Self-Organizing Systems: Cellular Automata

Report of the Advanced Seminar
Neuroscientific System Theory

submitted by

BSc Julia Müller

Neuroscientific System Theory
Technische Universität München

Univ.-Prof. Jörg Conradt

Supervisor:	MSc Cristian Axenie
Start:	15.04.2014
End:	27.06.2014

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
ORDINARIUS: UNIV.-PROF. DR.-ING./UNIV. TOKIO MARTIN BUSS

A D V A N C E D S E M I N A R

Self-Organizing Systems: Cellular Automata

Problem description:

Self-organization is ubiquitous in nature. Self-organizing systems are highly distributed, composed of large numbers of units that execute tasks with no direct control or guidance from an external supervisor. These systems exhibit complex global behaviors that emerge from the simple local interactions between their constituent units. In order to get a better understanding over the complex processes involved in the development of such systems a bottom-up analysis is needed. An interesting formal model of high interest are the cellular automata, capable to model biological pattern formation. Cellular automata are mathematical models for complex natural systems containing large number of identical components obeying local interactions. These systems can be seen also as discrete dynamical systems with simple construction but complex self-organizing behavior. Much as in the development of natural organisms, the localized construction process is a function of its surrounding structure and the description residing in every unit. The interactions, constructions principles and development into functional structures can me mapped into technical implementations which will be able to accommodate dynamical changes in their interactions with the environment. Furthermore, the cellular space will provide a representation of the environment such that spatio-temporal correlations can be learned and used in subsequent stages of self-creation and adaptation to accommodate changes in the environment. This advanced seminar aims to provide a survey of principles of self-organization in cellular automata, applicable to engineered systems (e.g. mobile robots) in order exhibit self-organizing behavior.

Tasks:

- Investigate the computational aspects of cellular automata.
- Investigate the self-reproducing capabilities of cellular automata.
- Investigate implementations of cellular automata in robotics.

Supervisor: Cristian Axenie

(J. Conradt)
Professor

Abstract

Cellular Automata are discrete dynamical systems, whose behaviour is specified by interactions between local cells. Hence there are some advantages compared to "traditional" computing. Cellular automata have a lot of advantages because of their simple units but yet their ability of accomplishing complex tasks. Those are the capability of self-assembling and self-repairing as examples. Features like them are convenient in a large area of subjects, especially in robotics there was great progress during the last years. For this reason, we focus on how simple rules determining each cells behaviour can lead to a complex overall behaviour.
This papers shows an overview over the historical development of cellular automata and we investigate in the principles of computation of cellular automata. Also a brief mathematical background will be provided. Next we will consider specific implementations which can be done using the advantages of cellular automata. After that we will focus on implementations of cellular automata in robotics, especially there are implementations shown, which are capable of self-reproducing cellular automata. At last there will be a final overview.

Contents

Chapter 1

Introduction and Motivation

In this first chapter we are considering the reasons why it is really useful to take advantages of cellular automata. Furthermore we will also show some history of the development of cellular automata starting by the ground-breaking idea of John von Neumann in 1947.

1.1 Motivation

For some applications there are no sufficient solutions at the moment. Many scientists mention the idea of collecting sun power in space using millions of nano-scaled mirrors focusing the light on some specific spots. Those robots can be built using so called cellular automata. So each cell will be able to find its best orientation by its own and does not need any supervision. Another quite useful application in space will be to have a robot which can explore other planets on its own. This robot hast to be able to adapt himself to the claims of these surroundings, which can be done by e.g. changing its shape. Also for this we can use cellular automata to create a robot, which is fully autonomous in sensing his surrounding, assembling itself, discovering and communicating.
But also in everyday life cellular automata can be of great use. Imagine a burning office building with hundreds of people. Due to the fire, some emergency exits are impassable. If there is just a static evacuation plan the impassable exits, but also exits where less people are running to are not taken in account. If we use some sensors communicating with each other, which are monitoring the way to those exits, there can be a dynamic evacuation model. Similar to an air plane there can be leading lights from each office individually to the nearest passable exit, controlled by those distributed monitoring sections.
To fulfil this claim, cellular automata models can be used. They have some big advantages. The most important advantage is, that each computing unit ("cell" in the following) runs just a simple program with local values as only input. A larger group of those simple cells however are able to perform complex tasks. These tasks are totally integrable in the less complex local program the cells are executing. How

this integration can be done will be explained in Section 2.4. Distributed computing is furthermore advantageous because of its possibility to perform all tasks in parallel, so that each cell will be updated at the same time. After all, as every cell runs the same algorithm and is also completely the same as all other cells, its easy to replace a cell if it is malfunctioning. Each defective cell can be replaced by an arbitrarily chosen other cell, running the same algorithm.
This leads to the next great feature cellular automata have: Whenever it is damaged, the system is in theory able to repair itself totally by just using working cells instead of the broken ones. How this can be reached in practice using robots will be the topic of Chapter 3. But we can go one step further: cellular automata systems are not only able to repair themselves, but also they can be able to assemble themselves. This specific feature is quite useful in many applications and will also be considered in Chapter 3.

1.2 Short History of Cellular Automata

With the ground-breaking idea of systems capable of self-reproduction, John von Neumann is the first one, who was thinking of systems like cellular automata in 1947. His idea was a unit cell containing three parts: A, B and C. Each part also contains a description of itself. While A is able to use an arbitrary description of a cell to build up the related cell, B is able to copy those descriptions. C is a control cell, telling A and B what to do. So for example C tells B to copy the description ΦX. Then C tells A to build up X by ΦX and then C splits $A + B + C$ from $X + \Phi X$. For this creating of a cell X each cell needs 29 values [NB66]. Now we choose $X = A + B + C$ and we know that X can be duplicated by $A + B + C = X$, hence we have the progress of self-duplicating. What sounds like this idea is simple copying DNA in formal language is really a new idea that time, considering that DNA was found in 1953.
We can see in the upper part of the graphic the tasks of each of the modules A,B, and C. The graphic below is showing the overall process: We insert a description ΦX as input of module B, which is duplicating the description. One of the descriptions is used by A to build up the module X out of it. At last C is combining X and the remaining description ΦX and split both from the complex $A+B+C$.

The next big step was done by Edgar Codd in 1968 as he designed a self-reproducing machine only using an eight-state cellular automata based on the model of von Neumann. As this model was not able to work because of four main issues, they were solved in 2010 as a proof of concept and the computer was implemented which can be found in [Hut10]. For this model there was a huge amount of cells necessary to build up a functioning model.
With Conway's game of life in 1974 cellular automata became much more known. In this "game" John Conway used cellular automata to simulate the evolution of a

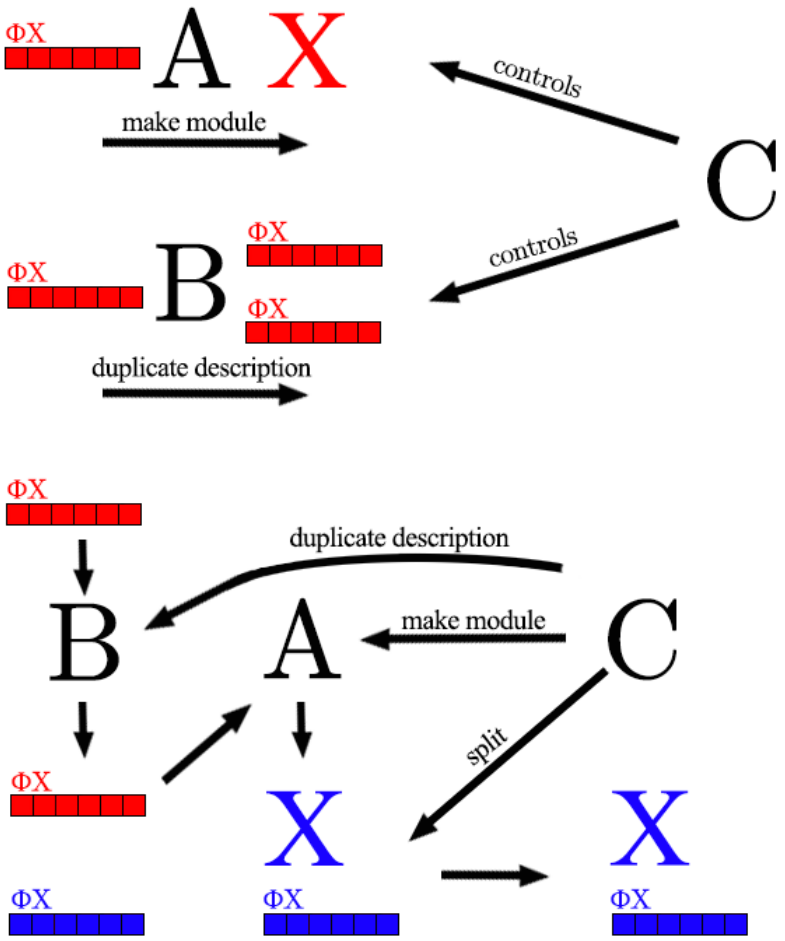


Figure 1.1: A sketch of von Neumann's idea of self reproducing cells. The upper part shows each modules tasks, the one graphic beyond shows the overall copying process

population over time. Here we see distributed computing: if a cell survives or not is only dependent on how many neighbours of this cell are alive. Hence each cell runs only a simple algorithm just collecting the value "dead" or "alive" from its neighbours and calculates the value of itself in the next time step only considering this values. In 2009 Philip Davies from Bournemouth and Poole College enhanced this 2D model to 3D [Dav09]. In Figure 1.2 we can see the behaviour of a population of cells over time. A cell has eight neighbours. If a cell has less than two neighbours, it dies because of lonelyness. If the cell has more than three neighbours, it dies because of overpopulation, if a cell has three neighbours, there are best conditions for a new cell to be birth.

In 1984, Christopher Langton created the so called Langton's loops by enhancing Codd's structure, while minimizing the number of total cells needed [Lan86]. Those loops are able to duplicate themselves by using the sequence of values in the loops. Some values were only for building up the wall of a loop, while the others are describing, what to do next. Even here we can see that this model operates with

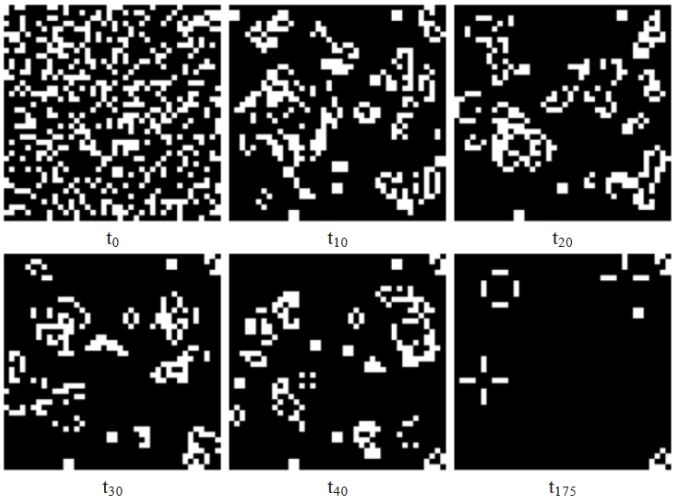
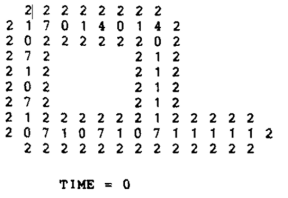
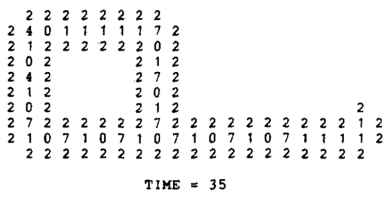


Figure 1.2: Conway's game of life: the behaviour of a cell population, 6 time steps, taken from [BT12]

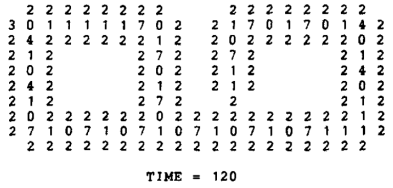
distributed computing, where each cell only needs information about its neighbourhood. In Figure 1.3 we can see, how the reproduction of a loop is done: From the initial loop, the sequence of ones start the reproduction, by breaking the wall out of 2-values. Each sequence has a different meaning, for example, while the "$7-0$" values extend the loop, two "$4-0$" signals create a left hand corner. For details to the sequence code, refer to [Lan84]. The cells are acting as the sequence of values tell them. So the loop is able to extend itself and finally split into to completely similar replicas of itself.



