly problems [3], [13]. No comprehensive dynamical systems model or programming discipline has yet been developed for these systems, although the graph grammar approach built on here and introduced elsewhere [8] is an important start. Besides the grammatical approach, the most relevant work toward understanding self-assembly "programs" in a stochastic setting is in nucleic acid design [5], where the free-energy landscape associated with DNA hybridization reactions can be engineered. The present work differs in that we suppose that local interactions are controllable, whereas in the DNA work the oligo sequences are the controllable elements.

III. The Dynamics of Programmed Self-Assembly

We describe a general mathematical model of programmed self-assembly. The model is phrased in the language of graphs, rewrite systems and Markov processes.

The state of a programmable self-assembling system is described by a *labeled graph* over a fixed set of vertices and having two edge types:

$$G = (V, E_{bond}, E_{temp}, l)$$

where

1) $V = \{1, 2, 3, ..., n\}$ is a finite set of particles;
2) E_{bond} is a set of "bonded" undirected edges of the form $\{u, v\}$ with $u, v \in V$.
3) E_{temp} is a set of "temporary" undirected edges of the form $\{u, v\}$ with $u, v \in V$.
4) $l : V \to \Sigma$ is a labeling function that associates to each particle u a label $l(u)$ in the alphabet Σ.

The absence of an edge between two parts indicates that they are not physically connected. A "temporary" edge indicates that the parts have attached to each other, but have not yet communicated their states. A "bonded" edge indicates that the parts have decided to remain attached.

In this paper, we simply use the term *graph* for graphs of this type. We use the notation uv for the set $\{u, v\}$ representing an edge. We draw graphs with elements of Σ representing labeled vertices, solid lines representing bonded edges and dashed lines representing temporary edges. We generally use the alphabet $\Sigma = \{a, a', a'', ..., b, b', b'', ...\}$ and assume that it is *closed under priming* (i.e. $x \in \Sigma \Rightarrow x' \in \Sigma$).

We assume the following concepts as standard. Graph *isomorphism* is written $G \simeq H$. Isomorphism implies the existence of a *witness* h that maps the vertices of G to the vertices of H and preserves structure and labels. If $U \subseteq V$, then $G|_U$ denotes the graph induced by U. The degree of a vertex is $d(v) = d_{bond}(v) + d_{temp}(v)$, the number of edges involving u. If $\{G_i\}_{i\in\mathbb{N}}$ is a sequence of graphs, then we may refer to the sequence of labeling functions by $\{l_k\}_{k\in\mathbb{N}}$ or the sequence of degrees of a vertex v by $\{d_k(v)\}_{k\in\mathbb{N}}$, and so on.

A. Graphs and Rewrite Rules

We consider two types of rewrite rules that model local interactions among particles. The first type is the *binary rule*, of the form $L \rightharpoonup R$ where L and R are graphs over the vertex set $\{1, 2\}$. For example, the rule

$$a \cdots a \rightharpoonup b - b$$

describes how two parts that are both labeled a and that have initiated communications may form a bond.

The second type of rule is the unary rule, of the form $L \rightharpoonup R$, where L and R are graphs over one vertex, which allows a part to change its own label. For example, the rule

$$b' \rightharpoonup a \tag{1}$$

allows a part that has no neighbors to change its label from b' to a.

The following definition is similar to that in [8], except adapted for graphs with two edge types and restricted to rules of the form described above. It states formally how trajectories are obtained by rewriting a graph locally according to a binary or unary rule.

Definition 3.1: Given an initial graph G_0 over vertex set $V_0 = V$ and a set Φ of binary and/or unary rules, a sequence

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

is a *trajectory* of (G_0, Φ) if for all i, there exists a rule $L \rightharpoonup R \in \Phi$ such that:

i. If $L \rightharpoonup R$ is binary, there exist distinct vertices $u, v \in V$ such that
   a. $G_i|_{\{u,v\}} \simeq L$ and $G_{i+1}|_{\{u,v\}} \simeq R$ via the same witness h.
   b. $G_i|_{V-\{u,v\}} = G_{i+1}|_{V-\{u,v\}}$.

ii. If $L \rightharpoonup R$ is unary, there exists a vertex $u \in V$ such that
   a. $G_i|_u \simeq L$ and $G_{i+1}|_u \simeq R$
   b. $G_i|_{V-u} = G_{i+1}|_{V-u}$.

If $G = G_i$ for some trajectory of (G_0, Φ), then G is *reachable*. The set of all reachable graphs is denoted $\mathcal{R}(G_0, \Phi)$.

A set of rules and an initial graph determine the set of assemblies, the components of reachable graphs, that can be formed.

Definition 3.2: If C is a component of G for some $G \in \mathcal{R}(G_0, \Phi)$ and C has no temporary edges, then C is called a *reachable component*. The set of all such components (up to isomorphism) is denoted $\mathcal{C}(G_0, \Phi)$.

The goal of the self assembly problem is to define Φ so that $\mathcal{C}(G_0, \Phi)$ contains desirable assemblies.

