Chapter 2
Lattice Automata for Control of Self-Reconfigurable Robots

Kasper Stoy

Abstract Self-reconfigurable robots are built from robotic modules typically organised in a lattice. The robotic modules themselves are complete, although simple, robots and have onboard batteries, actuators, sensors, processing power, and communication capabilities. The modules can automatically connect to and disconnect from neighbour modules and move around in the lattice of modules. The self-reconfigurable robot as a whole can, through this automatic rearrangement of modules, change its own shape to adapt to the environment or as a response to new tasks. Potential advantages of self-reconfigurable robots are extreme versatility and robustness. The organisation of self-reconfigurable robots in a lattice structure and the emphasis on local communication between modules mean that lattice automata are a useful basis for control of self-reconfigurable robots. However, there are significant differences which arise mainly from the physical nature of self-reconfigurable robots as opposed to the virtual nature of lattice automata. The problems resulting from these differences are mutual exclusion, handling motion constraints of modules, and unrealistic assumption about global, spatial orientation. Despite these problems the self-reconfigurable robot community has successfully applied lattice automata to simple control problems. However, for more complex problems hybrid solutions based on lattice automata and distributed algorithms are used. Hence, lattice automata have shown to have potential for the control of self-reconfigurable robots, but still a unifying implementation based on lattice automata solving a complex control problem running on physical self-reconfigurable robot is yet to be demonstrated.

2.1 Self-Reconfigurable Robots

The self-reconfigurable robot community grew out of the distributed autonomous robot systems community. The idea was that if multiple robots could automatically form physical bonds between each other the combined robot collective could adapt its shape and functionality in response to the environment and tasks. The basic scenario

K. Stoy (✉)
IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
e-mail: ksty@itu.dk

G.Ch. Sirakoulis and A. Adamatzky (eds.), *Robots and Lattice Automata*,
Emergence, Complexity and Computation 13, DOI 10.1007/978-3-319-10924-4_2

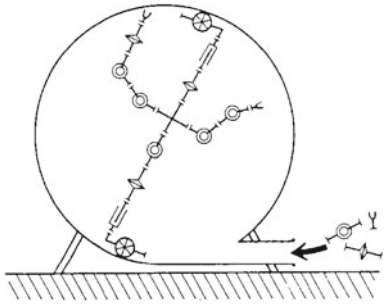

Fig. 2.1 This figure shows the original scenario that motivated the need for self-reconfigurable robots (Courtesy of Fukuda, © 1988 IEEE)

that motivated self-reconfigurable robotic research given by Fukuda et al. [8], shown in Fig. 2.1, was that individual robots could move into a storage tank through a narrow passage and once inside they could assemble for the purpose of cleaning the storage tank.

This vision of self-reconfigurable robots was and is still today attractive. However, the scientific challenges involved in realising this vision are significant. One aspect is the mechatronic realisation of self-reconfigurable robots and another central to the topic of this chapter is the question of their control.

In many self-reconfigurable robots modules are organised in a lattice structure like atoms in a crystal. These are called lattice-type self-reconfigurable robots. In these robots modules can move between lattice positions and thereby change the overall shape of the robot. The lattice organisation simplifies control of self-reconfiguration, because assumptions can be made about the precise position of neighbour modules and hence connection between modules can be performed open-looped.

Given the lattice organisation of self-reconfigurable robots, lattice automata are a natural basis for their control. However, another equally attractive feature of lattice automata is that each individual automaton acts independently and autonomously. This is crucial for self-reconfigurable robots because decoupling the controllers of individual modules will make the robot more robust to failures. A single failed module will not cause the whole system to fail which, for instance, is the case for centralised control strategies. Another desirable characteristic of lattice automata is the locality of their rules. Typically, the rules only consider the position and state of neighbour cells. These rules have a natural mapping to the sensors and the communication system that modules have which only provide functionality for inter-module communication and detection.