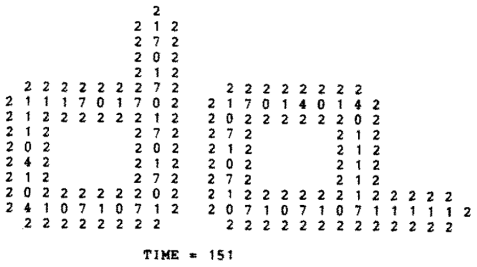
(a) Initial loop



(b) Start duplicating



(c) New loop visible



(d) Duplication finished

Figure 1.3: Langton's loops while self-reproducing, taken from [Lan84]

One of the most important scientists in this field is Stephen Wolfram who made a fully mathematical description of cellular automata and is well known for his publication about the basic one-dimensional classes of cellular automata [Wol84]. This is a cellular automata which evolves over time by all possible rules. Wolfram classified these rules and made several conclusions. We will take a closer look on Wolfram's work in Chapter 2.

Chapter 2

Introduction to Cellular Automata

After the brief overview over the history and advantages of cellular automata, we are now focusing on the formal description of cellular automata. This chapter therefore contains the rules cellular automata use and also the mathematical background of cellular automata computation.

2.1 Basic Operations

First of all, we can consider cellular automata as a way of building something out of small particles each the same shape. Let us first consider castles of sand. As we know, there are two main strategies to form a castle. The first one is quite easy: we add as many of the particles as we want to create a specific shape. This way of building up a complex structure by adding particles is used in cellular automata as a basic tool, too: these are the so called *construction rules*. But most of the time a castle of sand is not finished after only adding sand particles. To improve the castle we use the second strategy and remove some particles to gain a greater shape. Also this re-movement is used in cellular automata as *destruction rules*.

2.1.1 Construction Rules

Cellular automata are lattices containing cells. Let us start with a one-dimensional lattice with cells next to each other. Construction rules are the first basic operations a cellular automaton has to be able to perform. In our lattice should exactly be one initial black cell (Figure 2.1).

Figure 2.1: Simple one dimensional cellular automata, with initial one black cell

Each cell except the initial black cell, has in the first time step the value "white", let us say, this is equal 0, or in Conway's words, it would have been "dead". On the

other hand "black" equals "1", or is "alive". So every other cell than the initial cell is 0 first and thus has to be constructed, or has to be "born". Let us first consider the rules similar to Conway's game of life. Let's say a cell can be born if it has exactly one neighbour. This leads us to the following construction rules:

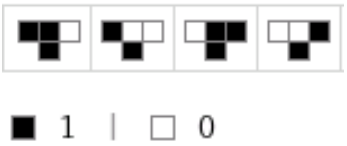


Figure 2.2: All possible construction rules for a more simplified game of life

The cells in the upper row shows the values of a cell an those of its *neighbourhood* of an arbitrary cell in the lattice. In one dimension the neighbourhood are the adjacent cells to the left and right of a cell in the lattice. The row below shows the next value of that cell, dependent on its own and the values of its neighbours the time step before.
By now we are at a point, where cells only can be born, but not die. This lead to the need for destruction rules.

2.1.2 Destruction Rules

Now we go on constructing our set of rules. As we know, our cell will die, if it has no or exactly two neighbours, there are these four additional rules: With these rules,

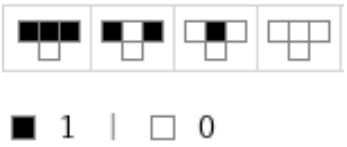


Figure 2.3: All destruction rules for a more simplified game of life

we have a complete set of rules, telling the cells in the one-dimensional lattice how they should evolve over time. If we now let the initial cell evolve over 20 time steps, we can see that the cells are always calculating its next value, by considering the values of its neighbours.
We have seen here one of the main features of cellular automata: to gain evolution, each cell in the lattice only needs to know about the values its neighbouring cells have. There is no need for a supervision over all cells.

2.1.3 Evolution of Cellular Automata

Now we can make a first conclusion over the evolution in cellular automata. We have construction rules on the one hand and destruction rules on the other hand.

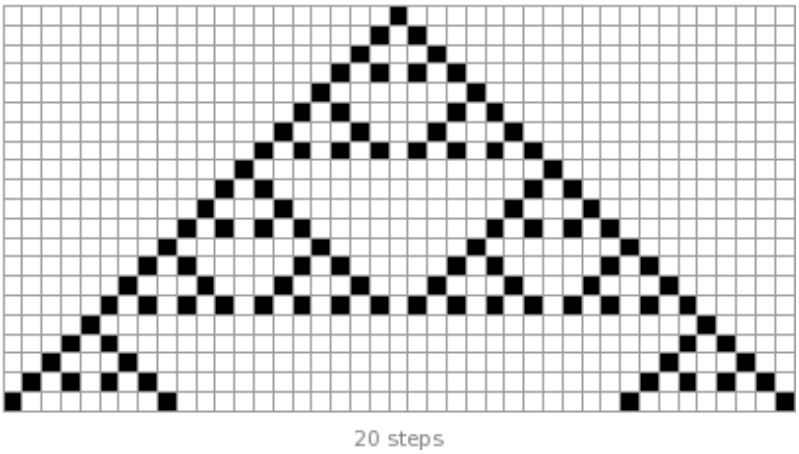
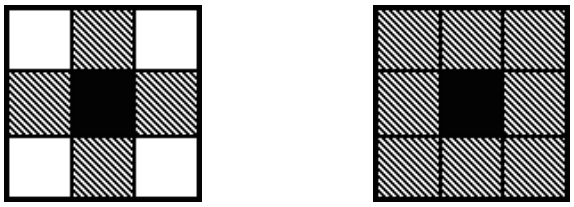


Figure 2.4: Evolving over time over the simplified game of life, taken from [Wol14]

A combination of them leads to the possibility of the cells in a lattice to change its values over time by only considering the values of its neighbours.
While in a simulation or in software, the construction or destruction really means creating or destroying parts of the system, we won't do this in robotics. But we use construction and destruction more as pieces of information if we use a cell for a complex system, built of many cells or not, therefore destruction means more disassembly than destroying. A good example for this are pebble robots, which are mentioned in Section 3.3.3. Those small robots are able to bond to each other, while each bonding is done by a "construction rule" or disassemble using "destruction rules". If there are redundant cells in the cell-complex, all connections to them will be pruned. The same will be done if cells are irrelevant for the cellular changes of states. This is considered in Section 2.4

2.2 Principles of Computation

We now have considered the special case of a one-dimensional lattice with only one starting cell, to explain the procedures of construction and destruction. But there are many more possibilities of using cellular automata.
Each cell contains k different values which are updated at each new time step. Both types of rules are implemented in a distributed fashion. This means that every cell has the same algorithm telling the cell how to update each of its values in the next time step. For this the algorithm implemented on each cell only uses the values of its neighbours as input.
The neighbourhood can be defined individually, but most of the time the von Neumann or the Moore neighbourhood are used. As can be seen in Figure 2.5 the von Neumann neighbourhood only contains those neighbours which has one edge to the cell. The Moore neighbourhood also includes those cells with one common vertex.



(a) (b)

Figure 2.5: von Neumann neighbourhood (a) and Moore neighbourhood (b)

For each cell we have the following values: n neighbours, s states with each v possible values. Also all neighbouring cells have those s states with each v possible values. Therefore there are (s^v) possible next states for each cell, dependent on a variety of $N = s^{v \cdot n}$ different possible neighbourhood configurations and thus s^N different rules how to compute the next step of one value of a cell in dependence on the neighbourhood. With v different states in a cell, we thus have a total of $R = s^{v \cdot N}$ different rules, which are possible for a cell to compute its next state-values.
The best way of understanding cellular automata was taken by Stephen Wolfram, who investigated in the so called elementary cellular automata [Wol84], [Wol98]. These are one-dimensional cellular automata with each cell having two neighbours and have just one state with two different values: black or white. The value is updated by each time step, considering the values of the two neighbours and the value of the cell itself. As each of these three cells can have two different colours there are $2 \cdot 2 \cdot 2 = 8$ possible binary states for the three neighbouring cells. Therefore we have $2^8 = 256$ possible rules, how to generate the next value of a cell depending on the neighbourhood. In Figure 2.6 we can see the set of rules, with the binary code 30. With this set of rules, each cell figures out, which state it will have in the next time step.

Wolfram tested all of these rules starting with an elementary cell, like shown in Figure 2.7 and from random initial conditions, shown in Figure 2.8.

By starting with an elementary cell, the development of the cells in the lattice is always the same. We only are able to see that in the different classes we have more or less periodic evolution of the cells in the lattice. Nevertheless it is difficult to differentiate between the classes. But if starting with a random condition, Wolfram could make out four different classes. In the first class, the cells will evolve in a single homogeneous state and their behaviour is thus totally predictable, we can see this in Figure 2.8a.

The second class (Figure 2.8b) is also predictable, because the cells finish in a specific pattern, which can either be a simple stable or a periodic pattern, dependent on the initial state. The behaviour of rules in class 2 is hence predictable from local initial states.

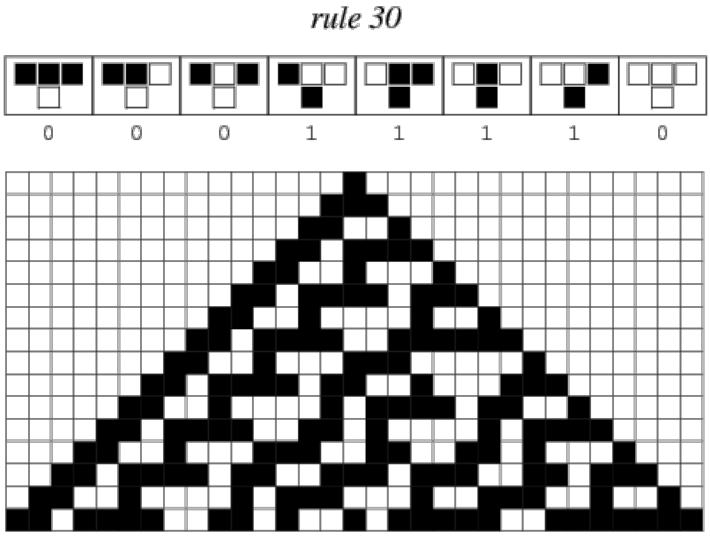
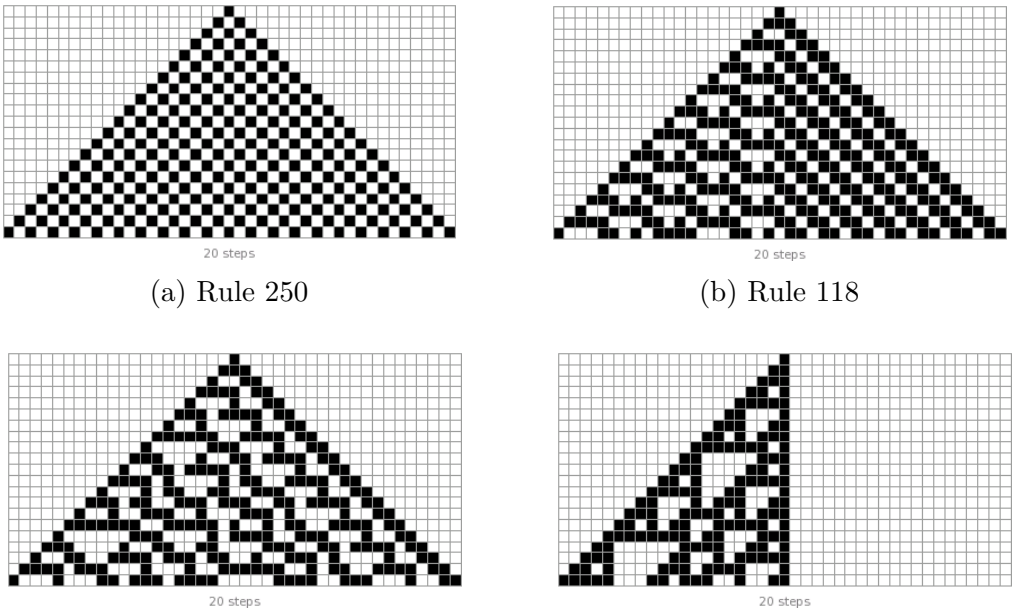


Figure 2.6: Elementary cellular automata with Rule 30, taken from [Wol14]