B. The Grammar Induced by a Program

The program stored on the parts induces a dynamical system when they are allowed to interact. By a *program* Φ_{prog}, we mean a set of unary or binary rules, where each binary rule has a connected left hand side. Thus, the binary rules in a program may change a temporary edge to a bond, change the labels of bound parts, or disassociate a bond (e.g. see the example in Section III-F).

Given Φ_{prog}, we define three other sets of rules. For convenience, we denote by $LHS(\Phi)$ the set

$$LHS(\Phi) = \{L \mid L \rightharpoonup R \in \Phi\}.$$

The induced rule sets are as follows.

1) *Formation rules* are applied by the environment when parts collide and temporarily attach:
$$\Upsilon_{form} = \{x \quad y \rightharpoonup x \cdots y \mid x, y \in \Sigma\}.$$
2) *No-op* rules apply when no rule in Φ applies to a newly attached pair:
$$\Phi_{noop} = \{x \cdots y \rightharpoonup x \quad y \mid x \cdots y \notin LHS(\Phi)\}.$$
3) *Break rules* are applied by the environment when bonds are broken due to energetic collisions:
$$\Upsilon_{break} = \{x - y \rightharpoonup x' \quad y' \mid x, y \in \Sigma\}.$$
The labels on the parts are "primed" by this rule to reflect the fact that the parts are programmed to "notice" when their local connectivity has changed unexpectedly.

The grammar induced by a given program Φ_{prog} is thus

$$\Phi_{system} = \Phi_{prog} \cup \Upsilon_{form} \cup \Phi_{noop} \cup \Upsilon_{break}. \qquad (2)$$

C. Natural Components

Given a system of particles that can attach to form assemblies, we wish to know what types of graphs can emerge, regardless of their programming. Thus, consider the system induced by the grammar

$$\Phi_{nat} = \{ \ a \cdots a \rightharpoonup a - a \ \}$$

except with

$$\Upsilon_{break} = \{a - a \rightharpoonup a \quad a\}.$$

In this system, everything sticks to everything else. The set $\mathcal{C}_{nat} = \mathcal{C}(G_0, \Phi_{nat})$ of all components of this system are called the *natural component types*. If $C' \in \mathcal{C}(G_0, \Phi)$ is a component of some other system Φ and C' is isomorphic to some component $C \in \mathcal{C}_{nat}$, not including labels, we call C the *type* of C'. For example,

$$a - b \cdots c \text{ has type } a - a \cdots a.$$

We denote the type of a component C by $T(C)$.

D. Reaction Types and Kinetic Rate Constants

The grammatical formulation of self-assembly describes only what behaviors are *possible*, but not how *probable* they are or how long they take. However, under the assumptions that (1) assemblies diffuse through their environment and (2) the system is "well mixed", we may use the *reaction-diffusion* model from chemical kinetics. In this case, a grammar together with a set of *kinetic rate constants* leads to a continuous time, discrete space *Markov Process*. In Section IV-D we explore the appropriateness of these assumptions for the robot testbed in [2].

There are two types of rate constants. First are the rates at which temporary edges form and bonds break. These rates are fundamental physical constants associated with the geometry of the interacting assemblies and the "temperature" and density of the system. The other type is the communications rate k_{com}. It results from the time it takes for two parts to apply a rule in their program (or a no-op rule). We assume that k_{com} is significantly higher than all other rate constants.

Rates are associated with *actions*. An action consists of a rule and a place in the current graph to apply it. For example, one application of a particular rule may be to attach two single parts together while another may be to attach two complicated sub-assemblies together. To make the distinction, we associate a *reaction type* μ to each action, which has one of the following forms:

$$A \rightharpoonup B \qquad \text{(R1)}$$
$$A + B \rightharpoonup C \qquad \text{(R2)}$$
$$A \rightharpoonup B + C, \qquad \text{(R3)}$$

where $A, B, C \in \mathcal{C}(G_0, \Phi)$.

We assume that it is possible to measure or otherwise obtain a *natural reaction rate* k_μ for each reaction type μ (e.g. see Section IV-C). This is the rate associated with the reaction type μ in Φ_{nat} when A, B and C are replaced by their natural component types $T(A)$, $T(B)$ and $T(C)$ (i.e. they are relabeled with all a's). We assume that these rates are measured at a fixed temperature and density and that subsequent experiments (with programs) occur at the same temperature and density.

For each reaction μ and rule r, we define the *caliber of the reaction via* r, denoted $n(\mu, r)$, to be the number of ways that the reaction μ can occur via the rule r. Also, for each reaction μ, we define the *caliber of the reaction type*, denoted $n(\mu)$, to be the number of ways that the reaction can occur by the addition or deletion of an edge, independent of labels. For example, if r is the rule $a - b \rightharpoonup a' \quad b'$ and μ is the reaction

$$a - b - c \ \rightharpoonup \ a' \ + \ b' - c$$

then $n(\mu, r) = 1$ while $n(\mu) = 2$.