Given the match between the features of lattice automata and requirements of self-reconfigurable robots, researchers enthusiastically applied lattice automata to self-reconfigurable robots. Early work demonstrated how simple local rules with some noise added could make a desired configuration emerge through self-reconfiguration [11]. Another focus was the use of lattice automata to allow a self-reconfigurable robots to perform locomotion by moving modules from the back of the robot to the front [3, 13].

While this work demonstrated the potential of a lattice automata-based approach control, there is still a risk that using this hand-coded rulesets the self-reconfigurable robot could reach dead states where no rules applied. In order to be sure no dead states existed proofs were developed for specific rulesets [5, 20]. Another practical problem was the development of the rule-sets by hand, which for physical robots became quite large (e.g. 927 rules in [13]). It then became crucial to develop methods that could automatically generate rule-sets given a desired behaviour. A possibility that was explored was the use of reinforcement learning [19] and evolutionary algorithms [12]. However, both for the human rule-set designer and the automatic algorithm it became difficult to device rule-sets bottom-up given a complicated task due to combinatorial explosion of the configuration space.

A possible answer is to only control critical parts of the self-reconfiguration and allow a looser, distributed control algorithm to control the rest of the self-reconfiguration process. This simplification makes it possible to make a global-to-local compiler that based on a three-dimensional shape could generate a set of rules that would realize this shape [16]. A useful extension is to have strict rule-sets in critical areas of the robot (e.g. where there was a risk that modules may be disconnected from the structure) and let the modules move randomly in other areas [15].

Most of this work is concerned with controlling the self-reconfigurable robot itself without considering the potential of having the robot adapting to its environment. A notable exception is Bojinov et al. [1] who used rules with conditions based on the neighbour being an obstacle to creating grasping hands and other interesting functional structures. However, this line of work has not been picked up again. Lattice automata also lost traction in the self-reconfigurable robotics community because it had not been possible to create mechatronic modules that reliably could produce the motions used in the lattice automata model (e.g. rotate around a neighbour module, slide along a surface of modules, etc.). Hence, there is to this day a worry that a lattice automata based algorithm would never find practical use on a physical system. However, this may change as new mechatronic implementations are emerging that do in fact implement these motion primitives [14]. Hence, this is an exciting time for self-reconfigurable robots and lattice automata because they may finally come together to form the basis for controllable and useful self-reconfigurable robots.

2.1.1 Origin, Features, and Applications

The concept of self-reconfigurable robots was from the beginning inspired by multicellular organisms [7, 9]. The idea being that from a limited number of cell types a huge number of organisms are and can be created. For instance, an organism as complex as the human consist of about 100 trillion cells, but there are only two hundred different cell types. Hence, from an engineering perspective you could design and implement a few robotic cell types and then on the fly assemble them into a specific robot depending on your need. This concept is in the self-reconfigurable robotic community referred to as versatility. A shortcoming of mechatronic modules

is that unlike natural cells they cannot grow and divide hence another mechanism is needed to simulate growth. The mechanism used is self-reconfiguration. Instead of modules dividing and growing the robot can change its shape by rearranging the way modules are connected. In other words, modules can wander on the surface of other modules. While this is less common in nature it does happen and is known as morphallaxis. The typical example is the small fresh water animal Hydra which if cut in two can reorganize its tissues to become two hydras of roughly half the original size. As a side point there is also evidence that suggests that Hydras do not age. In fact, it is a truly remarkable animal.

Another feature of self-reconfigurable robots is robustness. Given the robot consists of many independently functioning modules, failure of one module is not critical to the functionality of the whole robot. Even if a module is placed in a critical region of the robot e.g. connecting two parts, it may be possible to replace it through self-reconfiguration. The conceptual robustness of the system of course also requires the controller to be distributed otherwise the control reduces the robustness of the entire system.

A self-reconfigurable robot is based on only a few different module types and these module types can be mass-produced. Hence, although the assembled robot is quite complex the cost of individual modules can be kept relatively low.