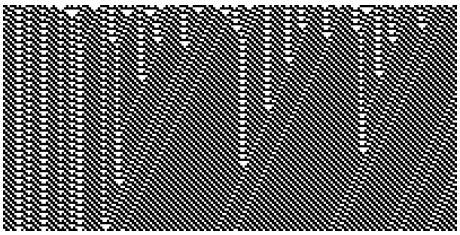
(a) Rule 250 (b) Rule 118

(c) Rule 118 (d) Rule 118

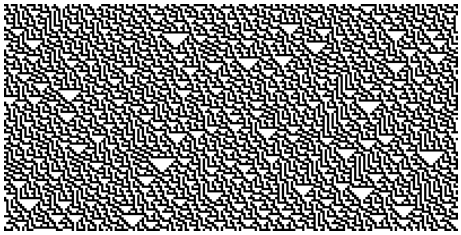
Figure 2.7: Cellular automata evolving from an elementary cell, taken from [Wol14]

Class 3 and 4 are much more interesting: as we can see in Figure 2.8c a class 3 rule leads to a chaotic and aperiodic pattern. These patterns are unpredictable and for each new initial state there will be another pattern evolving, which behaviour depends on an ever-increasing initial region. At last we have class 4 rules, like rule 118 in Figure 2.8d. These are the most interesting classes due to the fact that they are complex localized structures, sometimes also long-lived, their behaviour is effectively

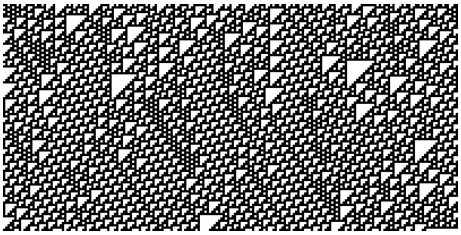
(a) Rule 250, class 1, steady state: limit points



(b) Rule 118, class 2, steady state: limit cycles



(c) Rule 118, class 3, steady state: chaotic attractors



(d) Rule 118, class 4, steady state: very long transients

Figure 2.8: Cellular automata evolving from random initial conditions, taken from [Wol14]

unpredictable and has to be simulated for each new initial state. Now an obvious question is, how to differentiate between class 3 and class 4 cellular automata. As we just take a look at them, they both seem simple chaotic. But we can make the following test: Change exactly one of the initial state values from black to whithe or vice versa. For class 3 cellular automata this leads to a change in values in the final pattern, for which the difference pattern looks like the elementary cell pattern. If we change exactly one of the initial state of a class 4 cellular automata, we are not able to make any predictions about the difference pattern. Hence the difference is totally unpredictable and has to be simulated.
Therefore we can conclude: While for classes 1,2 and 3 the dynamical systems theory is sufficient, for class 4 cellular automata computational theory is needed. This means, that as mentioned before, the evolution is computation itself in such a way, that we have universal computation here. Furthermore in class 4 cellular automata information processing is also possible.

Another interesting work done here is by Mitchell et al. [MCH94]. They are showing, that we can give cellular automata a specific task, in this case the cellular automata should find an algorithm which implements a (quasi-) period 3 function in a one-dimensional cellular automata. That means at every third time step there should be a periodic behaviour of the cells in the lattice, quasi means that not the overall lattice need to behave periodic, but only parts of it. This approach is reached by evolving cellular automata taking the word *evolution* quite serious. This means,

after each *generation* they evaluate the functions applied on the cells by their fitness completing the task of a period 3 function. Only the *Elite* functions E, e.g. the best 20% are copied to the next generation. The rest of the rules, will be crossovers of E. In this way they are evolving the fittest function over a total of G generations. Hence they are really copying the Darwinism into cellular automata.

This knowledge leads to the following: At the very first of creating a cellular automata we have to make several decisions. First we have to determine the neighbourhood. Are we using von Neumann or Moore or any other neighbourhood? Afterwards we have to take care, that each cell has the ability to communicate to its neighbours, to be able to gain the state-values of these cells. What is quite easy in simulating cellular automata will be an interesting issue in implementing cellular automata in robotics. At last each cell has to determine its next state-values by running the state-transition algorithm. This can be possible by implementing decision trees, but it is more simple, to calculate a state function with the neighbouring state-values as inputs. The transition from graphical decisions to a mathematical function is the content of the next Section.

2.3 Self-Reproduction

We have seen, that there are different ways of evolving cellular automata are possible. Now we investigate in a special way of evolving: the ability of self-reproduction.
For this reason, cellular automata are only able to do the same operations, which we considered in Section 2.1. Note that construction in the case here can be one for example: show a specific colour (see Figure 2.9 or bond to a specific or many specific neighbours. Vice versa this means for destruction rules: Show white or disassemble from one or all neighbours.
For this we need a way to tell the cellular automata how they should evolve to finish in a specific pattern. We only can tell the cellular automata at the beginning an algorithm, this algorithm is running for the whole time, thus we have to consider closely how to create this algorithm.
For self-reproduction we have a different approach than for self-assembling. Since the procedure of self-assembling is more simple, we consider this first. Self-assembling means, we have some loose cells, all the same, running the same code and we can not differentiate between them. Here the idea is, to implement a code on these cells, which is updating the states in the cells - this means, the bonding to the neighbouring cells or the color of the cell - in such a way, that after a finite number of time steps the pattern or complex cell structure converges to the requested pattern. An example for this can be seen in Figure 2.9. The way of finding the right transition rules will also be content of the following Section.
Let us now take a look at the main issues to be able to do self-reproduction. First the shape of the original copy hast to recognized in any way and translated into

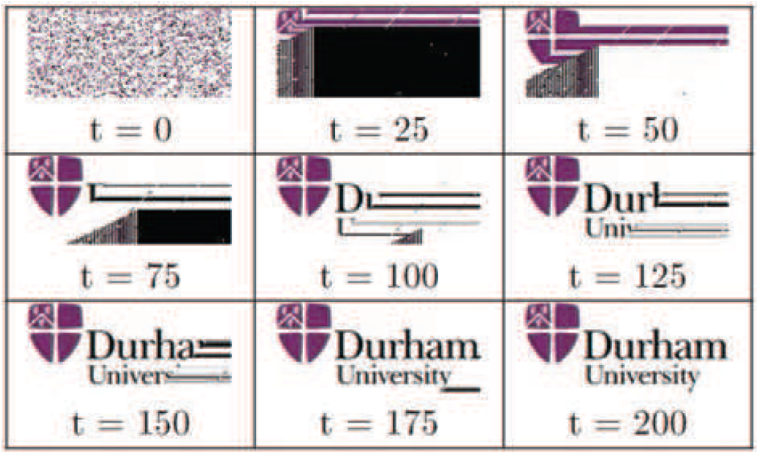


Figure 2.9: Self-assembling to a specific colour pattern, taken from [JMP11]

a description, then this description has to be communicated to other cells and last these cells should be able to rebuild the original copy. For the approaches of coloured cells and cell-complexes there are different solutions possible. As for robotics the reproduction of cell-complexes is much more interesting, we only consider this now. A possible way of doing this for cell-complexes is that we surround the original copy with a lattice of cells. One of these cells will act as a seed-cell. This one is copying its own state-values - that means, to which neighbours it bounds - and send it to an arbitrary other cell not part of the surrounding cells, also communicating, that this cell also is now a seed-cell. The new seed-cell now can bond to the same directions as the first seed-cell. Then each cell which is bonded to the original seed-cell is communicating one another its state values to the new bond cells on the new seed-cell. The order which cell should communicate next is e.g. clockwise. Since we have only a shape this is possible, so we just have to tell the cells which neighbour has the highest priority to be able to communicate and so on to the least priority neighbour. After each cell in the shape has communicated its state-values, there will be an exact replica of the first shape. This replica now has to tell each cells inside the shape to bond. After this, the shape itself is disassembling and the copy is done.

2.4 Mathematical Background

Cellular automata are evolving through state transitions, dependent on the state-values of their neighbours. Hence we have something like a graphical description, which we can look at and decide which values have to be updated and so on. Since we are human and are able to decide by "take a look and think about it", a computer cannot. So we need some formal description for the state transitions to be performed by each cell.

Graphical Description

The simplest way of formal description is, that we implement a decision tree. This means each cell checks each state-value of each neighbour one another and follows a tree. After the last state-value, the cell knows which state-values it will have in the next time step. As this procedure is neither easy to implement for large state numbers nor able to determine steady states in a trivial way, we have to abstract from this simple graphical way of computing and choose a more mathematical description.

Functional Description

As we know from signal processing in digital systems, we can transform these graphical rules into a lookup-table. There are several ways to derive a mathematical function from such a lookup table. Let us consider the rule 60 (Figure 2.10). We now

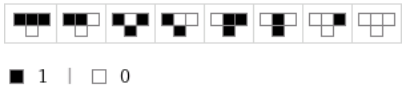


Figure 2.10: Rule 60

name n the next state of our cell. c is the centered cell itself, l is the left neighbour and r the right neighbour. As we can see, we have the following lookup table from

left	cell	right	new state
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Table 2.1: A lookup table for rule 60

this one, we are able to derive the logical function:

$$n = l \oplus c \tag{2.1}$$

which, transferred to a mathematical function is:

$$n = (l + c) \bmod 2 \tag{2.2}$$

With this formal mathematical description it is quite fast to implement an algorithm on each cell, which is calculating the successive state-values of a cell.

Matrix Representation

Whenever we have a mathematical description of a system of equations, we can transfer them into a matrix. Let us again consider rule 60 and the Equation 2.2. We get the Matrix

$$R_c = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \tag{2.3}$$

We now can use this matrix for a full state transition of cell c from time step t to $t+1$:

$$c_{t+1} = R \cdot \begin{pmatrix} l_t \\ c_t \\ r_t \end{pmatrix} \bmod 2 \tag{2.4}$$

Now we can go one step further: let us not only consider the next step of cell c but of all cells in the lattice. As we have a one-dimensional lattice, we name these cells now $c_1, c_2, \ldots, c_n$. Let us put all these cells in a matrix

$$C_t = \begin{pmatrix} c_{1,t} \\ c_{2,t} \\ \vdots \\ c_{n,t} \end{pmatrix} \tag{2.5}$$

For each cell the next state is computable using

$$c_{i,t+1} = R_i \cdot C_t. \tag{2.6}$$

With a matrix R_i which has the entries like in Equation 2.3 on the positions $i-1$,i and $i+1$ and 0 else. As we now know the $1 \times n$ matrices of each cell, we can combine them to a full matrix showing the transition rules for all cells.

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 1 & 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 & 1 \\ 0 & 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \tag{2.7}$$

Using this matrix, we can calculate the next state-values of each cell in the lattice easily with the following formula:

$$C_{t+1} = R \cdot C_t \tag{2.8}$$

State Representation

At this point we are able to transform our graphical rules to a matrix which we can use for figuring out the following state-values of each cell. This matrix is very useful

in determine the steady-state of the cellular automata using the *equivalent matrix model* [JMP09] and also [JMP11].
As we have seen

$$C_{t+1} = R \cdot C_t \tag{2.9}$$

and thus

$$C_{t+2} = R \cdot (R \cdot C_t) = R^2 \cdot C_t \tag{2.10}$$

or generally

$$C_{t+n} = R^n \cdot C_t. \tag{2.11}$$

If for some reason we have a constant offset such as

$$C_{t+1} = R \cdot C_t + D \tag{2.12}$$

in the transition function, with an arbitrary constant matrix D with $\dim R = \dim D$, then we have for n time steps:

$$C_{t+n} = R^n \cdot C_t + \frac{(\mathbb{1} - R^{n-1})}{\mathbb{1} - R} \cdot D \tag{2.13}$$

Equation 2.13 is using the geometric series equation, the full derivation can be found in [JMP09]. This equation fully determines the state-values the cellular automata have after n time steps.
What is left now, is to prove, that we can found a steady-state expression with Equation 2.13 for specific matrices R and D.
Generally convergence is possible under the following conditions: we must have given a sufficiently large n so that the cellular automata are able to converge. Additionally the final pattern C_n must be independent of the initial pattern, to make sure, that for every initial condition we converge to the same steady-state. For this reasons, R^n must equal 0, which is given, if R is an upper- or lower-diagonal matrix. Then we have:

$$C_n = (\frac{\mathbb{1}}{\mathbb{1} - R}) \cdot D \tag{2.14}$$

And therefore the final pattern will be the matrix D. As this explanation is kept short, the gentle reader will take a closer look on [JMP09].

Another possibility to make sure, that a cellular automata is converging into a specific final pattern is to use genetic algorithms for evolution. As we have covered one possibility by determine the steady state matrix, genetic algorithm is just mentioned here and not explained in detail.

Chapter 3

Cellular Automata in Robotics

After the introduction in cellular automata, their principles of computation and the mathematical background, we now can consider in which way cellular automata can be of great use in robotics. We investigate different approaches in a broad spectrum of applications. Especially we are focussing on the self-assembling and self-reproducing approaches in this Chapter.