Definition 3.3: Given Φ defined by Equation (2), suppose that G' can be obtained from $G \in \mathcal{R}(G_0, \Phi)$ via the rule $r \in \Phi$ and a reaction of type μ. Then the *rate* at which G transitions to G' is given by

$$k_{\mu,r}(G, G') = k_{com} \tag{3}$$

if $r \in \Phi_{prog} \cup \Phi_{noop}$ and

$$k_{\mu,r}(G, G') = \frac{n(\mu, r)}{n(\mu)} k_\mu \tag{4}$$

otherwise. If G' cannot be obtained from G via any rule, we set $k(G, G') = 0$.

The result is a continuous time, discrete state Markov Process with states $\mathcal{R}(G_0, \Phi)$ and rates $k(G, G')$. If Φ is induced by from Φ_{prog}, then this process is called the *Markov Process induced by Φ_{prog} and the natural rates k_μ.*

E. Macrostates and Markov Processes

It is useful to represent the equivalence class of graphs isomorphic to a given graph by listing the number of each component type present in graphs in the class. To this end, suppose that $\mathcal{C}(G_0, \Phi) = \{C_1, C_2, C_3, ...\}$. Then $\mathbf{v} : \mathcal{C}(G_0, \Phi) \to \mathbb{N}$ represents all graphs $G \in \mathcal{R}(G_0, \Phi)$ with $\mathbf{v}(1)$ components isomorphic to C_1, $\mathbf{v}(2)$ components isomorphic to C_2 and so on. We write these representatives in vector notation as in, for example,

$$\mathbf{v} = (\ 4\ 3\ 0\ 1\ 0 \dots\ )^T$$

which denotes that $\mathbf{v}(1) = 4$, $\mathbf{v}(2) = 3$ and so on. If $\mathbf{v}$ represents the equivalence class $[G]$ and $H \in [G]$, we write $G \models \mathbf{v}$. In statistical mechanics, G is called a *microstate* and $\mathbf{v}$ is called a *macrostate*.

The *multiplicity* of the reaction μ in the macrostate $\mathbf{v}$, denoted $M_\mu(\mathbf{v})$, is the number of ways μ can happen in $\mathbf{v}$. For example, if μ is $C_1 + C_2 \to C_3$, then

$$M_\mu(\mathbf{v}) = \mathbf{v}(1)\mathbf{v}(2).$$

Finally, if $G \models \mathbf{v}$ and $G' \models \mathbf{v}'$, then the rate from $\mathbf{v}$ to $\mathbf{v}'$ is

$$k(\mathbf{v}, \mathbf{v}') = \sum_{\mu,r} M_\mu k_{\mu,r}(G, G').$$

As implied by Definition 3.3, this rate can be determined solely by examining the components represented in the vector $\mathbf{v}$.

We interpret macrostates and the rates between them as describing a continuous time, discrete state Markov Process. More concretely, suppose that macrostates are ordered somehow $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, ..., \mathbf{v}_N$. The system can be summarized by a *rate matrix* $\mathbf{K} \in \mathbb{R}^{N \times N}$ where

1) If $i \neq j$, then $K_{i,j} = k(\mathbf{v}_i, \mathbf{v}_j) \geq 0$
2) $K_{i,i} = -\sum_{j \neq i} K_{i,j}$.

We denote the ratio $-K_{i,j}/K_{i,i}$ by $P_{i,j}$, for $i \neq j$. Setting $P_{i,i} = 0$ results in a new matrix $\mathbf{P}$ that is called the *embedded Markov Chain* associated with the process. The average behavior of the system is then described by

$$\dot{\mathbf{v}} = \mathbf{K}^T \mathbf{v}. \tag{5}$$

Distinct trajectories through the system in this paper are obtained using the method of Gillespie [6]. Given an initial state $\mathbf{v}_{j_0} \in S$ at time $t_0 = 0$, we produce a *random trajectory* $\{(\mathbf{v}_{j_i}, t_i)\}_{i \in \mathbb{N}}$ as follows. At step i

1) Choose the state $\mathbf{v}_{j_{i+1}}$ randomly according to the j_ith row of the embedded Markov Chain $\mathbf{P}$.
2) Choose $\tau > 0$ randomly according to the probability density function $p(\tau) = \lambda \exp(-\lambda\tau)$ where $\lambda = -K_{j_i,j_i}$ and set $t_{i+1} = t_i + \tau$.

Often, distinct trajectories of the system differ considerably from the average behavior. Sampling distinct trajectories can describe how the system behaves at thermodynamic equilibrium, while Equation (5) predicts an essentially static picture. Note that if a sample trajectory $\{G_i\}_{i=1}^n$ is obtained using rates $k_{\mu,r}(G, G')$, it can be "lifted" to a trajectory $\{\mathbf{v}_{j_i}\}_{i=1}^n$ where $\mathbf{v}_{j_i}$ is the macrostate corresponding to the equivalence class $[G_i]$.

Remark: One may also use the basic rates k_μ as described here and measured in Section IV-C to describe a continuous *mass action kinetics* model. However, such models assume a vast number of reactants, which is not appropriate for the present setting. Furthermore, the continuous model essentially predicts the average behavior (5), which does not capture many interesting details that may be important to programming and correctness arguments. Finally, a continuous model must describe the rate of change of the concentration of each component type in $\mathcal{C}_{nat}$. The number of such types in our systems is not usually known *a priori* and is in any case so prohibitively large as to render the approach analytically unwieldy and possibly useless.