Together these features could provide us with robot technology that is particularly well suited for applications where the tasks are not known in advance, where the transportation cost of equipment is significant, and where robustness is important. A clear application is extra planetary exploration, but currently the most successful modular robotics company is creating robots for educational and entertainment purposes.[1]

2.1.2 Mechatronic Implementation

We have so far discussed self-reconfigurable robots at the conceptual level, but not how they are implemented in practice.

A module of a self-reconfigurable robot is a complete robot by itself. Typically, a module has sensors, actuators, processor, battery, and means of communication:

- Sensors are often limited to infrared transceivers that allow modules to detect nearby obstacles. These transceivers also often are used to communicate with neighbour modules. Otherwise, sensors are mostly internal and include encoders and accelerometers.
- Actuators include various forms of electrical motors to control the motion of a module as well as connector mechanisms.
- Processors used to be relatively small, embedded ones, but given the advance in terms of energy efficiency and computation power processors employed today

[1] http://modrobotics.com, [Online], retrieved 28/1/2013.

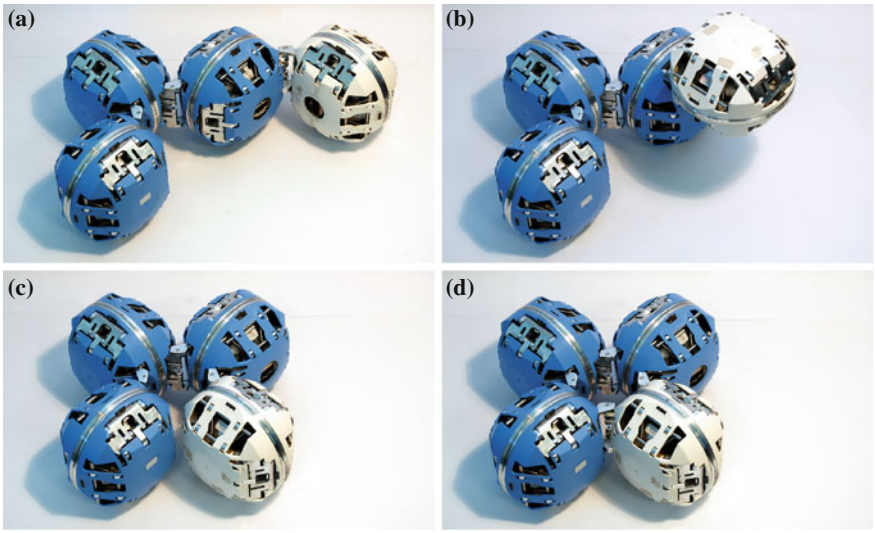

Fig. 2.2 The ATRON self-reconfigurable robot is performing one self-reconfiguration step. **a-b** First, the *top right*, *dark* module rotates the *white* module to its new position. **c-d** Once at the new position the *white* module extends connectors to attach to the new neighbour module as can be seen in the *bottom right* photo between the *dark* and *white* module

typically provide enough computation power to be able to run embedded variants of Linux.
- Batteries are typically Li-Ion allowing modules to functioning for an hour or two.
- Communication may be based on infrared communication, Bluetooth, or WiFi communication. In some cases electrical contact is made between neighbour modules allowing the modules to communicate across a shared CAN bus or similar technology.

In a typical self-reconfigurable robot, a self-reconfiguration step consists of a series of small steps as illustrated in Fig. 2.2. First a module disconnects from some of its neighbours, it then moves to a neighbour lattice position, and, finally, extends connectors to attach to neighbour modules at the new position.

The mechatronics design of self-reconfigurable robots is a major challenge given the physical constraints and the high requirements in terms of functionality. However, mechatronics is not the focus of this chapter so the interested reader can find more information on this topic in [10, 18].