3.1 Different Approaches

There are some formal approaches which can be useful for implementations. We have to remember that cellular automata are cells in a lattice, able to communicate only to their neighbours and updating their state-values by construction and destruction rules and thus are doing distributed computing (see Section 2.1). As we now consider the different approaches we always have to keep this basics in mind, to understand the advantages or disadvantages of an approach.

Build mathematical Function

Cellular automata are able to form a mathematical function like for example x^4. For this reason the cells have to be organized, so that they know, which cell is representing which value on the x-axis. This can be done by telling one cell it will be $X = 0$, we can call this cell *seed cell*. So each cell is able to communicate with its neighbours and figure out how far from the seed cell it is and thus, which value it should represent.

Evaluate Surrounding

Another task cellular automata can attend to is the evaluation of their surroundings. In this case, ingthe cellular automata work as a distributed sensor, compiling a lot of data and evaluate these. The principle is here, that every cell is sensing a specific value, for example an obstacle. As there are a huge amount of possible applications

starting with path planning and ending with micro structured sensor arrays, we consider the following thought experiment to get a concrete example:
Let us think about the case of emergency from the motivational part in Chapter 1. In such a case, a large office building has to be evacuated. Let us say we have one sensor in each passage. This sensor is able to detect fire, humans and so on. All sensors are cells in a cellular automata. The cells are ordered in such a way, that they are *bond* if the passage from the one and the passage from the other cell can are two successive parts of one possible emergency exit. If a cell detects fire or too many humans in its supervised passage, it communicates to its neighbours, that the passage is closed and thus *disconnects*. If this means one of the neighbours now can not provide a passable way to an emergency exit any more. It also *disconnects* from its neighbours and so on. As humans are not able to recognize the bondings, each sensor is coupled with green or red light. So it can visualize if it is an emergency exit.
We can summarize the parts needed for the evaluation of a surrounding: first we need a way for the cellular automata to communicate each other, second, they need a specific way of transforming the sensed data into their state-values. Third they need an algorithm, telling them, what to do with the state-values of its neighbours. At last they need some kind of output. All these factors will be considered in the following sections.

Motion

Until now we have considered cells which are more ore less static in a lattice or at least static in the same neighbourhood. We have to break this up for movement of the cells. For this reason the cells have to be able to move on their own, independently from their initial neighbours.
There are several approaches how this can be implemented. One of them is to build up a specific shape with the cells, which is able to rotating around one or two axes to move forward [JOL04]. This has the disadvantage, that such a construction is only able to move in flat surroundings, steps will be a serious problem for this construction. Another really interesting idea is locomotion. This means, the complex of cells should flow over obstacles, like water [BKRT01]. This can be done by rotating all cells clockwise around each other by specific rules. With this they will be able to climb at least as high as long the cells will be if we put them all in a row.

Self-Assembly

At last we consider the most interesting approach we can get with cellular automata: the task of self-assembling and also in special cases self-reconfiguring. As we have seen in Section 2.4 we have all the basic knowledge how this is working. Now it is interesting, how this approach can be reached with cellular automata. The idea is that we have a lot of small cells. Each cell has to be able to move or to use some way to get in the right place. For this the cells have to be autonomous systems

themselves. If the cells are something like sand or pebbles, we just have to tell those cells, what they should look like. If we want to do this once, for example we tell those cells "create a chair", we just have to implement the correct algorithm with "chair" as final pattern and we are done. But for this application we do not really need cellular automata. For this we just can buy a chair. Now imagine we are in a camping van out in the wild and are not able to take all furnishings with us. But we have a great amount of cellular automata cell. It would be nice, if we tell them "build a chair" and they will do. Wouldn't it be perfect if we tell them later "now build a bed"? For this, the cells only need to know the algorithm for "chair" and the one for "bed" and we have to tell them which one they should use. Here we are using the feature of self-assembling. But let us go one step further: After one week with the van, one wheel is broken down. We think of our cellular automata and tell them "build a wheel". But there is no algorithm for "wheel" implemented yet. So another function will be interesting: copying one of the functional wheels. This means our cellular automata are able to self-reconfigure any other object.
This excursion should give us a feeling, why the features of self-assembling and self-reconfiguring are quite interesting and helpful. As we know yet, self-assembling is possible, whenever we calculate a matrix, so that the cellular automata is converging to a static pattern. We also know, how the principle of self-reconfiguring works: surround the object, send the state-values, build a copy of the surrounding and fill this. We will see in the following sections that it is right now possible to implement these abilities in robotics and we also show the different ways more clearly.

3.2 Sample Implementations

In the last section we had an overview over applications which can be done using cellular automata. Hereupon we are focusing on our main interest: the implementation of cellular automata in the field of robotics. For this we are investigating different implementations. As before, we start with the simplest approach of implementing a mathematical function. Then we go further on implementations of sensors and also of those which are enabling mobility.

Mathematical Function

A mathematical function will be sent to an arbitrary cell. This one is now the seed cell c_{seed} and thus the starting point for the propagation of the function. All cells consider themselves as a function-relative position. Only the position of the seed cell is already part of the final function pattern.

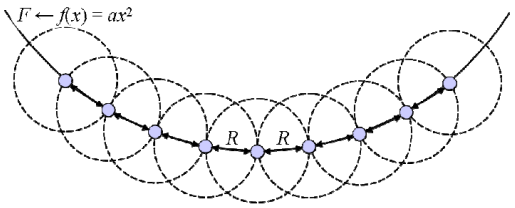


Figure 3.1: Calculated vectors lead to cellular automata evolving to a parabolic function, taken from [MW08]

All other cells are calculating dependent on the position of their neighbours their function-relative position as well as the vector pointing to this position. The final positions are calculated in such a way, that the distances between two neighbours are always the same, while the mathematical function is build by the cells, this can be seen in Figure 3.1. At last the cells move or evaluate to these positions.

Sensor Array

In the next approach we can see how cellular automata can be used as sensors. In their work Hackwood et al. [HW88] considered monitoring of aircrafts using cellular automata. For this they build up sensing arrays out of cells in the form of a ring (Figure 3.2). The cells have all a specific type of sensor integrated, this can be a camera or a sonar sensor. All cells are arranged in a ring, while half of them can point towards the center of the ring and the other half can only point outwards of the ring. The goal is now, that the cells arrange themselves in such a way, that if there are objects approaching, they point in this direction, which means they monitor this area.

We can see in Figure 3.2 (a) aircrafts approaching from different directions at the same time. To monitor all of them the cells arrange themselves alternating, so that all aircrafts can be monitored. This leads to a huge area monitored, but with low resolution. In case (b) there is an aircraft approaching only from one specific direction. Here the cells are arranging themselves in such a way, that all are pointing to the aircraft and can monitor its behaviour in more detail. This high resolution goes with the decrease of the area monitored.

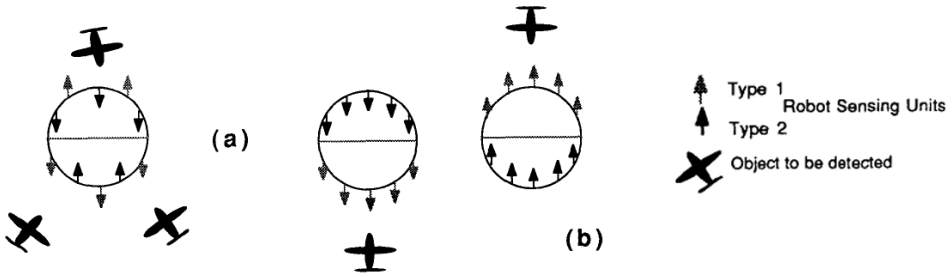


Figure 3.2: Array of sensors for detecting aircrafts, taken from [HW88]

At last we have to know, what cells should do, when one of their neighbours breaks down and they loose contact to this neighbour. As all cells are locally fixed sensor units, they cannot move, but they can ignore the broken cell and the neighbours of this damaged unit become neighbours now. This makes the system robust. But nevertheless it would be good to have something like a movement in the cells, so that the ring will not have holes, but all cells move a bit away from their neighbours so that all directions will be covered again. Since motion is a possible action in cellular automata, we consider a possible implementation next.

Locomotion

One possibility of movement in cellular automata is found by copying the nature. We consider the cells as drops of water, which is like the principle mentioned in Section 3.2. Like water the cells should be able to *flow* over obstacles. Butler et al. set up some rules for a movement to the right, which can be seen in Figure 3.3.
We can see, that the rules tell the cells to move to the upper right space in the lattice whenever it is possible. If the cell is the top cell, it moves to right and downwards. As we also see from the rules, the movement differentiates between free cells and other cellular automata cells and obstacles. This leads to cells moving like a wheel around an imaginary axis, while the axis is travelling rightwards.
As expected, we can see in Figure 3.4 that the cells are flowing around an arbitrary object. We can see here, that the neighbourhood is not static any more, but it is mandatory, that each cell has at least one neighbour, not to get lost from the

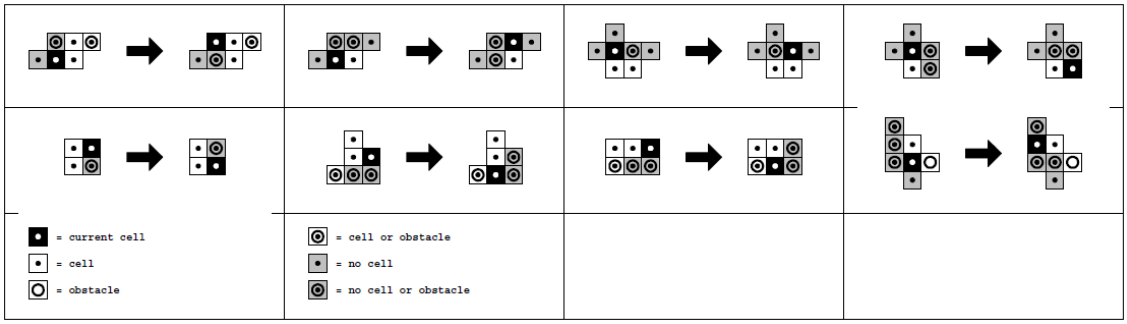


Figure 3.3: Rules which provide a locomotion of the cells to the right side, taken from [BKRT01]

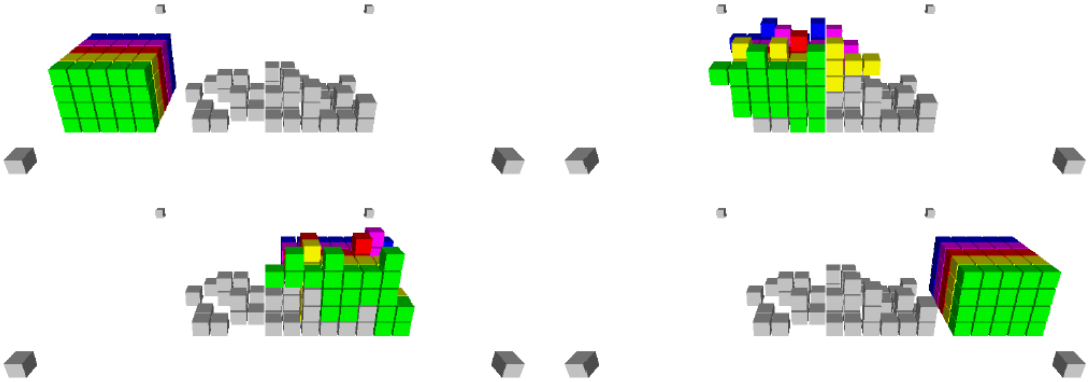


Figure 3.4: Four snapshots while the rules in Figure 3.3 were simulated, taken from [BKRT01]

cell-complex. If a cell breaks down here, it will be losing contact to the other cells and will be treated like an obstacle, therefore the system is also robust.

Real-Time Path Planning

Another approach combining sensing and motion is real time path planning. One possibility which we want to consider here is based on parent-child relationships between two cells in a Moore neighbourhood, which is explained in detail in [AAK12]. Each cell has for this complex task six values with different states, all of them to be updated each time step.

First we have to mention, that for all obstacles and also for the robot the Minkowski Sum is calculated, so that the dimensions of these objects were figured out and we can consider all parts now as points or cells in a lattice. Thus all obstacles and also the robot are occupied cells in the lattice. We name the start cell q_{init} and the goal cell q_{goal}.