F. An Example

Consider a *toy* system with natural rates as follows. If μ is a reaction that deletes an edge, then $k_\mu = k_r > 0$, a constant. If μ is a reaction that joins two components C_1 and C_2, then

$$k_\mu = \frac{k_f}{|C_1| + |C_2| - 1} \tag{6}$$

as long as vertices in the newly created component have degree 2 or less. Otherwise, set $k_\mu = 0$. Here $k_f > 0$ is a constant. The natural components of the system are thus chains and cycles of parts.

Suppose that Φ is induced by the program

$$\Phi_{prog} = \begin{cases} a \cdots a \rightharpoonup b - c \\ b \cdots b \rightharpoonup d - d \end{cases}$$

and that G_0 contains parts all initially labeled by "a". The goal of Φ_{prog} is to build graphs of the form $c - d - d - c$. Some example rates are shown in Table 1. There are many others, easily determined. In the third reaction, the $\frac{1}{4}$ comes from Equation (4) and the $\frac{1}{3}$ comes from Equation (6).

Because of the rules in Υ_{break}, all components generated by Φ eventually break apart, yielding single parts labeled by primed symbols. Thus, it is useful to define a *recovery*

μ	$k_{\mu,r}$
$a + a \rightharpoonup a \cdots a$	k_f
$a \cdots a \rightharpoonup b - c$	k_{com}
$c - b + b - c \rightharpoonup c - b \cdots b - c$	$\frac{1}{4} \cdot \frac{1}{3} k_f$
$c - d - d - c \rightharpoonup c' + d' - d - c$	k_r.

Table 1. Example reaction rates for the program defined in Section III-F.

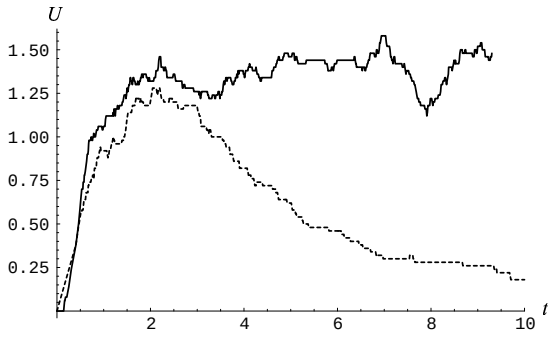


Fig. 3. The number U of final graphs $(c-d-d-c)$ versus time for the systems induced by Φ_{prog} (dashed line) and $\Phi_{prog} \cup \Phi_{rec}$ (solid line) described in Section III-F. Each curve represents the average of 50 simulations of 10 parts for 10 seconds.

program

$$\Phi_{rec} = \begin{cases} c - d' \rightharpoonup c - b & c' \rightharpoonup a \\ d - d' \rightharpoonup b \quad a & d'' \rightharpoonup a \\ b' \rightharpoonup a \end{cases}$$

Figure 2 shows an example initial trajectory of the system induced by $\Phi_{prog} \cup \Phi_{rec}$. Figure 3 shows the behavior of the systems induced by Φ_{prog} and $\Phi_{prog} \cup \Phi_{rec}$ respectively. Each curve represents the average of 50 simulations of 10 parts for 10 seconds. The value plotted is the number U of final graphs $(c-d-d-c)$ as a function of time.

IV. A Robotic Testbed

A. Hardware

The methods in this paper were implemented on a set ten of robots called programmable parts [2]. A *programmable part* (Figure 4) consists of an equilateral triangular chassis that supports three controllable latching mechanisms, three IR transceivers, and control circuitry.

Each latch consists of three permanent magnets: one fixed and the other two 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 and the parts communicate. If at any point they 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.

The parts float on a custom-made air table. To maximize useful collisions, we use various methods for stirring the

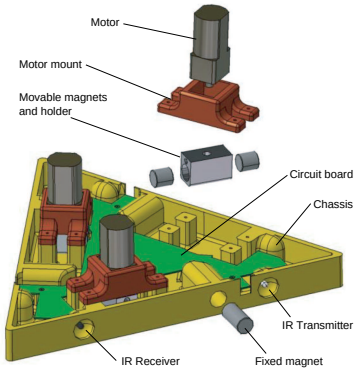


Fig. 4. The components of the programmable part include low power magnetic latches, infrared communications, and an on-board microcontroller.

parts. For the simulations and experiments in this paper, oscillating fans are placed over the table.

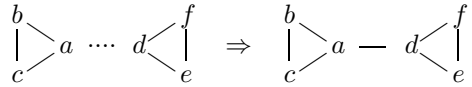
We primarily explore the programmable parts system in a simulation environment that allows us to "experiment" with a greater number of parts [2]. The simulation uses the *Open Dynamics Engine* [1] library, which can compute trajectories of hundreds of parts and determine the result of collisions and contact situations. The 1000's of simulations in this paper were run on an 16 processor *Blade* server and the data was collected and interpreted in MATLAB.