2.2 Assumptions of Lattice Automata

As we have already argued lattice automata have features that match the desired features of a controller for self-reconfigurable robots. However, the match is not perfect. In fact, there are several problems one has to consider when applying lattice

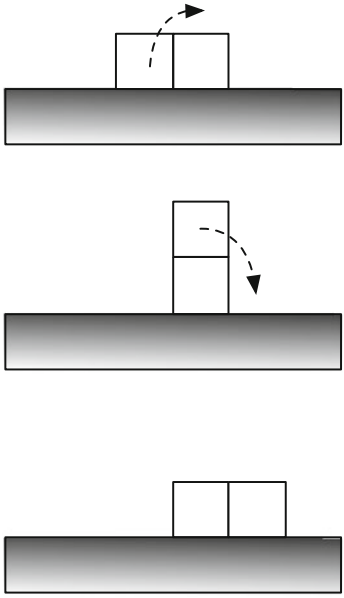

Fig. 2.3 This figure shows the two steps necessary to move a simulated self-reconfigurable robot consisting of two modules, represented by the *squares*, one step forward

automata as a basis for control of self-reconfigurable robots. We will introduce these challenges here and then continue to the actual application of lattice automata in the context of self-reconfigurable robots in the following sections.

Let us start by considering the simple motion sequence shown in Fig. 2.3. In this figure, two modules represented by squares are resting on the ground. The two-module self-reconfigurable robot can move forward by moving the rear module on top of the front module and then down in front of the module it is now resting on. This movement recreates the start configuration of the self-reconfigurable robot and the motion sequence can be repeated to generate forward locomotion. Implementing a controller based on lattice automata that realizes this concept is very simple. However, before we do this let us introduce some basic notation. There are eight directions that are relevant to these modules let us use the compass directions to identify those. For example, NE—north-east—is up and to the right of the current module. We now define the states of lattice positions, which can either be *Empty* or *Module*. Finally, we define a function *State* that maps a direction to a state. Using this notation the lattice automata rules resulting in the self-reconfiguration sequence shown in Fig. 2.3 could be as follows:

$$if\,(State(E) == Module)\;\; Move(NE) \tag{2.1}$$

$$if\,(State(S) == Module)\;\; Move(SE) \tag{2.2}$$

This may appear trivially correct, but it could in fact go wrong if there are more modules present. Even though, cell positions in a virtual world or simulator can be turned on or off easily, this is not the case here because the physical modules cannot be turned on and off. We have to simulate on and off by moving the modules. Hence, when one is turned off a neighbour cell has to be turned on. We also need to ensure that the cells a module passes through to reach the new *on* cell are free. Hence a ruleset like this is necessary:

$$if \begin{pmatrix} State(E) == Module \wedge \\ State(N) == Empty \wedge \\ State(NE) == Empty \end{pmatrix} \; Move(NE) \tag{2.3}$$

$$if \begin{pmatrix} State(S) == Module \wedge \\ State(E) == Empty \wedge \\ State(SE) == Empty \end{pmatrix} \; Move(SE) \tag{2.4}$$

This set of rules assumes that the physical module has to pass through a cell to get to the desired cell. Although this appears to be a sound rule set there are still problems. One problem is that a module does not transition instantly from one cell to another; in fact, physical modules typically take on the order of several seconds to make a transition. The implication of this is that there is significant period of time between a module detects a cell to be *Empty* and it has moved into this cell. This opens up for problems as other modules may perceive the cell as *Empty* and start a transition into it, leading to several modules moving into the same cell. From a lattice automata point of view and from the point of view of the moving module in our simple example a trivial solution could be that all cells a moving module has to pass through are marked as *Module* while the module is transitioning. However, this is impossible, because a cell can only be marked if there is a physical module in the cell able to transmit this information. A possible way to reduce this problem is to place proximity sensors in such a way that they minimize the amount of time a cell is wrongly categorised. A module leaving a cell is sensed as late as possible and a module entering a cell is sensed as early as possible. However, this of course does not solve the problem, but only reduce the probability that it will happen. One approach to solve this is to use communication to perform mutual exclusion, but in the worst case this requires time proportional to the number of modules because in a ring configuration with one hole the whole ring has to be informed. In order to completely solve this problem the lattice automata controller has to be complemented with a global communication and coordination mechanism. However, a better alternative is probably to use a roll-back strategy where modules attempt to move into cells as directed by the lattice automata, but if the module senses collision with another module moving into the same space it can roll back to its original position [6]. This might in rare cases lead to dead locks if there is a cycle, but the approach is attractive because it does not rely on global information.