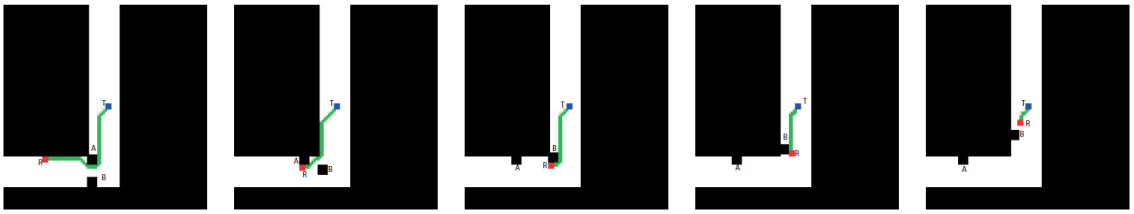


Figure 3.5: Real time path planning with two moving obstacles, taken from [AAK12]

Each cell has six values:

S_{state}	for the current state of the cell, this means if the cell is on (1) or off (0).
S_{cf}	is the child flag, it is 1 if the cell is currently a child.
S_{pf}	is the parent flag, it is 1 if the cell is a parent.
t_{on}	is saving the time step at which the cell transitioned to an active state.
$P_{i,j}$	is the address of the parent cell and
Φ_s	maintains the accumulative cost from the goal to this cell.

Now we need some rules, telling the cells how these state-values have to be updated. To calculate the states of these values, there are five rules implemented:

Rule 1: A cell is only activated if one of its eight neighbours of the Moore neighbourhood is activated in an earlier time step. Otherwise the cell remains deactivated.

Rule 2: A cell becomes a child, if one neighbour of that cell is already active with its $t_{on} < t$.

Rule 3: A cell becomes a parent, only if it is active, already a child and its $t_{on} < t$.

Rule 4: A cell only becomes a child of the neighbour with the lowest accumulative cost function Φ_s.

Rule 5: A cell saves the address of its parent cell, calculated in Rule 4.

This rules lead to a complete algorithm, which will find a way from the start to the goal, whenever a way is possible. If there is no way possible, the cells won't be activated and the process will terminate. If there is a way possible, it will be the shortest way, due to Rule 4.

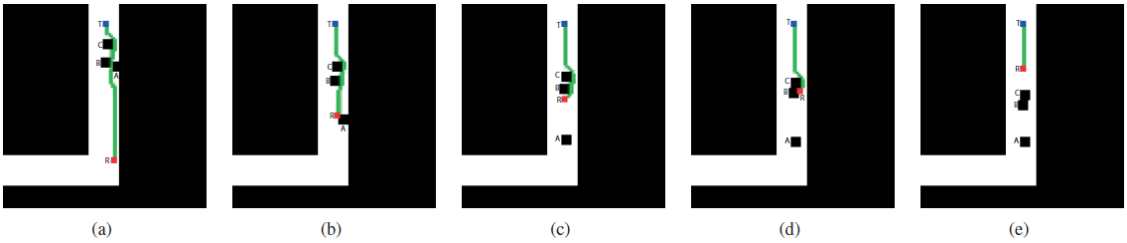


Figure 3.6: Real time path planning with three moving obstacles, taken from [AAK12]

Thus we can summarize: With the possibility of cells only communicating to their neighbours, we are able to update the possible way during the path finding and robot movement at every time. This is also true for an arbitrary number of moving obstacles occupying the space between the robot and the goal, see Figures 3.5 and 3.6. At every time the cells are always finding the shortest path, if there is a path existing from the robot to the goal. This only takes place by using simple construction or destruction rules, as we mentioned them in Section 2.1.

3.3 Self-Assembly and Self-Reproduction

In this section we begin to focus on the goal of implementing self-reproducing and self-assembling robots. There are four different approaches in robotics, which are capable of self-assembling or self-repairing. We show them in this section, emphasizing the similarities and variations between them.

3.3.1 ATRON

ATRON is a fully implemented cellular automata in robotics. The robots created for this use are nearly ball-shaped, with about 11cm in diameter. We can see some of them in Figure 3.7. The robot is built of two hemispheres with a connecting axis, so that the hemispheres are able to rotate against each other. All robots are placed in a lattice in such a way, that the axes of the robot are in parallel with the x-, y- and z-axis of the lattice. On each hemisphere are four connectors, those are alternating male and female connectors. Each robot is able to have up to eight neighbours. They communicate via infrared signals, which can be received at optical sensors. With them each cell send its state-values to its connected neighbours. Furthermore each robot is able to sense the position of its hemispheres.

As the ATRON have only the rotating axis as degree of freedom, they are quite limited in their mobility. To be able to change shape, there are so called meta-modules used. Meta-modules emerge from an unstructured group of modules and are not po-

Figure 3.7: Some ATRON modules connected together, taken from [JOL04]

sitioned or move in a permanent meta-lattice. Then this new meta-module is doing some meta-tasks and after finishing this tasks, it become passive again. Meta-modules can consist one ore more modules, in this paper we only consider the case of a single module meta-module. The gentle reader can find information about five other meta-module cases in [COL04]. A single meta-module is able to perform 16 different meta-tasks. All of this tasks have the same blueprint. A meta-module first disconnects from all neighbours except one and tells this one to move the meta-module by rotating his axis. For this action, this neighbour of the meta-module has to disconnect all other neighbours except the meta-module. After the neighbour moved the meta-module, they disconnect and the neighbour returns to its state before the movement. In Figure 3.8 we can see at which positions a single meta-module can be moved in one action.

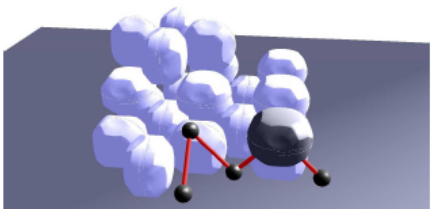


Figure 3.8: Reachable space of a single meta-module, taken from [COL04]

Now we have autonomous homogeneous robots, which represent the cells of the cellular automata. We have explained how they are able to connect to each other and also how movement is implemented. This leads to a system, which is able to do self-reconfiguring. At last we have to focus on the rules, which are implemented on these robots. The rule set is a simple point to point set here.Point to point means the rule maps from one point in the configuration space of the modules to one point in the action space. It can also be enhanced to a more generalized rules set, but

for explaining the working principle it is sufficient to explain the point to point rules.

Figure 3.9: Some ATRON modules connected together, taken from [ØKBL06]

Therefore a huge amount of rules is used to describe the overall behaviour of the ATRON modules. For defining the rules, there is only a concrete definition of the initial module configuration needed. To be able to apply the rules, the initial configuration of the modules have the one given to the point to point rules, otherwise they won't match. All rules are designed by considering the initial module configuration and create an action for each module and transferring this into a rule. Hence we easily can calculate the rules using a simulation and implement a specific final pattern in the point to point rules.
As this approach is focussing more on the technical details of the ATRON robots and the rules are quite simple, we will consider other approaches also and compare them to each other.

3.3.2 Claytronics

Claytronics also named Catoms are small atom-like robots, by 2005 there were cylindrical prototypes existing with only 44mm in diameter. At the end of the progress of inventing the Catoms, they should have a size smaller than one millimetre.

The Catoms should work like *voxels*, which means nothing else, than they should be pixels in a 3-dimensional shape, also able to change colours like pixels do. Additionally these voxels should be able to assemble and colour themselves in a 3-dimensional shape. This is gained by an initial flat structure and the application of external forces. The Catoms are activated by these forces like pixels are activated on a screen. Also the external forces lead to motion of the Catoms, which are not capable of locomotion on their own. Like in the ATRON approach, each Catom is running an algorithm on it. A Catom should be able to localize its current position in relation to other Catoms. With this information each Catom will be able to start moving on its own and change its alignment.
Different to the former approach the Catoms are not programmed as a single cell, but as an ensemble. This program for the ensemble-level has to be pushed to a low-level program which is implemented on the Catoms. As traditional imperative programming languages are not really useful for this approach, the creators found two new programming languages: *Meld* and *LDP* to fasten the programming of

(a) Plain Catom, with no moving parts

(b) Cube Catom with electrostatic latches

Figure 3.10: Two different Catom shapes Carnegie-Mellon University and Intel investigate in, taken from [Uni14]

each Catom, because the ensemble can be determined by the programmer and the languages automatically compile this description down to the cell-level. Both languages are declarative and its programs are about 20 times shorter than equivalent imperative programs which is quite useful for implementing them on Catoms.

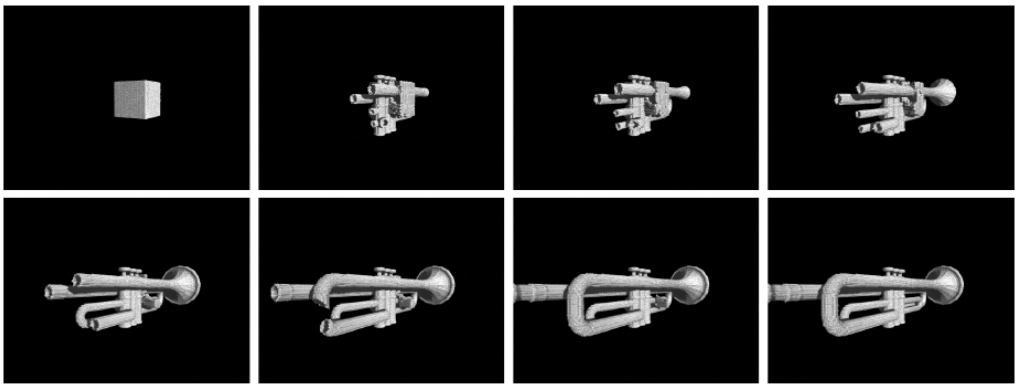


Figure 3.11: Self-assembling of Catoms to a trumpet shape, taken from [Uni14]

Because the programming takes place on the ensemble-level all the activity of the Catoms revolves around this level. Therefore the cells on their own are not capable of functioning on their own, it always has to be at least a group of cells to work together for essential functions, for example the group is placed on a powered surface and the power is distributed on the single cells.
The functionality of a Catom depends on its position in the ensemble. For this the cells establish a coordinate system which is used in the whole ensemble. To establish the coordinate system the cells have to sense where their neighbours are relative to their own distributed coordination system. This takes place like a cellular automata

approach only using information about each cell's neighbourhood. When the global coordinate system is determinated, the system is ready for self-assembling.
The first step of self-assembling is that the desired shape is sent to the ensemble and distributed to all Catoms. Next each cell executes a small distributed program, which leads to the construction or destruction of so called *meta-modules* to reach the local goal shape. As overall process the ensemble reaches the desired shape. Summed up, we have a top-down implementation of self-assembling cells. This is the biggest difference to the former shown ATRON approach. But as we see in Figure 3.11 the self-assembling is also possible for this approach.

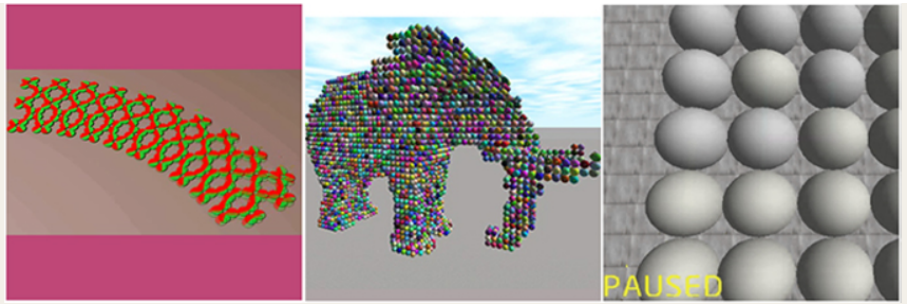


Figure 3.12: Different simulated applications of Catoms, taken from [Uni14]

A great idea for using this Catoms is *Pario*. Pario should be the next generation of communication. After we now are not only able to send our voice, but also a video of ourselves while communicating with friends far away, Pario will go one step further: The voxels should allow physical interaction with the person we talk to. This means the person will be rebuilt by the Catoms which are able to change their shape and colour to copy what the person does at that moment in real life. We can see in Figure 3.12 in the picture in the middle a coloured elephant. Like this elephant our person we talk to will be copied by the coloured self-assembling voxels and will be able to also copy the movements of the person.