B. Applying the Grammatical Approach

To apply the grammatical approach to the programmables parts, we associate a label in Σ to each latch of each of the N robots. Thus, the vertex set is

$$V = \{(i,j) \mid 1 \le i \le N, 0 \le j \le 2\}.$$

Binary rules have the form



where the *orientation* of the vertices is important. In particular, this rule applies to parts i and k in the graph $G = (V, E_{bond}, E_{temp}, l)$ if for some j_1 and some j_2 it is the case that $\{(i,j_1),(k,j_2)\} \in E_{temp}$,

$$l(i,j_1) = a,\ l(i,j_1+1) = b, l(i,j_1+2) = c$$

and

$$l(k,j_2) = d,\ l(k,j_2+1) = e, l(i,j_2+2) = f$$

where arithmetic on the edge indices is done modulo 3. The update to the labels of the two parts also must preserve orientation. In general, the extension to the definitions in Section III is straightforward – with *orientation-preserving isomorphism* replacing simple *isomorphism*. It is a somewhat surprising fact that this configuration model for the triangular parts along with appropriately defined rates is enough to capture the geometry of the system: Geometrically impossible assemblies simply have zero flux.

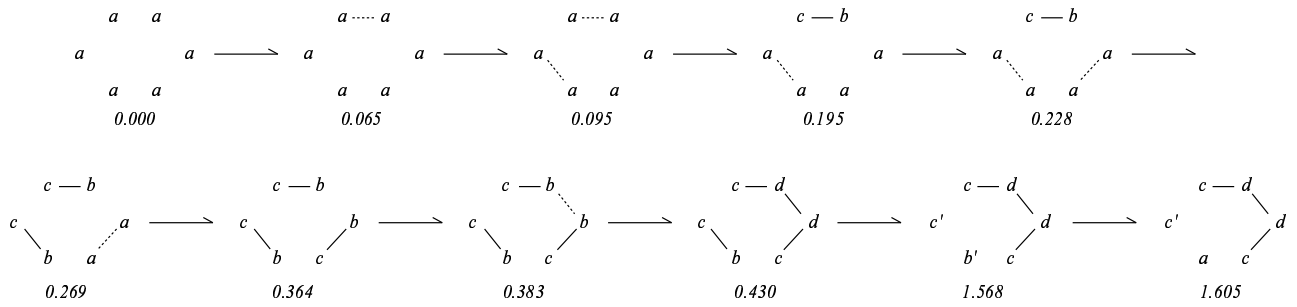


Fig. 2. An initial trajectory of the system induced by $\Phi_{prog} \cup \Phi_{rec}$ along with the time in seconds at which the transitions occured. The trajectory was obtained using Gillespie's method.

C. Obtaining Kinetic Rate Constants From Simulation

To build a Markov Process describing the dynamics of the programmable parts system, we require values for the kinetic rate constants. For this, we use the simulation tool discussed above. We fix the average kinetic energy of $K_{ave} = 5 \times 10^{-4}$, which is the kinetic energy measured from experiments using the actual robots in the laboratory. We also fix a density of $\rho = 5$ parts$/m^2$ and table size of $A = 12m^2$ that (1) approximately match the physical testbed and (2) come from a parameter regime where the reaction-diffusion model is valid (see the next section). This results in simulations with $N = \rho A = 60$ parts.

We describe the procedure used to estimate rates with an example. To determine the rate $k_{1+2\rightharpoonup 3}$ (see Figure 5) for a single part (component type 1) combining with a "dimer" (component type 2) to form a "trimer" (component type 3), we choose a macrostate of the form

$$\mathbf{v}_0 = (N_1\ N_2\ 0\ 0\ ...)^T$$

with N_1 single parts and N_2 dimers. We run n simulations from random initial conditions with the initial velocities chosen from a Gaussian distribution with mean equal to K_{ave}/m, where M is the mass of a part. As soon as a reaction occurs, in this case either $1 + 2 \rightharpoonup 3$ or $2 \rightharpoonup 1 + 1$, we restart the simulation with a new random initial condition. We do not reset the time t. Suppose that the times at which the first reaction occurs are

$$\tau_{1+2\rightharpoonup 3} = (t_1, t_2, ..., t_r).$$

The hypothesis is that the intervals $\Delta t_i = t_{i+1} - t_i$ are distributed according to a Poisson waiting process with mean $\lambda = 1/k_{1+2\rightharpoonup 3}$. We thus arrive at the estimate

$$k_{1+2\rightharpoonup 3} \approx \frac{1}{N_1 N_2}\left(\frac{1}{\langle \Delta t\rangle} \pm \frac{std(\Delta t)}{\sqrt{n}\langle \Delta t\rangle^2}\right)$$

where $\langle \Delta t\rangle$ is the average waiting interval for the reaction and $std(\Delta t)$ is its standard deviation. Said differently, the approximate rate constant is the rate at which the reaction occurs starting at $\mathbf{v}_0$ divided by the multiplicity of the reaction in $\mathbf{v}_0$. From the same set of simulations, we also obtain an approximation of the rate $k_{2\rightharpoonup 1+1}$ from the times $\tau_{2\rightharpoonup 1+1}$. In the next subsection we discuss the parameter regime where we expect that the basic rates are independent of the multiplicity.