Another complication is that the assumption of shared knowledge of orientations is not trivial for modular robots. Either this information has to come from embedded sensors, accelerometer and gyroscopes, or though a global coordination scheme.

Finally, several other practical issues also need to be handled: (1) one has to ensure that modules do not become disconnected from the configuration during the self-reconfiguration process because this may cause them to fall down and break, (2) one has to ensure that modules do not create hollow sub configurations that cannot be filled or where modules can be trapped.

In summary, when applying lattice automata to modular robotics one has to be concerned about the following problems.

- Motion constraints
- Mutual exclusion
- Spatial orientation
- Disconnection
- Hollow configurations

These problems arise from the physical nature of the modules and their ability to move in parallel. If not handled carefully, the problems may require global information, which again may reduce the responsiveness, scalability and robustness of the self-reconfigurable robot. While these problems are important, discussing them in detail is outside the scope of this brief introduction, but please refer to [16, 18] for details.

2.3 Lattice Automata-Based Control

Despite the complications outlined in the previous sections, many interesting examples of the use of lattice automata in the context of self-reconfigurable robots can be found in the literature.

One of the most thoroughly studied examples is cluster-flow locomotion of cubic self-reconfigurable robots, which is a generalisation of the trivial example we gave in the previous section. Butler et al. have in a series of papers extended the use of lattice automata rules for cluster-flow starting in two dimensions [3] and in later work moving to three dimensions. The algorithms were also able to handle obstacles in the environment [4]. A nice aspect of this work is also that since it has a strong theoretical basis in lattice automata it was possible to proof sufficiency and correctness for some of the rule sets. A fundamental problem, however, is that as the task and configurations become more varied the rule sets become complex. This was also a problem encountered by Østergaard et al. working with a more physical realistic simulation of the ATRON self-reconfigurable robot. In fact, for simple cluster flow locomotion 927 rules were necessary [13].

The use of hand-coded lattice automata rules becomes intractable as the complexity of the robot and the task increases. Hence, some effort has been made to automate

the design of lattice automata rules. Varshavskaya et al. [19] used a reinforcement learning based approach and Østergaard et al. [13] tried to use evolutionary algorithms. Both approaches were successful for relatively small tasks, but inherently suffer from scalability problems due to the size of the search space.

The lesson learned from this work is that lattice automata are useful if the configuration space is relatively small, but otherwise become impractical.

2.4 Hybrid Control

We have previously focused on locomotion through water-flow, however, in some cases it is not the function that is important, but the shape. Hence, transforming a self-reconfigurable robot into a specific shape becomes a task in its own right and is also important as a fundamental primitive that can be used as part of higher-level functionality. In self-reconfigurable robotics the transformation or self-reconfiguration problem is often stated as "Given an initial configuration and a goal configuration, find a sequence of module moves that will reconfigure the robot from the initial configuration to the goal configuration." [18]. Part of the effort to solve this problem has been to make global-to-local compilers. That is, compilers that take a high-level, centralised representation of a goal configuration and automatically compiles it into a lattice automata ruleset running distributedly on the modules.

Stoy et al. [16] use an approach where a three-dimensional CAD model is converted into lattice automata rules. We will describe this approach in detail below.

