Generic Decentralized Control for a Class of Self-Reconfigurable Robots

Zack Butler*, Keith Kotay*, Daniela Rus* and Kohji Tomita*†

*Department of Computer Science
Dartmouth College
Hanover, NH, USA

†National Institute of Advanced
Industrial Science and Technology (AIST)
Tsukuba, Japan

Abstract

Previous work on self-reconfiguring modular robots has concentrated primarily on hardware and reconfiguration algorithms for particular systems. In this paper, we introduce a new type of generic locomotion algorithm for self-reconfigurable robots. The algorithms presented here are inspired by cellular automata, using geometric rules to control module actions. The actuation model used is a general one, presuming that modules can generally move over the surface of a group of modules. These algorithms can then be instantiated on to a variety of particular systems. Correctness proofs of the rule sets are also given for the generic geometry, with the intent that this analysis can carry over to the instantiated algorithms to provide different systems with correct locomotion algorithms.

1 Introduction

Our goal is to build robotic systems that are extremely versatile in both their structure and the tasks they can perform. This type of versatility is the hallmark of self-reconfiguring robots. Self-reconfiguring robots are systems composed of a large number of identical modules that can autonomously reshape themselves to fit the task at hand. These robots are massively parallel distributed systems, with each module thinking for itself within the group context.

Previous work in self-reconfiguring robots has concentrated on designing, engineering, and controlling particular systems. Several interesting robot designs have been proposed [2, 3, 9, 10, 11, 12, 13, 15]. For each of these systems, architecture-dependent algorithms that couple planning to the specific actuation have been proposed. This includes real breakthroughs in both centralized planning [4, 16] and decentralized planning [6, 7, 12]. This body of work has brought valuable insight into planning and control by focusing on designing and building hardware and developing algorithms coupled to specific hardware.

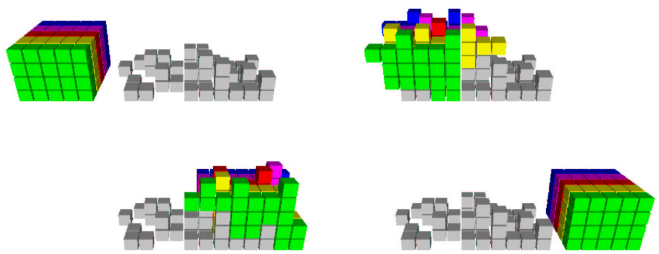


Figure 1: Four snapshots of a simulated locomotion task based on the rules of Fig. 4.

We believe we are now at a point where we can step back and examine more general questions about self-reconfiguration planning in an architecture-independent way. It is important to examine architecture-independent algorithms that can be instantiated for many different systems because they have the potential of providing a more general science-base for the field. By outlining general principles for reconfiguration planning, we hope to learn how to better design hardware and control algorithms. For example, the ability of a self-reconfiguring system to change shape can lead to "water-flow"-like locomotion algorithms that allow the robot to conform to the terrain on which it has to travel, as shown in Fig. 1. Such algorithms have the potential of working well in unstructured environments.

In most existing self-reconfiguring robot systems (e.g. [3, 4, 9, 14]), an individual module can move in general ways relative to a structure of modules by traveling on the surface of the structure. Specifically, an individual module is capable of: (1) linear motion on plane of modules; (2) convex transitions into a different plane; and (3) concave transitions into a different plane. The details for how to accomplish these motions are architecture-dependent.

In this paper we introduce a new class of distributed control algorithms based on the above motion abstractions. These algorithms are *generic*, in that they can

be instantiated for any self-reconfiguring robot where individual modules can move linearly on a structure of identical modules and make convex and concave transitions onto different planes of motion. Our control algorithms use local rules and are inspired by the cellular automata model. We illustrate our general approach to distributed control by developing a suite of algorithms for the locomotion task. More specifically, we describe three different locomotion algorithms with different assumptions, prove that they produce reliable and correct locomotion, and allow the shape of the robot to comply to the geometry of the terrain in the spirit of the "water-flow" metaphor. We also show that our correct abstract algorithms can be instantiated to three very different robots with their correctness properties intact. Finally, we illustrate our locomotion algorithms in simulations faithful to the characteristics of each hardware unit.