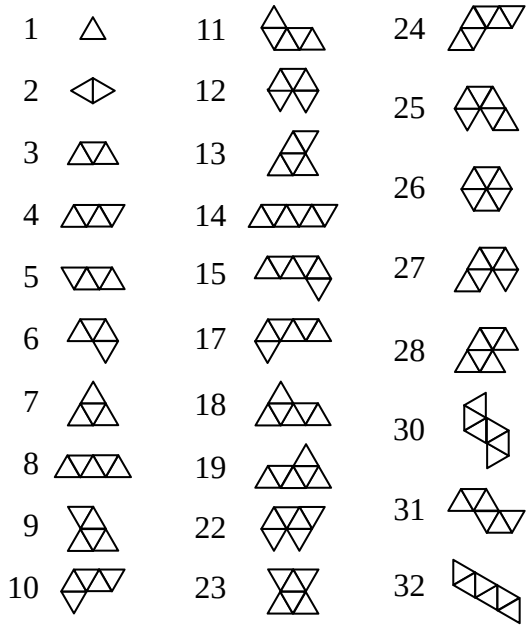


Fig. 5. Index of component types used in Figure 2.

$1+1 \rightharpoonup 2:$	$3.26 \pm .07$	$3+3 \rightharpoonup 17:$	$0.36 \pm .04$
$1+2 \rightharpoonup 3:$	$3.25 \pm .06$	$3+3 \rightharpoonup 18:$	$0.35 \pm .03$
$1+3 \rightharpoonup 4:$	$0.75 \pm .04$	$3+3 \rightharpoonup 19:$	$0.39 \pm .04$
$1+3 \rightharpoonup 5:$	$0.66 \pm .04$	$3+3 \rightharpoonup 22:$	$0.32 \pm .03$
$1+3 \rightharpoonup 6:$	$1.46 \pm .06$	$3+3 \rightharpoonup 23:$	$0.20 \pm .03$
$1+3 \rightharpoonup 7:$	$0.71 \pm .04$	$3+3 \rightharpoonup 24:$	$0.39 \pm .04$
$2+2 \rightharpoonup 4:$	$0.66 \pm .04$	$3+3 \rightharpoonup 25:$	$0.31 \pm .03$
$2+2 \rightharpoonup 5:$	$0.83 \pm .04$	$3+3 \rightharpoonup 26:$	$0.18 \pm .02$
$2+2 \rightharpoonup 6:$	$1.85 \pm .06$	$3+3 \rightharpoonup 27:$	$0.40 \pm .04$
$2+3 \rightharpoonup 8:$	$0.81 \pm .09$	$3+3 \rightharpoonup 28:$	$0.36 \pm .03$
$2+3 \rightharpoonup 9:$	$0.49 \pm .08$	$3+3 \rightharpoonup 30:$	$0.20 \pm .03$
$2+3 \rightharpoonup 10:$	$0.93 \pm .08$	$3+3 \rightharpoonup 31:$	$0.16 \pm .03$
$2+3 \rightharpoonup 11:$	$0.76 \pm .07$	$3+3 \rightharpoonup 32:$	$0.14 \pm .03$
$2+3 \rightharpoonup 12:$	$0.76 \pm .10$	$2 \rightharpoonup 1+1:$	$0.19 \pm .05$
$2+3 \rightharpoonup 13:$	$0.48 \pm .06$	$3 \rightharpoonup 1+2:$	$0.55 \pm .12$
$3+3 \rightharpoonup 14:$	$0.15 \pm .03$	$6 \rightharpoonup 1+3:$	$0.93 \pm .29$
$3+3 \rightharpoonup 15:$	$0.38 \pm .03$	$6 \rightharpoonup 2+2:$	$0.96 \pm .31$

Table 2. Some kinetic rate constants measured for the programmable part system with $N = 60$, $A = 12m^2$, $\rho = 5$parts$/m^2$ and $K_{ave} = 5 \times 10^{-4}$. The initial macrostate for each approximation is chosen so that there are an approximately equal number of the *forward* reactants (see Figure 5 for a listing of the reactant types). Data reflecting reverse rates were combined from those simulations in which reverse reactions occurred. All rates are to be interpreted $\times 10^{-4} s^{-1}$. Not all rates have been measured: Only those rates needed for the grammars explored in this paper are included.

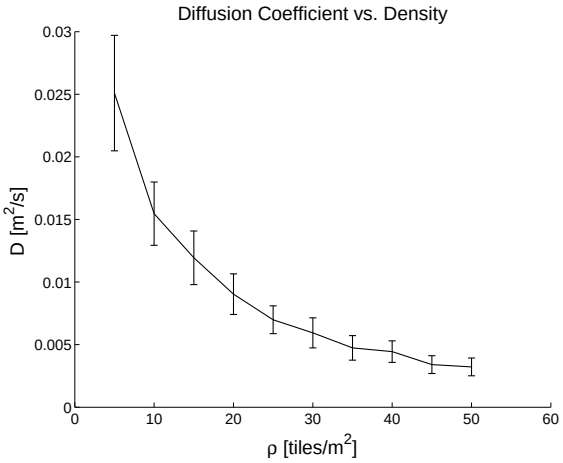


Fig. 6. The diffusion coefficient D versus ρ from simulations of $N = 60$ parts, $K_{ave} = 5 \times 10^{-4}$ and no binding between parts after collisions.