The first step is that a CAD model and a starting point contained inside this CAD model is given. At this point a cube is placed that represents a module. The algorithm then proceeds by adding neighbour cubes to this initial cube, but only if the position of the neighbour cube is also contained in the CAD model. Under the same conditions neighbour cubes are added to these cubes. From this point the algorithm continuous recursively until the contained volume has been filled. In effect, the surface-based representation of the CAD model is turned into a voxel-based representation. The position of each voxel corresponds to the desired position of a module in the goal configuration.

In the second step this voxel representation is turned into a set of lattice automata rules. The cubes are assigned a unique ID, in practice, an integer between 0 and N where N is the number of cubes contained in the representation. The second step is that for each pair of cubes with a shared face two lattice automata rules are generated. The direction orthogonal to the face determines the direction in which the two modules are connected. One rule creates one cube if the other cube is present and the other rule the other way around. Since they are symmetric let us just look at one rule. Let us assume we have two neighbour cubes with IDs $k1$ and $k2$ and the direction d is orthogonal to their shared face pointing from $k2$ to $k1$. If we then are in a situation where the cube with the ID $k1$ has been assigned we can then define that:

```
if (<module in direction d  equals k1 >) {
  <current module is assigned ID k2 >
}
```

We can now imagine a situation where a lattice automaton runs inside each cube with this rule set. If no IDs are assigned nothing happens. However, as soon as one module is given an ID the original assignment of IDs to cubes will be recreated.

The third step is to start a self-reconfigurable robot in a random configuration and give each module a copy of the above generated ruleset. We then trigger the self-reconfiguration process by assigning an ID to a random module. From this module IDs are assigned to neighbour modules and in turn to their neighbours and so on. Once this process has completed all modules either have an ID and belong to the goal configuration or they have no ID and are outside the goal configuration. The self-reconfiguration algorithm could now be made to work by letting modules without IDs move around randomly on the structure of module who has been assigned IDs. This would, given significant time, realize self-reconfiguration from the initial random configuration to the desired goal shape specified by the CAD model.

While acceptable in theory, the random walk of unassigned modules is not practically acceptable as the convergence to the desired configuration would take too long. In particular, when one considers that each self-reconfiguration step takes on the order of a couple of seconds. Hence, in order to speed up this process a separate algorithm is introduced that allow modules to travel to *growth points* more rapidly. A growth point in this context is a point where the neighbour module knows from the ruleset that a module should be present, but it is not. This implies that the configuration should be extended or grown in this direction. The intuition of the algorithm for attracting modules is fairly simple. A module, which is a neighbour of a growth point, sends out an integer. All modules listen for integers for a short period and propagate the smallest one they hear in this period plus one. This creates a gradient in the configuration. Now spare modules moving around in the structure can descend this gradient to locate the growth point. Once the growth point has been filled the initial module will stop transmitting and the gradient will adapt to create a gradient to a growth point further away or if no grow points are left slowly level out, because modules keep counting each other up until a maximum is reached.

Two important aspects of this approach is the division between the two control mechanisms. On one hand there are the lattice automata rules that handle the critical coordination element, e.g. ensure that the desired configuration is built, while on the other hand there is a simple algorithm for moving modules around on a global scale where the precise movement is less critical. This seems to be a critical aspect of applying lattice automata to self-reconfigurable robots because it is not tractable to encode rules for every configuration that the robot can assume. As a side point, careful configuration enumeration of the cube model has shown that for just 12 cubic modules there are more than 18 million different non-isomorphic configurations [17].

While the split into a local and a global control strategy is important, it does not have to be done as described above. Another approach explored by Rosa et al. [15] is to consider movement in the internal homogenous interior of the self-reconfigurable

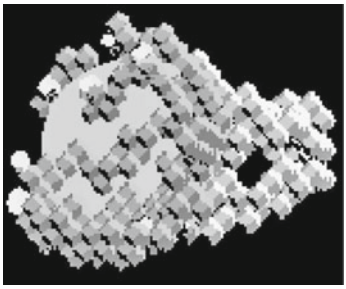

Fig. 2.4 This figure shows a self-reconfigurable robot reacting to and obstacle and grasping it. (Courtesy of Bojinov, © 2000 IEEE)

robot as part of a simple global coordination strategy while on the edges specific lattice automata rules were implemented to ensure that modules would not disconnect from the edge of the configuration which, if implemented in a physical system, would cause modules to fall down and potentially break.