Another great use of the self-reproduction capability of Catoms is that they enable a functionality for something like a 3-dimensional fax: we insert in our fax machine an object, sense it in three dimensions and send its description. Then we can take the object away from our machine. At the destination the Catoms will be able to reconstruct the shape of the object and its colour themselves, how this will be working is shown in Figure 3.13. Here we can see a huge amount of Catoms bonding together. To reconstruct the shape sent, all redundant bonded cells will unbind. They call this process of disassembling *digital sand casting*. The process here is like the one of self-assembling: first each Catom localizes its position and then figures out of the description of the shape if it is part of the shape or not. If not it will unbind, otherwise this Catom will stay connected to the ensemble. In this way the user just have to pick the ensemble out of the mass of unbound Catoms and gets a

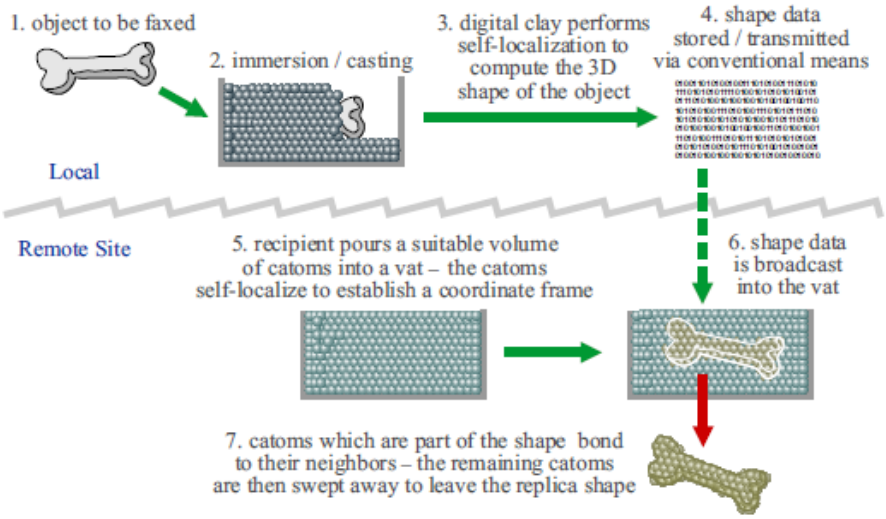


Figure 3.13: Overview over a 3D fax machine using Catoms, taken from [PCK+06]

perfect replica of the sent item.

3.3.3 Pebble Robots

Now we consinder one of the most interesting approaches in implementing cellular automata as self-assembling robots. With this pebbles we can build up any specific shape we want them to. They also have the ability to self-reconstruct other shapes, as long as we hand these shapes over to the robot pebbles.

As we can see in Figure 3.14 the small robots are all the same, starting with the same shape, same hardware and also the same algorithm running on it. Therefore they are not distinguishable for any supervisor. The idea here is, that we use this small cubic robots to create complex shapes with them, just like they were pebbles. The goal of this research group is, to create *smart sand* at the end, which means sand-size robot grains, which are able to rebuild every shape we want it to.
Let us first consider an individual cell, or robot pebble as they are called also. A cell has a cubic shape with an edge length of 12 millimetres. Each cell is autonomously able to actuate, sense, communicate and control. All of this abilities are only usable by a cell in interaction with its von Neumann neighbourhood. Each cell runs the same algorithm, like we have seen in Section 2.4 there are algorithms which lead to a steady state pattern. A self assembling process with pebble robots is quite easy: First we have to communicate a construction plan to each cell, this means we implement a transition matrix with the desired steady state. Then we put all the cells in a bag or some box. There all cells are bonding together. While running the algorithm on each cell, the unnecessary cells will be removed and the desired

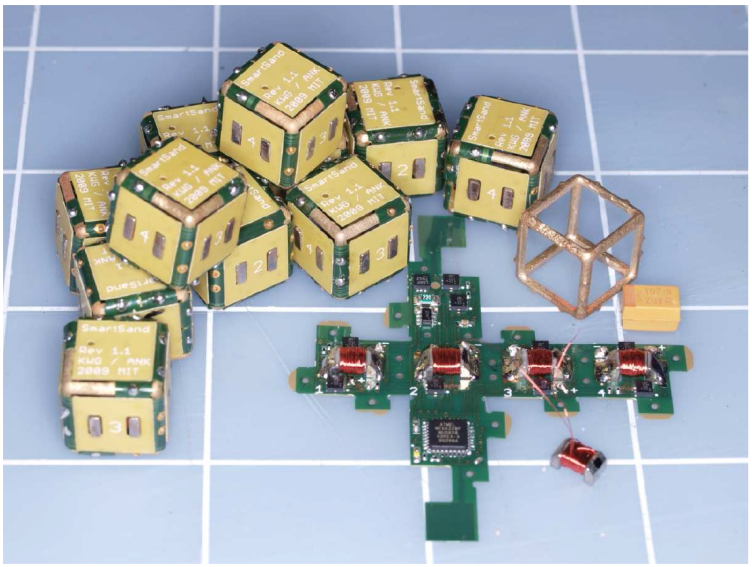


Figure 3.14: Each pebble is only 12mm per side and is running the completely same algorithm as all other pebbles, hence they are completely indistinguishable, taken from [GR10]

shape will be left. This way of building up a specific shape is much faster than using construction rules and also needs less energy.

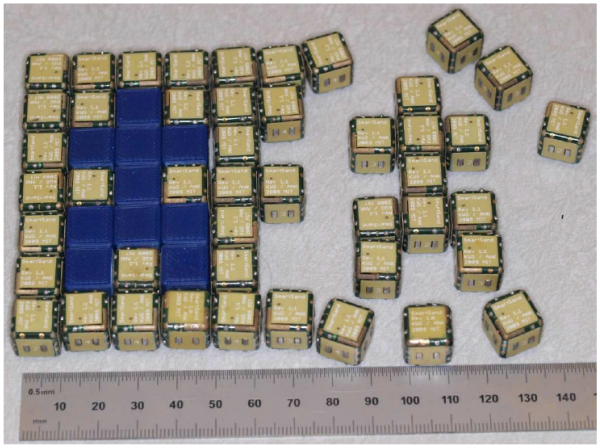


Figure 3.15: Pebble robots reproducing an arbitrary shape, taken from [GR12]

Now we will consider the self-reproduction with pebble robots. We have to sense the original first, before copying it, this is shown in Figure 3.15. Here we see an arbitrary blue shape inserted in a lattice of cells. As the cells are only macroscopic by now, we only can insert shapes, which edge lengths are the same as the edge lengths of the pebble robots. On the right side of the Figure, we see the perfect

replica of the initial blue shape. We have to mention here, that this picture is not the result of an implementation, because the pebble robots are by now not able to move on their own. But as we will see, the programs are set and in simulations the self-reproduction is working already.

Let us consider the idea behind self-reproduction with pebble robots. We can see in Figure 3.16 how the algorithm is working:

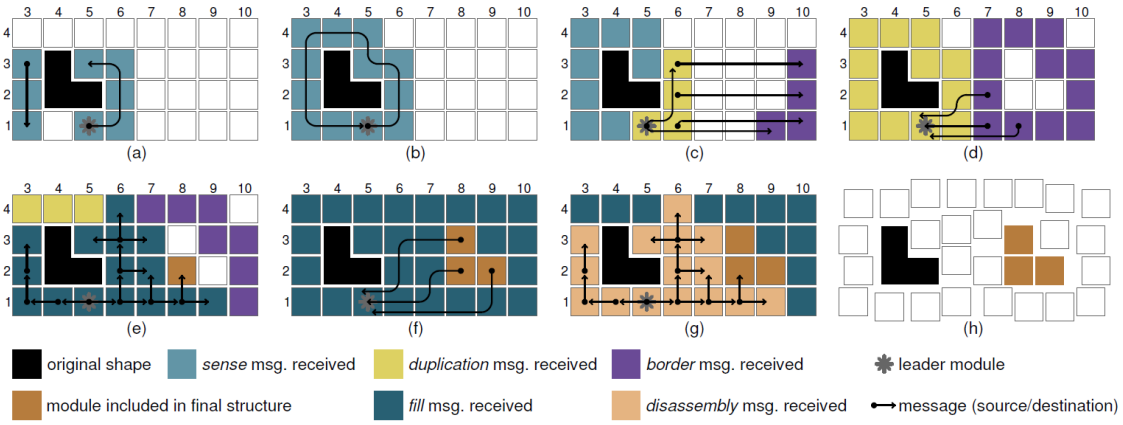


Figure 3.16: Principle of sensing a shape, communicating the information to neighbourig cells and replicating the shape, taken from [GR12]

For the duplication of an object we need the following steps:
First we need a *encapsulating and localization* algorithm. For this the user has to surround the object with pebble robots. In the future when the pebble robots will be minimized to sand-scale robots, the object can be buried under them. When the object is fully surrounded by robot cells, the user sends a start command to a single cell and this is starting the encapsulating and localization likewise. The cell which is receiving this command is now a seed-cell at the position $(0|0)$ and tells its neighbours their coordinates in the lattice. The moment a cell receives its coordinates it also bonds to the cell complex containing the seed-cell. Then this cell also tells its neighbours about their coordinates and so on.
Next each cell enters *shape sensing and leader election* phase. Here the goal is on the one hand to determine the perimeter and area of the original obstacle and on the other hand to elect a leader module located on the perimeter of the original copy. At the beginning of this phase each module corresponds to its neighbours, if there is no response the cell assumes that on this position is the object to be duplicated. Using the bug algorithm a module attempts to route, as the object is not able to response, the message will run around the entire perimeter of the object. The message we send has the functionality that it senses the way it is running. If the message now returns to the sender, it contains the whole perimeter and area of the object. For detecting a leader, each cell discard incoming messages which are sent from cells

with lower IDs, than themselves. Hence all messages except one will be discarded and the cell which receives its own message back is the only leader now.
Now we are in the step of *border notification*. Here the boarder of the original copy is sent to nearby cells and duplicated. For this we need three types of messages: *duplication messages* tell the cells on the perimeter of the original object, that they are building up the border of the object. Then these cells are sending *border messages* to nearby cells in the distant of a specific vector, informing them, that they will be the boarder of the duplicated object. These cells answer with *confirmation messages* so that the leader can determine when we have a complete border of the replica.
Now that we have a duplicate of the border, we inform all cells inside the border, to connect to each other, they also get the information to stay connected, even if all cells should disassemble. This phase is the *shape fill* phase.
Since we got a connected copy of the object inside the border, there is no use for the two borders to stay connected any more, and the cells are told to *self-disassemble*, while the cells inside the border should stay connected. The user now has to remove all single cells and get the exact replica of the original object.

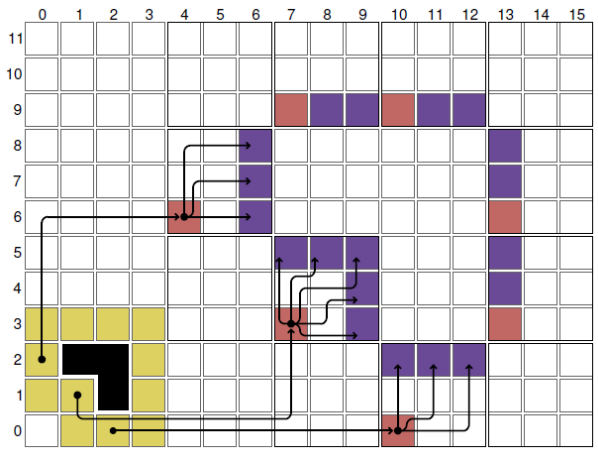


Figure 3.17: Using the principle of replicating a shape to be able to scale the shape before assemble it, taken from [GR12]

As we can implement different algorithms on the cells, we are also able to tell them to make more than one copy or also scale the object. This is shown in Figure 3.17. Now each message sent from a boarder cell has to contain a additional factor depending on the magnifying factor. With this border messages the red cells are found. These cells now are local leaders determining the new border. The gentle reader can find an exact explanation for scaling and multiplying the object in [GR12].

We can summarize that we are using only cellular automata rules in this approach. Like the ATRON modules and also the Catoms we only have neighbourhood communications. We also need external forces to gain locomotion, like the Catoms. But

here it is up to the user to place the cells around the object, or lets say put everything together in a bag, which leads to cells filling up the space around the object and to a quite good lattice of cells. Also for the issue of missing cells in the lattice are answers in [GR12] provided.

3.3.4 SuperBot

What we consider now is a chain type modular robot. This robot is called *SuperBot* because of its ability to disassemble from one configuration and assemble to another dependent of its environment only by its own decision. The main idea behind it, is to use this robot for high-risk environments, such as space explorations or nuclear sites. We can use the robot for this special tasks, because it will be able to adapt itself on different environmental needs, such as rolling over flat areas, climbing steep sand dunes or also climbing ropes. With this abilities the robot will be able to move itself forward on every possible ground.