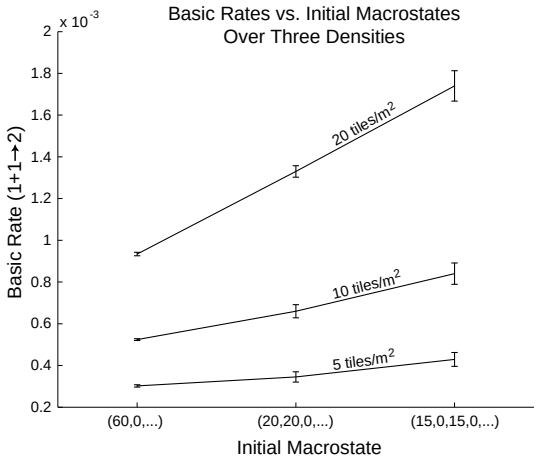


Fig. 7. Kinetic rate constant for the $1 + 1 \rightharpoonup 2$ reaction measured at different densities and from different macrostates. The variation is low at the density $\rho = 5\text{parts}/m^2$ used in the experiments in this paper.

In Table 2 we summarize the basic rates obtained from simulation for a number of the most primitive reactions of the system. These reaction rates are sufficient to test several simple grammars.

D. Where the Model is Valid

The main assumption allowing us to use the reaction-diffusion model is that the system is "well-mixed": The rate at which novel collisions occur is greater than the rate at which reactions occur [6]. In our system, diffusion drives mixing. Thus, a quantifiable notion of "well-mixed" is to require that $D/k_{av} > A$ where D is the *diffusion coefficient*, k_{av} is a typical basic rate and A is the area of the table.

The diffusion coefficient D depends on the density ρ of parts on the table and the average kinetic energy K_{ave}, which we have fixed to 5×10^{-4}. Figure 6 shows D versus ρ measured at a fixed A. According to the figure and taking k_{av} to be approximately 10^{-4}, we conclude that a density of 5 parts/m^2 and an area of $A = 12m^2$ satisfies our well-mixed criterion. The experiments in this paper occur at this density.

To further test that these values capture the regime where the proposed model fits our system, we checked the method for measuring rates in the previous section. Figure 7 shows the basic rate for the reaction $1 + 1 \rightharpoonup 2$ measured from various macrostates. The figure shows that at 5 parts / m^2, the measured rate varies only slightly with the choice of macrostate. At higher densities, however, this is not the case, as the figure also shows.

V. Design and Experiments

The ultimate goal of this effort is to use the model we have described in this paper to build grammars for given tasks. The model allows us to quickly (without a full mechanics-based simulation) evaluate the performance of a grammar with respect to a given task. In fact, we can rigorously use the model for (1) stochastic search for and optimization of rules sets and (2) stochastic model checking of grammars with respect to formal logic specifications. We plan to report on both of these possibilities in future papers.

For now, we consider the task of building a chain of four programmable parts (assembly 6 in Figure 5) in a manner similar to that described in Section III-F. We consider three different grammars for this task:

1) Φ_1 : build chains one-by-one to get a chain of four;
2) Φ_2 : build pairs and then chains of four from pairs;
3) Φ_3 : allow all interactions, even those that build larger structures. When a larger structure that has assembly 6 as a sub- structure is built, detach any part not in the substructure.

These grammars are easily constructed. To save space, we omit the listing of Φ_1 and Φ_3. The grammar Φ_2 is listed in the appendix to this paper.

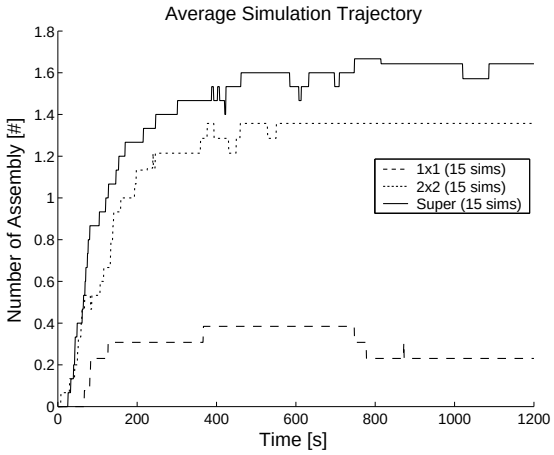


Fig. 8. Comparison of the systems induced by Φ_1, Φ_2 and Φ_3 using the average of 15 simulations of the full mechanics-based simulation of the programmable parts system.

Figure 8 shows the results of using the full simulation

with each grammar. We used $N = 10$ parts, $K_{ave} = 5 \times 10^{-4}$ and $\rho = 5$ parts/m^2. Each curve represents the number of assemblies of type 6 averaged over 15 trajectories. The grammar Φ_3 clearly out-performs the other two grammars in terms of the average number of product assemblies at the end of the simulation and in terms of the initial rate at which product assemblies are formed.