Bojinov et al. [1, 2] also explore a combination of gradients for global coordination and lattice automata rules for local coordination. However, in contrast to the above work their rules are based on interaction with surrounding obstacles. For instance, if a module touches an object it can attract other modules to grow a structure around an object as shown in Fig. 2.4.

Considering this volume of work on lattice automata in the self-reconfigurable robotic community it is evident that lattice automata rarely can stand alone as a unifying control strategy for self-reconfigurable robots. However, it has been demonstrated that it is a powerful mechanism for controlling local aspects of a self-reconfiguration process that requires precise module movements.

2.5 Conclusion

There is a good match between the features of lattice automata and what is desired and possible to implement in self-reconfigurable robots. The rules of lattice automata typically only depend on the state of neighbor automata. On real hardware, this information can easily be obtained through neighbor-to-neighbor infrared communication. The distributed nature of lattice automata is also appealing from a self-reconfigurable robotic point of view since distributed control is a key element of making a self-reconfigurable robot robust.

There are, however, assumptions made in the use of lattice automata that are not easily handled in physical self-reconfigurable robots. One assumption is that unlike cells in a computer simulation, modules do not blink in and out of existence. In fact, in order to turn off one cell and turn on another a physical module has to move from the first to the latter. This is a physical process that easily can take on the order of several seconds. This immediately leads to a few problems e.g. the mutual exclusion problem of modules trying to move into the same cell, the problem of motion constraints

where not only the from and to cells are of interest, but also the cells that the module is physically moving through to get from one cell to another are also important. A final assumption, is that in computer simulations it is assumed that cells agree on directions, e.g. based on implementation in a two or three dimensional array. However, obtaining this information in a self-reconfigurable robots either requires dedicated sensors or a distributed consensus algorithm.

While the above problems have been less of a focus in the self-reconfigurable robot community, maybe because they are not apparent in simulation-studies, the problem of scalability has been a corner stone of the work. The main challenge arises from the combinatorial explosion of configuration neighbourhoods and the practical need to differentiate the active lattice automata rules in different parts of the robot.

As we have discussed in this chapter there are a few solutions to these problems. The first and most basic is to limit the task domain and, in addition, consider tasks where the individual module has a large degree of autonomy. In situations, where the modules start to depend on each other more careful considerations have to be done. Typically, this involves a split where local aspects of the control is performed by lattice automata, but there is a distributed algorithm that handles the global aspects of the problem.

From a lattice automata point of view it may be instructive to develop lattice automata where the underlying assumptions are a better fit for the physical modules of self-reconfigurable robots. From a self-reconfigurable robotic perspective it seems there is still potential in exploring hybrid algorithms with a lattice automata basis and a more globally oriented algorithm. From the point of view of mechatronics it may also be possible to develop modules where breaking the assumptions of lattice automata is less of problem. One avenue of research could be soft modular robots that may be able to squeeze past each other passing through a single cell alleviating the mutual exclusion problem.

In conclusion, the interaction between lattice automata research and research on self-reconfigurable robots has been productive and there is significant potential in exploring how the two can benefit each other further.