Each locomotion algorithm employs a small set of local rules we implement as a cellular automaton. Each rule requires a set of preconditions on the neighborhood of the cell and when activated, causes the cell to move to an adjacent configuration (see Figures 1 and 2). We represent the basic module of the robot as a cube, but our proposed abstraction can be replaced by any geometric structure that supports the formation of lattices.

2 Activation Models

The three sets of rules presented in this paper use different evaluation models for the modular robot. We represent a generic self-reconfiguring robot as a collection of cells, each cell corresponding to one robot module. We view the resulting structure as a particular type of cellular automata. Before introducing the specific distributed locomotion algorithms, we discuss several different models that support distributed evaluation and control for such systems.

Traditional cellular automata are evaluated as a group, such that the next configuration is generated from the current configuration by simultaneously evaluating all the cells. This model is not concerned with physical aspects of the underlying system. Since our robots are physical systems, we modify the evaluation model for traditional cellular automata to sequence the cell evaluation process[1].

We define three evaluation models, based on the relative delay of the activation of the cells. In the simplest model, which we call D_0, the cells are evaluated in a set cyclical order, and so there is no variation in the relative delay between any two cells. For the majority of the rule sets presented here, we use an activation model referred to as D_1. Under the D_1 model, a cell can delay activation at most one cycle relative to any other cell, or equivalently, one cell can activate at most twice before another cell activates once. This model relies on the homogeneity of the modules of the robot, which ensures that they run at approximately the same speed. We have implemented D_1 using the technique shown in Algorithm 1. Our most flexible model, D_∞, is used for the rule set in Sec. 5. In this model, a cell may delay for an arbitrary amount between activations (although infinite delay may halt progress of the system). This is implemented using Algorithm 2. This model requires more complex rule sets and analysis than the D_1 model, but allows for more variability of the individual cells, making it more robust for instantiation onto a distributed system.

[1]We can easily ensure no collisions in simulation by simply evaluating the cells one at a time, while in hardware a local locking system is required (note that cells at non-interfering locations in the group can activate simultaneously).

Algorithm 1 Instantiation of D_1

```
1: while (1) do
2:   flag = 0 in all cells
3:   while (Any flag == 0) do
4:     Choose unflagged cell at random
5:     Set flag = 1 for current cell
6:     while (Not all rules tried) do
7:       Evaluate random rule for current cell
8:       If rule applies, use it and break
```

Algorithm 2 Instantiation of D_∞

```
1: while (1) do
2:   Choose cell at random
3:   while (Not all rules tried) do
4:     Evaluate random rule for current cell
5:     If rule applies, use it and break
```

3 Locomotion without obstacles

The simplest locomotion algorithm is one which does not allow explicit obstacles (i.e. obstacles other than the floor). The rules shown in Fig. 2 are sufficient to move a robot represented as a rectangular array of cells eastward, where eastward motion is to the right. The rules are shown as productions, in which the left side represents the preconditions which must exist in the environment of the cell being processed, the "current cell", represented in this figure by a black square with a white dot. The right side of the rule represents the environment of the current cell after the rule has been applied. The rules in Fig. 2 require three different existence tests: whether a cell exists at a location,

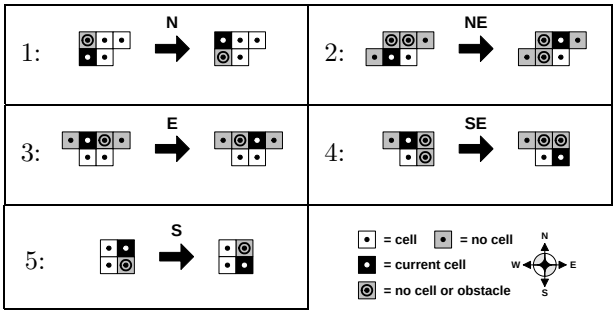


Figure 2: Five rules for eastward locomotion without obstacles, with the direction of motion given for each.

whether a cell does not exist at a location, and whether a location is empty. For example, Rule 5 in Fig. 2 can be applied only if there is an empty location below the current cell, and cells to the west and southwest. If Rule 5 is applied, then the result is that the current cell has moved by one in the southward direction.

