

Brief Announcement: Self-Assembly as Graph Grammar as Distributed System

Aaron Sterling
Laboratory for Nanoscale Self-Assembly
Department of Computer Science
Iowa State University
Ames, Iowa, USA
sterling@cs.iastate.edu

ABSTRACT

In 2004, Klavins *et al.* introduced the use of graph grammars to describe—and to program—systems of self-assembly. It turns out that these graph grammars can be embedded in a graph rewriting characterization of distributed systems that was proposed by Degano and Montanari over twenty years ago. By applying techniques obtained from this observation, we prove a generalized version of Soloveichik and Winfree’s theorem on local determinism. We also obtain a canonical method to simulate asynchronous constant-size-message-passing models of distributed computing with systems of self-assembly.

Categories and Subject Descriptors

F.1 [Computation by Abstract Devices]: Models of Computation—*Relation between models*

General Terms

Theory

Keywords

Self-Assembly, Graph Grammar, Graph Assembly System, Grammars for Distributed Systems

1. INTRODUCTION

Two main research areas have studied algorithmic self-assembly: nanotechnology and robotics. Nanostructure self-assembly dates back to the 1980’s, when Seeman engineered “tiles” from DNA strands that could connect to other tiles [3]. Our focus, though, in this announcement, is on a theoretical advance that came out of the field of robotics: *graph assembly systems*, introduced by Klavins *et al.* in 2004 [2]. Graph assembly systems are a special class of graph grammars, and they provide a symbolic and topological characterization of a wide variety of systems of self-assembly. It turns out that graph assembly systems are a special case of a graph rewriting characterization of distributed systems that was proposed by Degano and Montanari over twenty years ago [1]. We explore that observation, so the theoretical questions of self-assembly—such as management of fault

tolerance—can benefit from thirty years of upper and lower bounds in distributed computing.

A *graph* $G = (V, E, l)$ is a triple, where V is a set of vertices, E a set of edges, and $l : V \rightarrow \Sigma$ a labeling function. If G is a graph, we sometimes write V_G and E_G to denote the vertices, or edges, of G , respectively. A *rule* is a pair of graphs $r = (L, R)$ where $V_L = V_R$. L is called the *left-hand side* of r , and R , the *right-hand side* of r .

Let G_1 and G_2 be graphs. A function h from V_{G_1} to V_{G_2} (often written $h : G_1 \rightarrow G_2$) 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) $l_{G_1} = l_{G_2} \circ h$. A rule r is *applicable to a graph* G if there exists an embedding $h : L \rightarrow G$. An *action* on a graph G is a pair (r, h) such that r is applicable to G as witnessed by embedding h .

Definition 1. Let $G = (V, E, l)$ be a graph, $r = (L, R)$ a rule applicable to G , and (r, h) an action. The *application of* (r, h) to G produces a new graph $G' = (V', E', l')$, defined as follows.

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

Definition 2. A *graph assembly system* is a pair (G_0, Φ) , where G_0 is the *initial graph* and Φ is a (finite) set of rules, called the *rule set*.

Intuitively, G_0 represents the initial configuration of self-assembling agents, before any binding rules have been applied; while Φ characterizes the binding rules of the system.

2. ANNOUNCEMENT OF RESULTS

In 1987, Degano and Montanari proposed a characterization of distributed systems based on graph rewriting [1], which they called “Grammars for Distributed Systems,” or GDS. We omit the full definition of GDS from this announcement, but briefly, a GDS is a triple $(\Sigma, \mathcal{D}_0, P)$, where Σ is an alphabet of events and processes that can legally appear in a distributed computation, \mathcal{D}_0 is an initial finite distributed system with no events, and P is a set of graph productions that characterizes legal computation steps. Degano and Montanari defined an ultrametric space of temporally ordered computations, and used this to prove that any weakly

fair GDS computation has a result that is final, *i.e.*, it converges to a limit to which no graph production can be legally applied. We use this fact to generalize the notion of local determinism in self-assembly, initially defined by Soloveichik and Winfree [4], who were interested in guaranteeing that a tile assembly system would always form a unique terminal assembly. We generalize their notion as follows.