References

1. Bojinov, H., Casal, A., Hogg, T.: Emergent structures in modular self-reconfigurable robots. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 2, pp. 1734–1741. San Francisco, California (2000)
2. Bojinov, H., Casal, A., Hogg, T.: Multiagent control of self-reconfigurable robots. In: Proceedings of 4th International Conference on MultiAgent Systems, pp. 143–150. Boston, Massachusetts (2000)
3. Butler, Z., Kotay K., Rus, D., Tomita, K.: Cellular automata for decentralized control of self-reconfigurable robots. In Proceedings of IEEE International Conference on Robotics and Automation, Workshop on Modular Self-Reconfigurable Robots. Seoul, Korea (2001)
4. Butler, Z., Kotay, K., Rus, D., Tomita, K.: Generic de-centralized control for lattice-based self-reconfigurable robots. Int. J. Robot. Res. **23**(9), 919–937 (2004)

5. Butler, Zack, Rus, Daniela: Distributed planning and control for modular robots with unit-compressible modules. Int. J. Robot. Res. **22**(9), 699–715 (2003)
6. Christensen, D.J.: Experiments on fault-tolerant self-reconfiguration and emergent self-repair. Proceedings. Symposium on Artificial Life part of the IEEE Symposium Series on Computational Intelligence, pp. 355–361. Honolulu, Hawaii (2007)
7. Fukuda, T., Kawauchi, Y., Buss, M.: Self organizing robots based on cell structures—CEBOT. In: Proceedings of IEEE/RSJ International Workshop on Intelligent Robots and Systems, pp. 145–150 (1988)
8. Fukuda, T., Nakagawa, S.: Dynamically reconfigurable robotic system. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 3, pp. 1581–1586 (1988)
9. Fukuda, T., Ueyama, T.: Cellular Robotics and Micro Robotics Systems, vol. 10 of World Scientific Series in Robotics and Autonomous Systems, vol. 10. World Scientific (1994)
10. Murata, S., Kurokawa, H.: Self-Organizing Robots. Springer, New York (2012)
11. Murata, S., Kurokawa, H., Kokaji, S.: Self-assembling machine. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 441–448. San Diego, California (1994)
12. Østergaard, E.H., Lund, H.H.: Evolving control for modular robotic units. Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 886–892. Kobe, Japan (2003)
13. Østergaard, E.H., Lund, H.H.: Distributed cluster walk for the ATRON self-reconfigurable robot. In: Proceedings of 8th Conference on Intelligent Autonomous Systems, pp. 291–298. Amsterdam, Holland (2004)
14. Romanishin, J.W., Gilpin, K., Rus, D.: M-blocks: Momentum-driven, magnetic modular robots. Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 8288–4295. Tokyo, Japan (2013)
15. Rosa, M.D., Goldstein, S., Lee, P., Campbell, J., Pillai, P.: Scalable shape sculpting via hole motion: Motion plan- ning in lattice-constrained modular robots. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 1462–1468, Orlando (2006)
16. Støy, K.: Controlling self-reconfiguration using cellular automata and gradients. In: Proceedings of 8th International Conference on Intelligent Autonomous Systems, pp. 693–702. Amsterdam, The Netherlands (2004)
17. Stoy, K., Brandt, D.: Efficient enumeration of modular robot configurations and shapes. In: Proceedings of IEEE/RSJ International Conference on Robotics and Intelligent Systems, pp. 4296–4301. Tokyo (2013)
18. Stoy, K., Christensen, D.J., Brandt, D.: Self-Reconfigurable Robots: An Introduction. MIT Press (2010)
19. Varshavskaya, P., Kaelbling, L.P., Rus, D.: Automated design of adaptive controllers for modular robots using reinforcement learning. Int. J. Robot. Res. **27**(3–4), 505–526 (2008)
20. Walter, J., Welch, J., Amato, N.: Concurrent metamorphosis of hexagonal robot chains into simple connected configurations. IEEE Trans. Robot. Autom. **18**(6), 945–956 (2002)

http://www.springer.com/978-3-319-10923-7

Robots and Lattice Automata
Sirakoulis, G.; Adamatzky, A. (Eds.)
2015, XII, 313 p. 180 illus., 91 illus. in color., Hardcover
ISBN: 978-3-319-10923-7