The general idea of how this rule set works is that cells along the westward edge of the cell array move to the top of the cell array, where they then move eastward until reaching the eastern edge of the cell array. At that point, the cells move southward until no more motion rules apply. Thus, the motion is akin to that of a caterpillar tread.

3.1 Analysis

We now show that the rule set in Fig. 2 produces eastward locomotion on a robot represented as an array of cells that is initially rectangular. It is sufficient to show that 1) some rule from the rule set can always be applied to the cell array, 2) eastward motion must result from all possible sequences of rule activations, and 3) the array remains connected, i.e. the cell array always consists of one simply connected component. These claims are proven in the following three lemmas. The lemmas assume that the cell array meets minimum width and height bounds which do not preclude any rule from being applied. In the case of the rule set in Fig. 2 the minimum width is three cells and the minimum height is two cells.

Lemma 1 and Lemma 3 also assume that the D_1 activation method described in Algorithm 1 is used. This method produces a bound on the relative activation frequency of any two cells: a cell can be processed at most twice between two processing times of some other cell. For example, cell P could be processed before cell Q in round 1, and cell Q could be processed before cell P in round 2. The result is that Q was processed twice between the processing times of P.

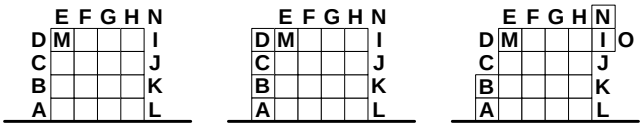


Figure 3: Diagrams for the proofs of Lemmas 1-3.

Lemma 1 *Given an initially rectangular cell array, in the absence of disconnections some rule from the rule set shown in Fig. 2 can always be applied.*

Proof: Considering the left diagram of Fig. 3, we see a generic cell array with border locations labeled A-L (for convenience cells are indicated without the black dot in the center). It is easy to see that if a cell X is located at any of the locations A-L and the rules in Fig. 2 are continuously applied only to X, then X will eventually come to rest at location L. This is because at any location only one rule can be applied: Rule 1 at locations A-C, Rule 2 at location D, Rule 3 at locations E-G, Rule 4 at location H and Rule 5 at locations I-K. Furthermore, by examination of the rule set it can be seen that placing a cell, X, at any of the locations A-L cannot satisfy the preconditions for a rule to be applied to any other cell.

In this proof, we denote cells by the letter at their initial location in Fig. 3. Now consider the diagram in the center of Fig. 3 in which cells have been positioned at locations A-D. If we apply the rule set to all cells, we see that only cell D can move, and only by using Rule 2 to move to location E. By the above argument, this cell will move to location L unless some other *moving* cell can interfere with its motion. Since D's motion on the surface of the array cannot cause any cell on the surface to move, the only other moving cells must result from D's original northeast move.

Indeed, D's original move allows the cell at location C to move northward using Rule 1. At this point C is free to follow D around the surface of the cell array. If it can be shown that C cannot interfere with D's motion (by preventing Rule 3, Rule 4, or Rule 5 from executing), then D will arrive at location L. C can only prevent D from executing Rule 3 or Rule 4 if C can move to the location immediately west of D. But C can only move eastward via Rules 2 or 3 which guarantee 2 free locations to the east, so neither Rule 2 or Rule 3 can move C to a location one location west of D. And, since Rule 5 has no preconditions to the north or east, C would have to move in front of D to interfere with D's motion. However, one cell can only move in front of another using Rule 2 followed by Rule 4, and since C can never be immediately west of D, it will not be able to execute Rule 2.