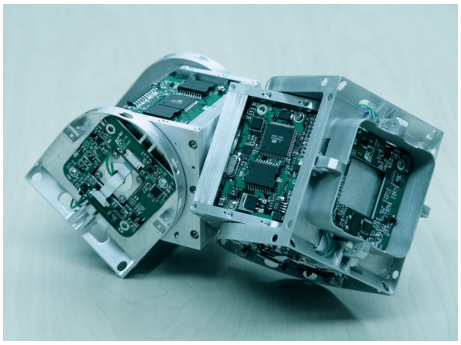


Figure 3.18: SuperBot robot, taken from [ISI14]

As we can see in Figure 3.18 one robot module contains two similar hemispheres, one of them will be the master and the other one the slave. The hemispheres have in sum six connectors for reconfiguration, are able to communicate to other modules using infrared and have an onboard real-time operating system. There is a distributed control mechanism also used for the SuperBot, so that the modules are communication like cellular automata.

As we see in Figure 3.19, there are many possible configurations of the robot modules. Now we are interested in how the SuperBot is able to change from one shape to another.

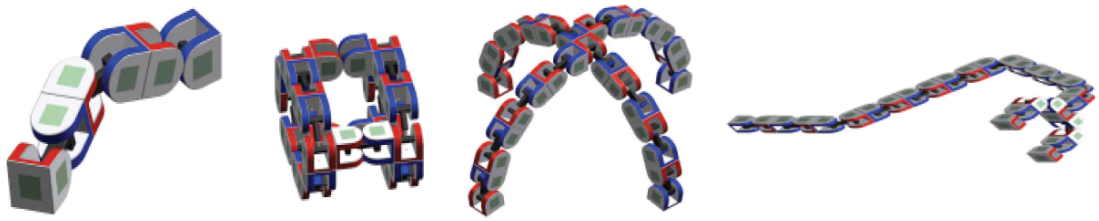


Figure 3.19: Different simulated SuperBot shapes, taken from [ISI14]

For this we consider the algorithm, how the reconfiguration of SuperBot modules is working. As we have shown for the lattice based cellular automata we also need a mathematical description of the modules here. As we do not have a lattice in this approach, we can not use the transition matrix, but we can represent the modules as a graph, where each robot module is represented as a node and a connection is represented as an edge. This is shown in Figure 3.20. Each module can only have connections to its immediate neighbours.

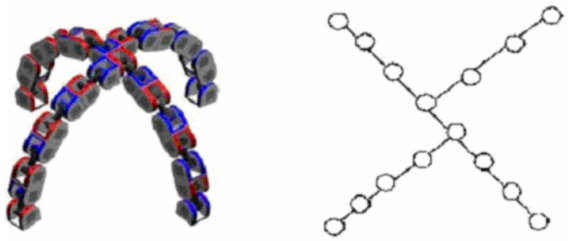


Figure 3.20: A SuperBot and its modular graph, taken from [HS08]

We call two acyclic configurations of modules equal, if they have the same graph. The configuration space is the set of all possible configurations that a specific set of modules is able to form. If one configuration is able to transform into another by specific reconfigure actions, we call these two configurations adjacent. These reconfigure actions are: first an attach action, then a detach action together with motion of the modules. A demonstration how this is working for an example initial configuration of eight modules is shown in Figure 3.21.

For reconfiguring the SuperBot from one shape to another, we only can use this transitions from one configuration to an adjacent one as often, as needed. First the modular graph is distributed in active (grey) and inactive (white) nodes (Figure 3.22). The inactive ones are those, which remain the same during the reconfiguring from the initial to the final shape. A module is able to detect if it is active or not by communicating with its neighbours. If it determines itself inactive it is not part of the reconfiguring process. We thus only have to consider the active nodes.

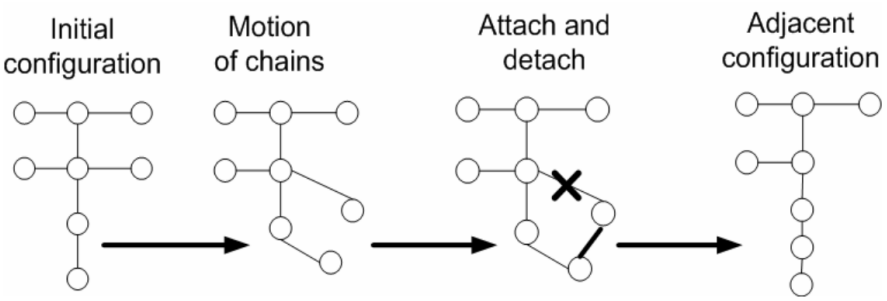


Figure 3.21: The transition of one configuration to an adjacent one by using the reconfigure actions, taken from [HS08]

As we sent the reconfiguration task to the robot (or the robot senses its environment and decides to change its shape), the graph of the goal shape is communicated to all nodes. As the nodes are all homogeneous we not can represent the goal shape using an adjacent or incident matrix etc. For this they use the the *connection number* CN. As each module is able to have maximum four connections, each module calculates its quadruplet CN by sensing how many modules are connected in total for each of the possible four connectors. For example, we consider node A in Figure 3.22: On one edge, the 10 nodes B to K are connected to A, on the next 6 nodes from L to Q and on the last edge the 4 nodes from R to U are connected. A fourth connection is missing. Thus $CN_A = [10, 6, 4, 0]$.

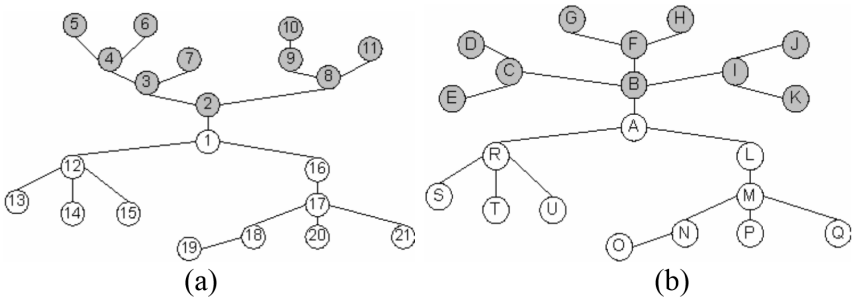


Figure 3.22: Modular graph of the initial and goal configuration, taken from [HS08]

Each module is sensing its CN value on its own and is comparing it with its neighbours. This is the fastest ways and we do not need a lot of computation resources for sensig the whole configuration information. Then we use desired goal CN values to inform the modules how they should change from their current position to the goal configuration (see also [HS08]).
As for different types of environment different shapes of the modular robot will allow a variety of gaits, they investigated in an online gait adaption for the SuberBot.

How one kind of movement is implemented is shown in Figure 3.23.

Figure 3.23: Different tested SuperBot shapes, taken from [ISI14]

Here the SuperBot is moving forward like a caterpillar. This is quite useful for rough ground for example. But if there is a smooth surface, the robot would be faster if it uses a cyclic configuration as the second configuration shown in Figure 3.19. We define the best gait as the one which allows the fastest movement for a particular environment. Initially the SuperBot is told a specific number of different gaits, so it is able to decide which of these gaits and therefore shapes it will use. For the decision it will need four steps executed iteratively:

Step 1: Recognize feature of environment, for example slope gradient

Step 2: Correct the accelerator sensors calibration

Step 3: Learn the preference of each gait under different conditions

Step 4: Select the best gait, with the highest preference for the recognized environment

For the learning they use probability with Bayes' rule, thus the SuperBot does not have to memory previous sensed data or decisions, but is learning with each decision. Here we just want to show the possibility of learning implemented in a modular robot. The gentle reader can find more information about the learning process in [THS12].

Chapter 4

Conclusion

Finally we know many interesting approaches and can compare their usability as well as we are making a final overview over the different possible applications. First we make an overview over this paper, then we compare the different applications of cellular automata in robotics and focus on those capable of self-assembly. At last we look at future work and some interesting ideas.

4.1 Overview

Cellular Automata are cells, which have all the completely same features and are running the same program. Each cell is only able to communicate with its neighbourhood. Hence cellular automata are distributed computers. For this special way of computing there it is possible to create universal computing machines using cellular automata.
We have seen in this paper that there are different neighbourhoods possible, of them the von Neumann neighbourhood and the Moore neighbourhood are used in the majority of applications. Each cell is able to determine its next state-values by considering the state-values of its neighbours. For this there are different rules necessary which have to be implemented on each cell as an algorithm or program. We also can apply specific rules on the cells, which leads to a static final pattern. With this rules we can tell the cells to form specific shapes, which is useful for a self-assembly task.
As we are able choosing specific rules, the cellular automata has a broad variety of applications. Therefore, we considered a broad amount of possible applications for cellular automata in the next step. They are ranging from building mathematical functions, sensor arrays and locomotion to self-assembly. Especially the ability to self-assemble and also repair themselves makes cellular automata this interesting. Thus we investigated in different implementations of self-assembling cellular automata. Now we make a final summary and compare the usability of those implementations.

4.2 Summary and Comparison

As shown in Section 3.2 there is a wide range of possible implementations for cellular automata in robotics. We have seen cellular automata building up mathematical functions, distributed sensors are able of locomotion or accomplishing the task of real time path planning. For each of this applications it is only necessary to tell all cells some rules, how they should calculate their succeeding state-values.
Let us here consider the real time path planning application particularly. Using cellular automata we are able to determine first, if there is at least one possible way. Then we have a dynamic detection of the shortest path. Whenever one cell is occupied by an obstacle or even broken down, the path planning algorithm will find the shortest way not using this cell in the lattice. Thus the application is quite robust, fast in finding possible ways and always finding the shortest way.
To sum it up: we use the advantages of distributed computing to be independent on a single cell, also there is no need for a supervisor as the cells are calculating the shortest way on their own. Hence the cells are accomplishing a complex task by only following simple rules on their own.

Further we have seen four different applications of self-assembly for modular robots. Each of them is using more or less cellular automata rules. While the pebble robots are using only cellular automata structure, e.g. being cubic cells in a lattice and are only using local communication, the other applications have some small differences from that pure cellular automata approach. First a small overview over pebble robots:
Pebble robots are able to communicate with its von Neumann neighbourhood and one program is running on all cells the same time. Thus they are capable of distributed computing. Self-assembly is managed using a steady-state matrix, for duplicating an object the pebble robots organize themselves to duplicate the border or the object first and then fill it. Hence the pebble robots are able to self-assemble, self-reconfigure and self-duplicate.

Let us consider now the other three applications one by one and show their differences to the pebble robots:

ATRON robots are only communicating to their Moore neighbourhood and their possible actions is connecting or disconnecting to neighbours, which is quite the same like for pebble robots. But the ATRON robots are using meta-modules for movement, which is a bit more advanced than the cellular automata approach with only connecting or disconnecting modules in the lattice. Furthermore ATRON has no homogeneous rules on each cell, but a point to point mapping and the initial configuration is needed to be able to reconfigure the shape. So we can summarize that the greates difference between ATRON pebble robots is the algorithm leading to a reconfiguration.

Claytronics are also a bit different from the pure cellular automata approach. As pebble robots, Catoms are able to sense their position dependent on their neighbours. Also Catoms are running all the same algorithm. The first difference is in programming this algorithm: while for pebble robots we use a bottom-up approach with a steady-state matrix, for Catoms they use a top-down approach, programming the final state of the ensemble. This leads to an inability of the cells to functioning alone, but only in groups. Another difference is their ability to not only assemble or disassemble but also to change its colour. Next there have to be external forces applied to gain 3D shapes, or in the case of 3D fax application the cells are in a box and forming a 3D lattice, like the idea of putting the pebble robots in a bag. Thus we have a lot of similarities between pebble robots and claytronics but also some great variations.

SuperBots are quite different from the other three approaches, especially because they are chain shaped and not lattice based. This leads to a graph description and not a matrix like for the others. The chain shape also leads to a completely different field of applications: while ATRON, pebble robots and Catoms can be used to form arbitrary shapes and thus are able to copy every object, SuperBots are robots especially for forming different shapes to perform a variety of gaits. Hence it is not possible to copy objects using SuperBots, as they only have 4 connections per module, which is not sufficient for a 3D object. But there are some great ideas behind this modular approach: As we have seen, the SuperBot is already able to learn about his environment and decide on his own, which shape it chooses. For this also the SuperBot is running a distributed algorithm on each module, like the other approaches also. In this way each module calculates its goal position and the robot is changing its shape.

Now we can see, that all these approaches are a bit different in structure but also in purpose. As we are considering cellular automata, we can conclude, that the pebble robots are the most interesting approach among them.

4.3 Future Work

Until now we were considering what is gained in the field of cellular automata based robots already. But let us take a look in the future. What possibilities have cellular automata? What do we expect and to which fields can we transfer cellular automata robots?

