Towards a common comparison framework for global-to-local programming of self-assembling robotic systems

Justin Werfel and Radhika Nagpal
Harvard University EECS

Self-assembling robotic systems are a class of modular robotic systems composed of many identical modules that, when mixed randomly, can bind together to assemble into complex shapes [1, 2, 5, 6, 14]. Modules in these systems have dynamic state and local binding rules that drive the overall self-assembly process. A key question is how to generate module rules such that the system assembles to form a particular prespecified global shape.

Recently, several groups have demonstrated decentralized, local agent rules for shape formation, not only in the context of self-assembly but also for self-reconfigurable robots and collective construction by mobile robots [3, 4, 7–11]. An important feature of these particular examples is that they achieve *global-to-local programming:* there is a principled way to derive local rules to achieve a prespecified global shape from a given class. In some cases, it has also been shown that the derived local rules are provably correct regardless of variations in the order and timing of agent movements and actions. It may be possible to apply many of these algorithmic approaches in the context of self-assembling modular robots. However, it is often difficult to understand and compare the theoretical properties of these global-to-local compilers because of the different settings—both the assumptions about agents (shape, computation and communication capability, self-propelling or not, homogeneous or bipartite, physical movement constraints, etc.) and the global shape descriptions can differ significantly between frameworks. For example, the MakeGraphGrammar algorithm by Klavins *et al.* [7] operates on homogeneous point agents that are randomly mixed, and generates agent rules that provably will terminate only at the completion of arbitrary desired topological graphs. By contrast, the family of CollectiveConstruction algorithms by Werfel *et al.* [11, 13] operates on bipartite agents (self-propelled robots and passive square blocks) with potentially restrictive movement assumptions, and provably produces arbitrary desired geometric structures in 2D without holes. Given such different settings, it is often difficult to see whether these algorithms can solve the same problems, and if so how they would compare in measures such as complexity, correctness, and parallelism.

In this workshop talk, we will present some initial steps toward comparing the theoretical properties of different global-to-local programming approaches in the context of self-assembling robotic systems. We do so by adapting the approaches to work in a common and generic framework. We will present three aspects of this problem:

1. We describe a simple, generic model for self-assembling robotic systems.

   In this framework, modules are homogenous square tiles that move (either self-propelled or driven by an external force, potentially at random) in a 2D space. They can attach to other modules along any of their four faces, and can detach again purposely depending on their interactions. A desired global shape for the assembly is described as a contiguous 2D grid region. Modules can have state associated with each of their four faces ("face state") and/or state associated with the module as a whole ("body

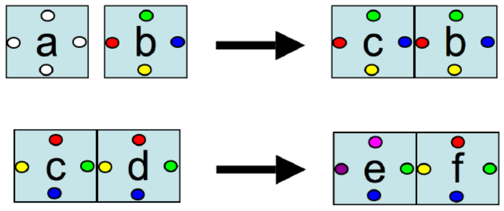


Figure 1: Body and face states (shown as letters and colored dots, respectively), and grammars. A rule in a grammar may involve a free module acquiring new state and orientation information when it binds to the structure (top), or it may involve state changes among already-bound modules (bottom).

state"); both can change as a result of interactions with other modules (Fig. 1). The module's local behavior is described by a *geometric graph grammar*, which is similar to the graph grammars decribed in [7] except that both body and face states are used in the interaction rules. This use of face states allows rulesets to constrain the relative direction of bindings and thus respect geometry in addition to topology. Rules in these grammars specify initial and final states of the modules involved and any attachments to be made or broken; when a set of modules is in the specified initial state, the rule applies and the modules transition to the specified final state. The complexity of the agent program can be determined by expressing the grammar as a lookup table, and counting the number of unique labels for the body/face states and the number of distinct rules.