Thus we have shown that the cell originally at location D must move to location L. By using the same argument we can show that cell C must move to location K unless D can interfere with C's movement. While D is moving east along the north edge of the cell array it can prevent C from moving by not moving itself, but D and C cannot interact as shown above. While C and D are moving south along the east edge D can again prevent C from moving by not moving itself, but C must stay to the north of D since Rule 4 cannot be applied due to the presence of a cell on C's west face. Similarly, cells moving north on the west edge of the cell array must maintain their single file order. Therefore, the only possibilities for multiple moving cells to interact are at the corners of the cell array. At the northwest corner there is no problem because a cell can only execute Rule 2 when there is enough free space between it and some other cell on the north edge of the cell array so that after Rule 2 has been applied there is still free space between the two cells. At the northeast corner, however, there is a possibility for moving cells to interact. It will be shown in Lemma 3 that this interaction could theoretically lead to the disconnection of the trailing cell, but that the D_1 model prevents disconnection from happening. Here we assume the disconnection does not happen and therefore, cell C will reach location K.

Cells B and A will follow in order, assuming that the cell at location M cannot move. M must be the next cell to move other than A-D for the same reason D was the only cell that could move originally. The issue of whether M could move before A has passed by M's north face is dependent on the amount of delay cells might have in occupying locations which the rules allow. Under the D_1 model, M cannot move before A has passed by M's north face because before D can move to location H, the prerequisite for M to move, C has moved to location D, preventing M's move. Likewise for cells B and A. The result is that M cannot move until the cell originally at A is at location H. At that point M will move, starting a new cycle of motion which will create a new column on the eastern edge of the cell array. Thus we have shown that the movement is cyclic and will continue without bound and, therefore, a rule will always apply as long as the structure remains connected. □

Lemma 2 *Given an initially rectangular cell array, eastward motion must result from all possible sequences of continuous rule activations from the rule set of Fig. 2.*

Proof: Since by Lemma 1 some rule can always be applied, the cell array will move eastward as long as there are no oscillations between complementary rules. Since Rules 2-4 have an eastward component of motion and there is no rule that produces westward motion, only Rules 1 and 5 are complementary, i.e. have offsetting motions. But Rule 1 can only activate on the western edge of the cell array and Rule 5 can only activate on the eastern edge of the cell array. Therefore no oscillation can occur, and the cell array must move eastward. □

Lemma 3 *Given an initially rectangular cell array, no possible sequence of rule activations from the rule set shown in Fig. 2 can disconnect the cell array.*

Proof: By examination of the rule set it can be seen that no rule applied to a single cell on the surface of a cell array can cause that cell to become disconnected from the cell array, because every rule contains as a prerequisite the existence of a post-motion connecting cell. Thus, a disconnection can only occur as a result of some configuration of multiple moving cells. As shown in Lemma 1, the only possible location for interaction which results in more than simple delay is at the northeast corner of the cell array. Here, a cell moving from location H to location I (see center diagram of Fig. 3) satisfies the preconditions of Rule 3 for a trailing cell, allowing the trailing cell to move from location H to N. At this point, if the cell at location I moves to location J via Rule 5, the cell at location N will be disconnected (see right diagram of Fig. 3). Alternately, the cell at N can move to location O before the cell at location I moves, but the result is the same, a disconnection after the cell at I moves to location J (note that no rules apply to a cell at location O).

Although a disconnection is theoretically possible using this rule set, it would require the trailing cell to move at least three times between the move of the leading cell from location H to location I and the move of the leading cell from location I to location J. This is because at the time when the leading cell moves from H to I, the trailing cell can be no closer to the leading cell than location F since Rule 3 enforces a single empty cell location between moving cells on the north edge of the cell array. Thus, after the leading cell has moved from H to I, the trailing cell would have to apply Rule 3 three times to be at location N (four moves would be required to reach location O) before the move of the leading cell which disconnects the trailing cell. However, the D_1 simulation model we are using does not allow one cell to move more than twice between the moves of another cell. Therefore, given our simulation model the cell array must remain connected. □