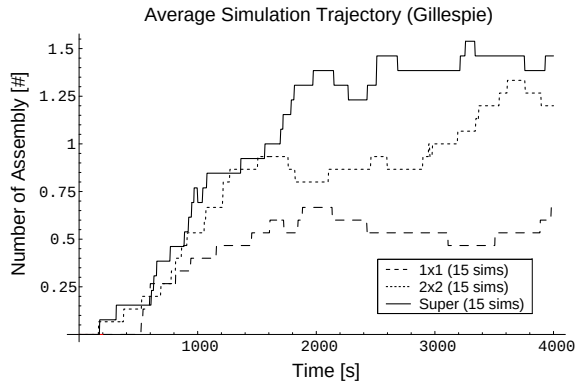


Fig. 9. Comparison of the systems induced by Φ_1, Φ_2 and Φ_3 using the average of 15 Gillespie simulations of the Markov process induced by the grammars.

Figure 9 shows exactly the same data, except that it was generated using a Gillespie simulation of the systems induced by the grammars and the measured basic rates in the table. The data sets from the two simulation methods are in agreement with respect to how well the grammars perform – although the timescale in Gillespie simulation is different. This is most likely due to the fact that the simulations are with $N = 10$ instead of $N = 60$ parts.

These experiments suggest that the two methods of simulating the actual system yield the same results when evaluating grammars.

VI. Discussion

We have developed a framework for programmed robotic self-assembly that describes how a model of the system can be induced by a given graph grammar program. The model requires that the kinetic rate constants for the allowable reaction types are known or can be measured. We demonstrate how a low-level simulation can be used to generate the rate constants so that any grammar can be represented in terms of rates and an induced grammar.

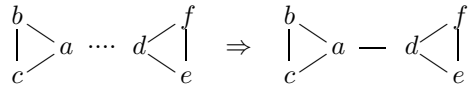
The model applies low-density systems that are well-mixed. As we stray away from such systems, we expect the model to be less able to predict the behavior of the system. However, grammars that perform well under the assumption of low-density are very likely to perform well at other densities – making the approach still useful in these regimes.

The next step in our research is to generate and optimize graph grammar programs for specific tasks such as assembling copies of a desired structure. We believe that the representation of the system as we have described here will facilitate this process, allowing us to do both stochastic search [10] and optimization quickly, and to perform stochastic model-checking [14], [9] and controller synthesis. This will hopefully lead to a practical method for designing and programming robotic self-assembling systems in general.

Appendix

A. 2x2 Pacman Grammar

Binary rules have the form



which can be rewritten more compactly as

$$abc \sim def \quad \rightharpoonup \quad abc - def,$$

where the first character in a triple (e.g. "a" in "abc") labels the programmable part edge involved in the interaction, the remaining characters labeling a tile in the counter-clockwise direction from this face. Thre are three possible connection types with this notation: " " represents no connection, "$\sim$" represents a physical connection which hasn't carried communication, and "$-$" represents an established connection. Unary rules have the form $a'bc \rightharpoonup def$. Using this notation, we write Φ_2 as follows:

$$\begin{array}{rcl}
aaa \sim aaa & \rightharpoonup & baa - baa \\
aab \sim aba & \rightharpoonup & ead - eda \\
dea - baa & \rightharpoonup & dea - iaa \\
dae - baa & \rightharpoonup & dae - iaa \\
eah - eda & \rightharpoonup & aaa \quad hda \\
dah - baa & \rightharpoonup & baa - baa \\
eha - ead & \rightharpoonup & aaa \quad had \\
dha - baa & \rightharpoonup & baa - baa \\
dah - iaa & \rightharpoonup & baa - baa \\
dha - iaa & \rightharpoonup & baa - baa
\end{array}$$

$$\begin{array}{rcl@{\qquad}rcl}
b'aa & \rightharpoonup & aaa & i'aa & \rightharpoonup & aaa \\
e'ad & \rightharpoonup & had & e'da & \rightharpoonup & hda \\
d'ea & \rightharpoonup & hea & d'ae & \rightharpoonup & hae
\end{array}$$

References

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

[2] 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.

[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] K. Dill and S. Bromberg. *Molecular Driving Forces*. Garland Science, 2003.

[5] R. M. Dirks, M. Lin, E. Winfree, and N.A. Pierce. Paradigms for computational nucleic acid design. *Nucleic Acids Research*, 32(4):1392–1403, 2004.

[6] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.

[7] K. Hosokawa, I. Shimoyama, and H. Miura. Dynamics of self-assembling systems: Analogy with chemical kinetics. *Artifical Life*, 1(4):413–427, 1994.

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

[9] K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of blackbox probabilistic systems. In *16th conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 202–215. Springer, 2004.

[10] J. C. Spall. *Introduction to Stochastic Search and Optimization*. Wiley, 2003.

[11] 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.

[12] P. J. White, K. Kopanski, and H. Lipson. Stochastic self-reconfigurable cellular robotics. In *Proceedings of the International Conference on Robotics and Automation*, New Orleans, LA, 2004.

[13] 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.

[14] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *14th conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.