2. We recast two existing algorithmic approaches to work in this common setting: the MakeGraph-Grammar approach from [7] and the "communicating blocks" variant of the CollectiveConstruction approach from [11].[1]

   The former requires modifying the MakeGraphGrammar approach to respect geometry; not only must the generated grammar achieve a prespecified geometry and not simply any topological equivalent, but it must also avoid defects and intermediate geometric shapes that physically block further progress (Fig. 2). We present a modified algorithm, MakeGridGrammar, and show that this algorithm can provably create a restricted class of 2D solid geometric shapes while guaranteeing no defects or deadlock. MakeGridGrammar works by embedding a tree graph in a principled way into a grid-based representation of a desired assembly, and modifying MakeGraphGrammar's tree-construction algorithm to take into account face state and enforce geometry. We also modify the CollectiveConstruction approach to work with homogeneous agents that are not self-propelling. Doing so requires expressing the shape formation process using explicit grammar rules. We show that the mapping in this case is quite natural and so the CollectiveAssembly algorithm preserves all the algorithmic guarantees of the original setting, such as provable construction of 2D solid shapes with no defects or deadlock, and without requiring modules to travel down narrow tunnels.

3. We compare the two algorithms using several criteria, including: (a) the complexity and scalability of the generated agent program; (b) the best-case parallelism possible; (c) the likelihood of approaching best-case performance; (d) the class of shapes for which correctness is provable; (e) the assumptions about limitations on possible module movement that the algorithm is able to accomodate.

   Briefly, the results of these comparisons are as follows.[1]

[1]Due to space limitations, our algorithms and analysis for the claims made here do not appear in this extended abstract, but will be made available for the workshop in a technical report currently in preparation.

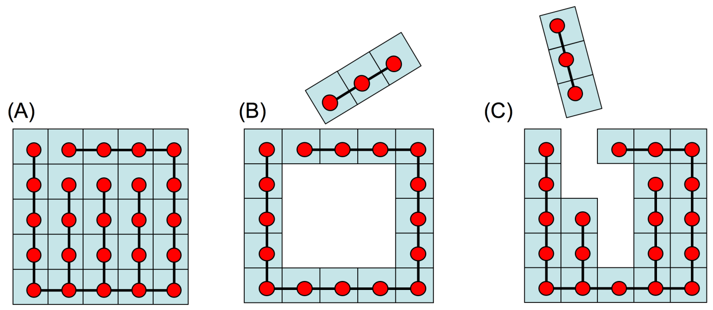


Figure 2: MakeGridGrammar involves embedding a tree graph into a desired structure. Such an embedding needs to be done with some care; the choice of tree affects how the structure self-assembles. If the tree is inappropriately chosen (A), then the structure may end up assembling in such a way that an internal space becomes entirely blocked off and subparts are physically unable to reach their intended binding sites (B), or the geometry of subassemblies may otherwise physically prevent them from coming together as necessary (C). MakeGridGrammar accordingly chooses trees to avoid these problems.

(a) The MakeGridGrammar approach generates agent programs whose complexity scales with structure size in the same way as CollectiveAssembly ($O(n)$, where n is the number of modules in the desired assembly). This result suggests that approaches using explicit coordinates (e.g., [3, 8, 9, 11]) are not inherently worse than automatically generated graph grammars.

(b) The MakeGridGrammar approach is able to achieve much greater best-case parallelism. For example, for a chain of length n, the best-case parallelism is 2 steps for MakeGridGrammar and $O(n)$ for CollectiveAssembly. This result stems from the fact that in the former case many subparts can assemble simultaneously and then merge, whereas the latter approach builds a connected assembly from a single initiation point. We generalize the theoretical bounds to more complex 2D structures for both algorithms.

(c) Performance is likely to be closer to the best case for CollectiveAssembly than for MakeGridGrammar. For the latter, where separate subparts assemble simultaneously, it can be a slow process for large subparts to find each other among a sea of others when needed, and to line up with the necessary relative locations and orientations. With CollectiveAssembly, free modules remain interchangeable until incorporated into their final location in the assembly, so that if the density of free modules is high, the waiting time for a module to reach an available site is likely to be small.

(d) MakeGridGrammar is provably correct for a smaller class of 2D structures than is CollectiveAssembly (Fig. 3A). The former generates rules that will terminate only at the desired assembly for assemblies that can be described by a straight backbone with straight side chains; the latter will produce any assembly without internally enclosed spaces, so long as any "alleys" in the assembly are wide enough to allow unbound modules to drift down them.