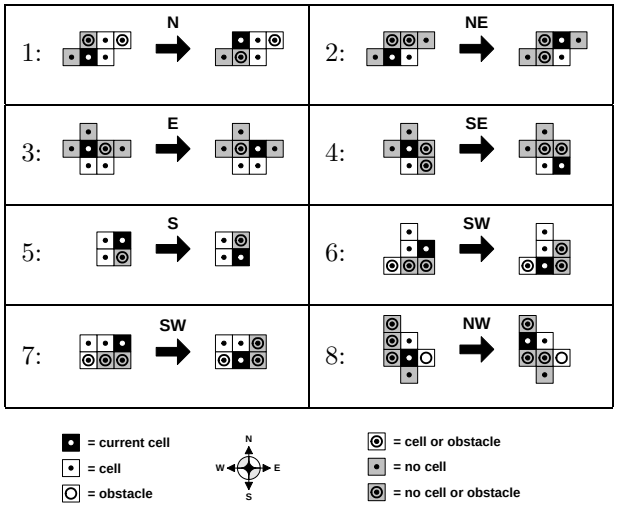


Figure 4: Eight rules for eastward locomotion with obstacles based on the rules of Fig. 2 with two additional SW rules and one new NW rule.

Theorem 4 *The rule set in Fig. 2 produces eastward locomotion on any initially rectangular array of cells.*

Proof: Lemmas 1, 2, and 3 prove that some rule from the rule set can always be applied to the cell array, eastward motion must result from all possible sequences of rule activations, and the array remains connected. Therefore, the rule set produces eastward locomotion on a rectangular array of cells. □

4 Locomotion over obstacles

It is possible to augment the rule set in Fig. 2 to allow the cell array to crawl over obstacles. The new rule set, Fig. 4, has eight rules. The three new rules allow for southwest and northwest cell movement which is needed for the cells to comply to the obstacle field (two rules are used to implement the southwest movement). Preconditions that involve the presence of obstacles are also added as shown. The basic idea is still the same, in that a cell moves upward along the west side of the group, across the top, and down the east side, however in the presence of obstacles the exact path is slightly modified. In our simulation, obstacles are represented as cubes, but cells do not connect to them, and the algorithm can be used with irregular obstacles given proper sensing.

Figure 1 shows four snapshots from a simulation run of the rule set of Fig. 4. In this case, the robot is composed of five separate layers, each running the rule set independently. The observed motion of the robot is very compliant to the terrain and appears to flow over

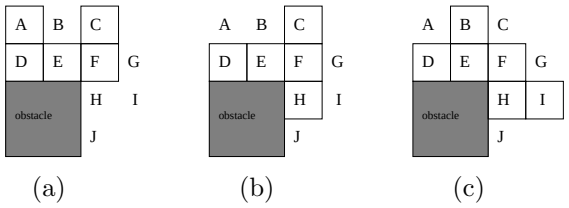


Figure 5: An example of cells conforming to the front of an obstacle, used in the proof of Theorem 5.

the obstacles. For this rule set, the maximum obstacle height is one less than the height of the cell array.

4.1 Analysis

To prove the correctness of this rule set, we begin with the correctness of the five rule set shown above. First, note that in the absence of obstacles, the current rule set reduces to the five rule set: the NW rule (8) can never apply without obstacles present, and the SW rules (6, 7) will not, since in the absence of obstacles columns are filled one at a time, so there could not be an empty space to the west of a cell on the east face. We now explore the situations in which obstacles are encountered, and show that for each, progress will continue without deadlock or disconnection.

Theorem 5 *The rule set of Fig. 4 will produce eastward locomotion for any (initially rectangular) group of cells over any obstacle field that is (a) shorter everywhere than the initial height of the group, and (b) everywhere supported from directly below.*

Proof: The group of cells will locomote as described by the five-rule set until the first obstacle is reached. At this point, the cells moving to the east edge of the group will settle on this obstacle using the Rule 5 just as they would settle on other cells in the no-obstacle case. We now show that the cells that will make up the next column (assuming the obstacle is shorter in the next column) will arrive there without disconnecting the group.

Once the column on top of the obstacle is completed, the next cell will execute Rule 4 followed by Rule 5 until it reaches the same height as the lowest cell in the previous column. If the obstacle is of the maximum height, this cell will simply extend the top row, with empty space above and below it, as shown as cell F in Fig. 5a. We must now show that the following cells will continue to their correct destinations and not disconnect. We show this using the notation of Fig. 5a to describe positions in the shape.