Definition 3. Let $G = (\Sigma, \mathcal{D}_0, P)$ be a GDS. We say G is *locally deterministic* if the following holds for all computations $\{\mathcal{D}_i\}$ generated by G . For any $k > 1$ and any process $s \in \mathcal{D}_k$, let \mathcal{D}_j be maximal such that $\mathcal{D}_j \in \{\mathcal{D}_i\}$ and $s \notin \mathcal{D}_j$. Then there is exactly one production applicable to the parents (*i.e.*, immediate \leq -predecessors, where \leq is the temporal ordering) of s , and that production produces s in the location where it appears in \mathcal{D}_k .

In words, the initial graph and the productions of G are such that the local subsystems of any finite computation entirely determine their children at the future computation step when a production is applied to them.

THEOREM 1. *Let $G = (\Sigma, \mathcal{D}_0, P)$ be a GDS that is locally deterministic. Then all (finite or infinite) weakly fair computations generated by G have the same result.*

The proof idea is that self-assembling systems are assumed to be weakly fair: if something can bind to a particular location, it eventually will. Hence, the computation will converge to a final result. If the system admits two computations that converge to two distinct results, then those computations must differ at some finite stage, in which distinct subsystems share the same finite predecessors. That is impossible because of the local determinism of the system.

COROLLARY 1 (SOLOVEICHIK AND WINFREE [4]). *Let $\mathcal{T} = (T, \sigma, \Sigma, \tau, R)$ be a tile assembly system that is locally deterministic (in the sense of tile self-assembly). Then \mathcal{T} has a unique terminal assembly.*

The proof idea is that \mathcal{T} can be simulated by a graph assembly system, which, in turn, can be embedded in a locally deterministic GDS.

We also show that systems of self-assembly can simulate systems of distributed computing. Informally, graph assembly system G simulates distributed system \mathcal{M} if each configuration of \mathcal{M} can be mapped to a unique graph derivable from G so that legality of operations in \mathcal{M} is preserved (both operations inside a given processor, and also that messages sent between processors always eventually arrive), and, if all processors in configuration C have halted, then the corresponding graph derived from G is final.

THEOREM 2. *Let \mathcal{M} be an asynchronous message-passing model of distributed computing such that all processors run forever, and each processor can send messages of size bounded by some constant k . Then there is a graph assembly system (and, hence, GDS) that simulates \mathcal{M} .*

The weakness of the graph grammar characterization of self-assembly is that it captures only the topology, and not the geometric constraints of the system. This means that we can simulate any single processor (Turing machine, finite state machine, *etc.*) using self-assembling agents, but it may not be possible to simulate a network of distributed processors, because the communication between processors may interfere with the system’s ability to grow, depending on how

the agents embed themselves into their geometric environment. To provide a specific example, it is not possible for the Winfree-Rothemund Tile Assembly Model [6] to simulate a 3-consensus object in two-dimensional tile assembly—though it can be done in three dimensions—because there is no way to ensure that three independently-growing sub-assemblies can have wait-free access to a common decision point [5].

We are, however, able to demonstrate that 4-regular agents embedded in the plane can simulate a two-processor message-passing model.

THEOREM 3. *For any two-processor message-passing model \mathcal{M} of distributed computing in which each process runs forever and sends messages of constant size, there is a tile assembly system \mathcal{T} (in the standard Winfree-Rothemund Tile Assembly Model) that simulates \mathcal{M} in two dimensions.*

3. CONCLUSION

Perhaps the principal theoretical and practical barrier currently facing the field of self-assembly is the lack of management of fault tolerance. In order to import results about either crash failures or Byzantine failures from the world of distributed computing, we will need a better understanding of the relationship between what can be simulated topologically (*i.e.*, by GDSes) and what can be built when self-assembling agents are embedded into a particular geometric space.

4. REFERENCES

- [1] P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *J. ACM*, 34(2):411–449, 1987.
- [2] E. Klavins, R. Ghrist, and D. Lipsky. Graph grammars for self assembling robotic systems. In *ICRA*, pages 5293–5300. IEEE, 2004.
- [3] N. C. Seeman. De novo design of sequences for nucleic acid structural engineering. *Journal of biomolecular structure and dynamics*, 8(3):573–581, 1990.
- [4] D. Soloveichik and E. Winfree. Complexity of self-assembled shapes. *SIAM J. Comput.*, 36(6):1544–1569, 2007.
- [5] A. Sterling. Distributed agreement in tile self-assembly. In *DNA 15*, June 2009. To appear.
- [6] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.