(e) CollectiveAssembly accomodates modules with stricter motion constraints than does MakeGridGrammar. MakeGridGrammar requires that modules and straight chains of modules be able to move freely down straight narrow tunnels (Fig. 3B), while CollectiveAssembly does not require movement in such physically restricted spaces. This difference may affect assembly speed and ease of physically realizing a system using each algorithm.

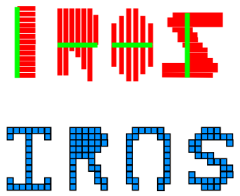
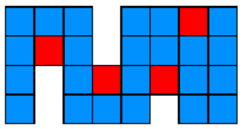
(A)  (B) 

Figure 3: (A) Examples of the structure classes for which MakeGridGrammar and CollectiveAssembly are provably correct. MakeGridGrammar (top): structures that can be drawn as straight backbones (green) plus orthogonal straight side chains (red). CollectiveAssembly (bottom): more general structures without internal holes.
(B) Examples of sites (red) situated down straight narrow tunnels. MakeGridGrammar requires that modules and straight chains of modules be able to traverse such tunnels.

We expect that in the future it will be possible to place other global-to-local shape algorithms in this common framework and compare them along the same axes.

We conclude by discussing three interesting types of shapes that present opportunities not exploited by either algorithm: (1) shapes with repeated elements, where some parts of the shape can clearly be identified as similar, and grammars can be made more compact by taking advantage of the repetition; (2) scale-invariant shapes, where a single grammar is used for a shape whose proportions adapt to the number of modules; and (3) environment-relative shapes, where some key aspects of the shape are determined by the environment in which self-assembly happens [12]. In biological organisms, shapes with these properties are quite common and play a role in both robustness and evolution. For these types of shapes, it may be possible to generate local agent rules of significantly less complexity, and assemble shapes with considerably more flexibility. We will illustrate this idea with both biological and artificial examples and motivate why these types of shapes are an interesting area for future study.

References

[1] Daniel Arbuckle and Aristides Requicha. Active self-assembly. In *Proceedings of 2004 IEEE International Conference on Robotics and Automation*, pages 896–901, New Orleans, Louisiana, 2004.

[2] J. Bishop, S. Burden, Eric Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Self-organizing programmable parts. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Canada, 2005.

[3] Michael De Rosa, Seth Goldstein, Peter Lee, Jason Campbell, and Padmanabhan Pillai. Scalable shape sculpting via hole motion: Motion planning in lattice-constrained modular robots. In *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, Orlando, Florida, USA, May 2006.

[4] Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, USA, 2003.

[5] Ying Guo, Geoff Poulton, Phil Valencia, and Geoff James. Designing self-assembly for 2-dimensional building blocks. In *ESOA'03 Workshop*, Melbourne, Australia, July 2003.

[6] Chris Jones and Maja Matarić. From local to global behavior in intelligent self-assembly. In *Proceedings of 2003 IEEE International Conference on Robotics and Automation*, pages 721–726, Taipei, Taiwan, 2003.

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

[8] Keith Kotay and Daniela Rus. Generic distributed assembly and repair algorithms for self-reconfiguring robots. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[9] Kasper Støy. How to construct dense objects with self-reconfigurable robots. In *Proceedings of European Robotics Symposium*, pages 27–37, Palermo, Italy, May 2006.

[10] Serguei Vassilvitskii, Mark Yim, and John Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, pages 117–122, Washington, DC, USA, 2002.

[11] Justin Werfel, Yaneer Bar-Yam, Daniela Rus, and Radhika Nagpal. Distributed construction by mobile robots with enhanced building blocks. In *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, Orlando, USA, 2006.

[12] Justin Werfel, Donald Ingber, and Radhika Nagpal. Collective construction of environmentally-adaptive structures. In *Proceedings of 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, USA, 2007.

[13] Justin Werfel and Radhika Nagpal. Extended stigmergy in collective construction. *IEEE Intelligent Systems*, 21(2):20–28, March-April 2006.

[14] Paul White, K. Kopanski, and Hod Lipson. Stochastic self-reconfigurable cellular robotics. In *Proceedings of 2004 IEEE International Conference on Robotics and Automation*, pages 2888–2893, New Orleans, Louisiana, 2004.