C will go to location G before A can move. A can therefore get only as far as C before G moves to H (using Rule 7). Note that during this process, F can't

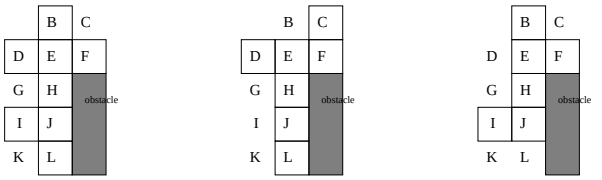


Figure 6: An example of cells climbing the back of an obstacle, used in the proof of Theorem 5.

move. This leads to the configuration of Fig. 5b. A (now at location C) will go to G and then I before another cell can get on top of it (as shown in Fig. 5c), before using Rule 6 to move to J.

Now we must show that the cells beginning motion at the back of the group will be able to actuate in the correct order to produce motion without disconnections (and while maintaining the general arrangement of cells across the top to ensure no deadlocks).

First we note that due to the nature of Rules 5-7, there will never be a cell with obstacles both on its immediate left and right, so there will always be at least two cells in any row between two obstacles. We examine two cases based on the relative heights of the westernmost two columns: first, where the obstacle in the westernmost column is taller than (or the same height as) that of the second column, and second, where the obstacle in the westernmost column is shorter. In this latter case, there are some cells at the bottom of the west column that have an obstacle to their east (e.g. H, J, L in Fig. 6a).

In the first case, none of the cells in the western column will have obstacles to their east, and so Rule 8 will never apply. This is the same as in the five-rule case, and the argument there follows. In the second case, the cell(s) with obstacles to their east will move NW using Rule 8 once the opportunity presents itself.

The second column before the obstacle will actuate from the top down as in the no-obstacle case, eventually reaching a state similar to that of Fig. 6a. Note that as the cells in this column move upward, the cell at L cannot move NW. Only once the last cell in the column (cell K) moves to D (as in Fig. 6b), cell L can move to I. K may be as far as B by this point, but not further. If K is at location B, there are two spaces between the cells (see Fig. 6c), which can be transiently increased to at most three as they move over the surface of the group. But since Rule 2 requires four consecutive empty spaces around the perimeter, E cannot move until all cells below that have an adjacent obstacle have moved past it. In addition, the only rules that could apply to cells H and J (which have a cell to their north and an obstacle to the east) are Rules 5-6, both of which require two adjacent cells at the bottom of the column to the east. This cannot happen since by the time one cell moves NW, the spots above and below it are guaranteed to be empty. □

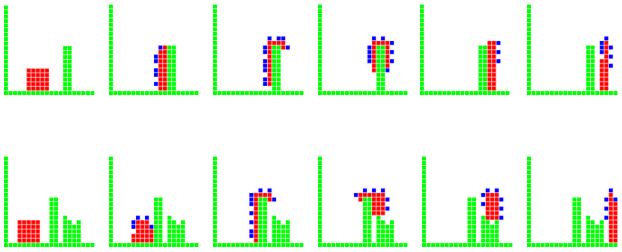


Figure 7: Snapshots of two different simulations based on the D_∞ rule set.

5 D_∞ rule set

In this section, we discuss briefly an extension to the locomotion algorithm that operates under the D_∞ activation model. This rule set also has the ability to climb over obstacles higher than the initial height of the group. It is composed of 22 rules and uses only the Moore neighborhood (eight adjacent cells). Each unit is in one of two states: activated and inactivated. Only activated units are permitted to move. The rules are classified into three types: five activation rules, 12 moving rules for activated units, and five inactivation rules. Two examples of simulated sequences using these rules are shown in Fig. 7. Gray, dark gray, and black squares represent an obstacle, an inactivated unit, and an activated unit, respectively. Correctness of this rule set can be shown through the use of a loop invariant that represents all valid configurations. We can prove that all cell motions that start in a valid configuration lead to another valid configuration. In addition to this basic rule set, we are also investigating techniques to automatically switch from one rule set to another based on the local form of the environment.

6 Instantiations

A key feature of our abstract locomotion algorithms is that they can be instantiated and realized on a variety of self-reconfiguring robot systems. This primarily requires developing actuation primitives for the system that can realize the types of motion called for by the cellular automata. In this section we discuss three different instantiations. We have currently developed these instantiations in simulation and are working on implementing the control on the hardware. It should also be noted that there exist 2-D self-reconfiguring

Figure 8: Simulation sequence of fracta modules.

systems, those of Hosokawa *et al.* [3] and Chiang and Chirikjian [1] that have the exact geometry and motion capability of our abstract model.

6.1 Meta-modules

The simplest way in which these automata may be instantiated onto several different systems is through the use of *meta-modules*. For many systems, it is possible to construct a group of several non-isotropic modules in such a way that the group has isotropic motion capability of the type required by the algorithms presented here (motion in a unit lattice over the surface, convex and concave transitions). For the Molecule, a structure called a *tile* has been developed with this property [5], while the *grain* meta-module was developed for the Crystalline Atom [10]. These meta-modules can then immediately use the rules of the automata, although communication between the individual modules would be necessary to determine the meta-module's neighbors and coordinate its actuation. The use of meta-modules requires many more modules to achieve the same amount of reconfigurability. Therefore, instantiations of the rules on to individual modules may be preferable.

6.2 Hexagonal lattice systems

The 5 rule set can be instantiated to systems with hexagonal modules with minor modification. Systems of this nature include the Fracta system [12] and the metamorphic robot of Chirikjian *et al* [9]. The adapted set of rules are shown in Fig. 9. We assume the D_1 model and no obstacles as for the original rule set. Because the lattice configuration is different, we assume that the initial configuration is a trapezoid instead of a rectangle, with a top row of at least four units.

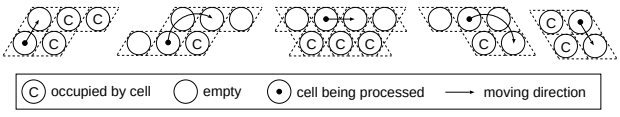
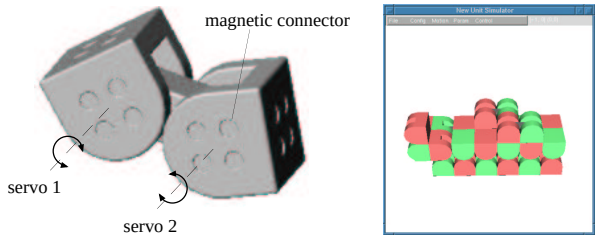


Figure 9: Rule set for hexagonal lattice modules.

The rules are obtained by tilting the original rules, and adding one cell condition to the east rule. Directions of tilt are shown in the figure around each cell as a parallelogram. Rules applied to the left part of the group (north and northeast rules) are tilted right, and the rules for the right part (southeast and south rules) are tilted left. The east rule combines them, resulting in an additional cell condition.



(a) (b)

Figure 10: (a) One module of the system of [8] and (b) a screenshot of the simulated instantiation of the five rule set.

A simulated sequence is shown in Fig. 8. Units are shown in black and the floor in gray. The figure shows eastward locomotion of the group.

The correctness proof is similar to the proof in Sec. 3. One difference is that the northeast rule makes a unit move one step further from the point of view of the left tilt. When another moving unit is still on the same height of the top row, the unit might move eastward over this unit, which does not happen using the original rule. This case happens when the top row contains only three units. By restricting robots to have at least four modules on the top row, we can ensure eastward progress.

6.3 Modular Transformer (MTRAN)

Another system that we are currently investigating is that of Murata *et al.*[8]. This system is quite powerful, but the modules are somewhat unusual, as can be seen in Fig. 10a. A module in this system cannot connect to neighbors on all sides, as is assumed in our abstract geometric model. Therefore, at least two 2-D layers must be present to maintain connections between all modules. In addition, each module consists of two articulated bodies, and so it inhabits a configuration space larger than the two dimensions of the simple automata described here.

Despite these complications, it is possible to instantiate the locomotion algorithms to this system. We have done this with the five rule set of Sec. 3. Neigh-

bor relationships translate in a straightforward way, and movement of a module along a straight line is also easy. For turning around corners, we have had to add the concept of a helper, and explicit message passing, since a module will need its neighbor to perform the actuation to make the northeast and southeast transitions. Proper message passing is also required to ensure that the group remains connected despite the limited connection ability of the individual modules. A distributed simulation has been implemented as the instantiation of these rules and has produced successful forward locomotion. The results of this simulator can be displayed in the GUI developed for these modules at AIST, as shown in the screenshot in Fig. 10b.

7 Extensions

In addition to locomotion over obstacles, we are currently investigating using this control approach for other tasks. One small set of added rules (which can be added to either of the sets of rules presented here) allows the group of cells to duck under overhanging obstacles and thereby navigate tunnels and fill caves. Another extension allows a 3-D group of cells (such as shown in Fig. 1) to stop, regroup into a cube and turn 90° when a tall obstacle is detected. A larger set of rules that involves giving the cells more state allows a group of cells to split in two halves that locomote in opposite directions and recursively split again.

8 Conclusion

Previous work in modular robotics has focused on algorithms that apply to particular systems. This paper has presented a set of generic locomotion algorithms for modular robots based on cellular automata. These algorithms apply to an abstract model and have been shown to produce correct locomotion without deadlock or disconnection. The algorithms can then be instantiated on a variety of different robot systems.

Acknowledgments

This work was done in the Dartmouth Robotics Lab. The authors would like to thank the members of the Distributed System Design Group at AIST for insightful discussion and for providing the simulation tool used with the instantiation of Sec. 6.3. Support for this work was provided through the NSF CAREER award IRI-9624286 and NSF awards IRI-9714332, EIA-9901589, IIS-9818299, and IIS-9912193.

References

[1] C.-H. Chiang and G. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10(1):91–106, 2001.

[2] T. Fukuda and Y. Kawakuchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. In *Proc. of IEEE ICRA*, pages 662–7, 1990.

[3] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Koruda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. of IEEE ICRA*, pages 2858–63, 1998.

[4] K. Kotay and D. Rus. Locomotion versatility through self-reconfiguration. *Robotics and Autonomous Systems*, 26:217–32, 1999.

[5] K. Kotay and D. Rus. Scalable parallel algorithm for configuration planning for self-reconfiguring robots. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers*, Boston, 2000.

[6] J. Kubica, A. Casal, and T. Hogg. Complex behaviors from local rules in modular self-reconfigurable robots. In *Proc. of IEEE ICRA*, pages 360–7, 2001.

[7] S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji. Self-repairing mechanical systems. *Autonomous Robots*, 10:7–21, 2001.

[8] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, pages 2210–7, 2000.

[9] A. Pamecha, C-J. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proc. of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*, 1996.

[10] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with unit-compressible modules. *Autonomous Robots*, 10(1):107–24, 2001.

[11] W.-M. Shen, P. Will, and A. Castano. Robot modularity for self-reconfiguration. In *SPIE Conf. on Sensor Fusion and Decentralized Control in Robotic Systems 2*, 1999.

[12] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Trans. on Robotics and Automation*, 15(6):1035–45, Dec. 1999.

[13] C. Ünsal, H. Kiliççöte, and P. Khosla. A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, 10(1):23–40, 2001.

[14] M. Yim. A reconfigurable modular robot with multiple modes of locomotion. In *Proc. of JSME Conf. on Advanced Mechatronics*, Tokyo, 1993.

[15] M. Yim, D. Duff, and K. Roufas. PolyBot: a modular reconfigurable robot. In *Proc. of IEEE ICRA*, 2000.

[16] E. Yoshida, S. Murata, A. Kaminura, K. Tomita, H. Kurokawa, and S. Kokaji. Motion planning of self-reconfigurable modular robot. In *Proc. of Int'l Symposium on Experimental Robotics*, 2000.