Referring to von Neumann's idea

Let us start our trip to the future in the past: in 1947 John von Neumann started the field of cellular automata. He compared artificial automata with the nervous

system. He says in his lecture in 1966: "The only way to handle a complicated situation with analogue mechanisms is to break it up into parts and deal with the parts separately and alternately, and this is a digital trick [NB66]". We know that with cellular automata, remember: cells in a lattice all running the same algorithm only communicating to theis neighbours, we have simple units working in concert capable of complex tasks. The cellular automata we know distribute a complex task on small computing units and the overall outcome is the solution to the complex task, by solving many small easy local tasks. This is exactly what von Neumann meant in this description of artificial automata.

Another interesting point is von Neumanns view on errors: "It's very likely that on the basis of the philosophy that every error has to becaught, explained, and corrected, a system of the complexity of the living organism would not run for a millisecond. [NB66]" Further von Neumann mentioned that a system has to detect an error as soon as one occurs, but then does not have to stop immediately, but test if this error matters or not. If not, the system ignores the error. If the error matters, the system bypasses the erroneous area and analyses the region separately while running using the bypass [NB66]. This is just what natural automata, like the nervous system, is doing with errors: ignore it and work around. Just consider the tactile sense: whenever we loose nerves in our skin because of cuts or similar wounds, all nerves around undertake the tasks of this nerve, so that we are able to feel on that part of the skin further on. Are we already able to use cellular automata like natural automata in the case of errors? Let us treat this point with an example: Remember the real time path planning approach. Whenever a cell is breaking down, it will be treated as an obstacle. Therefore we can say: yes, cellular automata are already able to function with errors. But we have not found a strategy to treat this errors other than replacing, yet.

As last great point von Neumann was talking about in 1966 we consider the ability of self-reproduction. He said in his lecture "But it is easier [...] to construct an automaton which can produce an object starting from a logical description." In our mathematical Section, we have seen, that for the description of an object there only is a steady-state matrix needed. Implemented in robotics this leads to the production of an object only by its logical, mathematical description. This is implemented in pebble robots, for example.
Hence we can say this aspect is done already. What von Neumann also predicted, was a construction of robots using a description like DNA. As we have seen in Section 1.2 we can use a set $A+B+C+D$ of modules which are able to generate itself. Well we are by now constructing modules and they are able to combine itself to a set of modules, if we tell them to. But we did not find a set of module which is also able to totally construct its own parts out of loose material, like natural automata, e.g. cells using DNA and proteins, are doing pretty well since thousands of years. Here we have found a point where future work will be quite useful and interesting.

To sum it up: We have seen that quite much of these 50year old ideas are possible to implement right now. Hence we are focussing on the fields where future work will be interesting and what we will be able to gain in the next years using cellular automata.

Best Ideas for Future Implementations

We can start with the open points from von Neumann's ideas. First it will be of great use, if cellular automata are not only able to work around errors, but to analyse and also repair them if this is possible. Thus it will be possible, that the complex cell structure can **heal** a single cell.

The next big goal will be constructing a huge cellular automata complex, which is able to **produce its own cells** like the DNA aspect we mentioned before. If this is possible the cellular automata complex will be able to grow and evolve to a more complex machine on its own.

Both of this tasks are focusing on cellular automata in robotics, but there are no more applications in other fields than classical robotics considered.

A large field where cellular automata robots can be of great use is **medical treatment**. As we see in [Cav05] there is a large potential for nanorobots in surgery. Imagine, we can insert some nano scaled cellular automata robots in the body and assemble whenever they are needed on a specific point, e.g. they measure some values of the blood running through arteries. Whenever it is at a risk, they assemble to a excavator to reduce the number of lipoproteins in the arteries. So they will be able to minimise the likelihood of a cardiovascular occlusion. After finishing this task, the robots can disassemble and flow with the blood to measure other risks in the body. As this is more science fiction than reality, let us consider which challenges we have to deal with: One of the main issues is the size of the cellular automata robots. As they have to be at nano scale to be able to flow with the blood, we have to gain great advantages from now, remember at the moment we are at 12mm in size for example the pebble robots. The next big challenge is to handle the environment in vivo. This is quite more difficult to handle than a usual cellular automata lattice. Also the sensing, the localization of a cell at which point in the body it is right now are some more challenges to go for. But if this is possible that cellular automata nano robots are able to use their advantages in a human body, we will be able to handle probably some of the most severe illnesses of this time.

Let us consider also another severe disease of humans: **cancer**. As we know, cells are only able to survive in an appropriate environment, if there are changes they might die. Further we know, that above a temperature of 42 C cells will decay and

finally die. Thir process is one approach to treat cancer: we heat the tumor cells with paramagnetic activity over 43 C, while the tissue around the tumor will stay beyond 42 C. Jinghua et al. [JZJ14] found a way to simulate the temperature in the cells using cellular automata models. But this approach has some errors, if the millimetre scaled magnets have some spacial difference to their desired position. If now they are not only simulate the heat using cellular automata but implant instead of permanent magnets cellular automata with paramagnetic functionality, they can localize their correct position in the body and thus the simulation errors can be corrected. Thus it might be able to maximise the likelihood of only heating the tumor over 43 C.

This were just some aspects, how cellular automata can be useful in other fields than classical robotic. As we have seen, this approach might be able to improve the human life quite a lot.

4.3.1 Resume

The lesson is clear: with cellular automata we have a distributed computing opportunity for improving many areas of the life on earth. If this will be in the field of surgery or for space missions or even for improving evacuation of buildings: there are many applications which gain great improvements if they take the advantages of a cellular automata structure.
Especially in the field of robotics we know many useful implementations of cellular automata. They can enable the duplication of arbitrary objects, also enabling the functionality of a 3D fax or a quasi-life communication with a person far away. In regions of high risk cellular automata robots are able to accomplish tasks on their own without a human supervisor needed. Thus there are many aspects where cellular automata robots can be of great use.

List of Figures

Bibliography

[AAK12] Syed Usman Ahmed, Arsalan Akhter, and Faraz Kunwar. Cellular automata based real time path planning for mobile robots. In *Control Automation Robotics & Vision (ICARCV), 2012 12th International Conference on*, pages 142–147. IEEE, 2012.

[ABF+06] M Aubert, M Badoual, S Fereol, C Christov, and B Grammaticos. A cellular automaton model for the migration of glioma cells. *Physical biology*, 3(2):93, 2006.

[AS13] Kohei Arai and Steven Ray Sentinuwo. Mobile robot motion using cellular automaton model to avoid transient obstacles. *International Journal of Modern Education & Computer Science*, 5(8), 2013.

[BBCM01] C Behring, M Bracho, M Castro, and JA Moreno. An algorithm for robot path planning with cellular automata. In *Theory and practical issues on cellular automata*, pages 11–19. Springer, 2001.

[BKRT01] Zack Butler, Keith Kotay, Daniela Rus, and Kohji Tomita. Cellular automata for decentralized control of self-reconfigurable robots. In *Proc. IEEE ICRA Workshop on Modular Robots*. Citeseer, 2001.

[BØ04] David Brandt and Esben Hallundbæk Østergaard. Behaviour subdivision and generalization of rules in rule based control of the atron self-reconfigurable robot. In *Proceedings of International Symposium on Robotics and Automation (ISRA)*, pages 67–74. Citeseer, 2004.

[BT12] Francesco Berto and Jacopo Tagliabue. Cellular automata. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2012 edition, 2012.

[Cav05] Adriano Cavalcanti. Robots in surgery. In *Plenary Lecture, Euro Nano Forum*, 2005.

[CC07] David Johan Christensen and Jason Campbell. Locomotion of miniature catom chains: Scale effects on gait and velocity. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2254–2260, 2007.

[COJ+] David Johan Christensen, Esben Hallundbaek Ostergaard, David Johan, Christensen Esben, Hallundbaek Ostergaard, and Henrik Hautop Lund. Metamodule control for the atron self-reconfigurable robotic system.

[COL04] David Johan Christensen, Esben Hallundbok Ostergaard, and Henrik Hautop Lund. Metamodule control for the atron self-reconfigurable robotic system. In *Proceedings of the The 8th Conference on Intelligent Autonomous Systems (IAS-8)*, pages 685–692. Citeseer, 2004.

[CSJH08] Adriano Cavalcanti, Bijan Shirinzadeh, Robert A Freitas Jr, and Tad Hogg. Nanorobot architecture for medical target identification. *Nanotechnology*, 19(1):015103, 2008.

[Dav09] Philip Davies. 3d cellular automata. *BPC Research Bulletin*, (5), 2009.

[DRGL+06] 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 *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1462–1468. IEEE, 2006.

[FJ] Institute for Molecular Manufacturing Freitas Jr, Robert A. Nanomedicine and medical nanorobotics.

[GGJ05] Saul Griffith, Dan Goldwater, and Joseph M Jacobson. Self-replication from random parts. *Nature*, 437(7059):636, 2005.

[GKR10a] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2485–2492. IEEE, 2010.

[GKR10b] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2485–2492. IEEE, 2010.

[GM04] Seth C Goldstein and Todd C Mowry. Claytronics: An instance of programmable matter. 2004.

[GMC+09] Seth Copen Goldstein, Todd C Mowry, Jason D Campbell, Michael P Ashley-Rollman, Michael De Rosa, Stanislav Funiak, James F Hoburg, Mustafa E Karagozler, Brian Kirby, Peter Lee, et al. Beyond audio and video: Using claytronics to enable pario. *AI Magazine*, 30(2):29, 2009.

[GR10] Kyle Gilpin and Daniela Rus. Self-disassembling robot pebbles: New results and ideas for self-assembly of 3d structures. In *IEEE International Conference on Robotics and Automation Workshop Modular Robots: The State of the Art*, pages 94–99, 2010.

[GR12] Kyle Gilpin and Daniela Rus. A distributed algorithm for 2d shape duplication with smart pebble robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3285–3292. IEEE, 2012.

[Gri04] Saul Thomas Griffith. *Growing machines*. PhD thesis, Massachusetts Institute of Technology, 2004.

[HM99] Owen Holland and Chris Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.

[HS08] Feili Hou and Wei-Min Shen. Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots. Pasadena, CA, May 2008.

[HS10] Feili Hou and Wei-Min Shen. On the complexity of optimal reconfiguration planning for modular reconfigurable robots. Anchorage, Alaska, USA, May 2010.

[HS13] Feili Hou and Wei-Min Shen. Graph-based optimal reconfiguration planning for self-reconfigurable robots. *Robotics and Autonomous Systems*, 2013.

[Hut10] Tim J Hutton. Codd's self-replicating computer. *Artificial life*, 16(2):99–117, 2010.

[HW88] Susan Hackwood and Jing Wang. The engineering of cellular robotic systems. In *Intelligent Control, 1988. Proceedings., IEEE International Symposium on*, pages 70–75. IEEE, 1988.

[ISI14] Polymorphic Robotics Laboratory Information Sciences Institute. Superbot, 2014. URL: `http://www.isi.edu/robots/presentations/SuperBot-Public.pdf`.

[JMP09] David H Jones, Richard McWilliam, and Alan Purvis. Designing convergent cellular automata. *Biosystems*, 96(1):80–85, 2009.

[JMP11] D Jones, Richard McWilliam, and Alan Purvis. Design of selfassembling, self-repairing 3d irregular cellular automata. *New Advanced Technologies (A. Lazinica, ed.)*, pages 373–394, 2011.

[JOL04] Morten Winkler Jorgensen, Esben Hallundbk Ostergaard, and Henrik Hautop Lund. Modular atron: Modules for a self-reconfigurable robot. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 2068–2073. IEEE, 2004.

[JZJ14] Wu Jinghua, Guo Zhendong, and Chen Jian. Efficient cellular automata method for heat transfer in tumor. *Journal of Heat Transfer*, 136, 2014.

[Lan84] Christopher G Langton. Self-reproduction in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1):135–144, 1984.

[Lan86] Christopher G Langton. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1):120–149, 1986.

[MCH94] Melanie Mitchell, James P Crutchfield, and Peter T Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D: Nonlinear Phenomena*, 75(1):361–391, 1994.

[Mea08] Ross Mead. *Cellular Automata for Control and Interactions of Large Formations of Robots*. PhD thesis, Southern Illinois University, 2008.

[MI97] Kenichi Morita and Katsunobu Imai. Logical universality and self-reproduction in reversible cellular automata. In *Evolvable Systems: From Biology to Hardware*, pages 152–166. Springer, 1997.

[MK07] Satoshi Murata and Haruhisa Kurokawa. Self-reconfigurable robots. *Robotics & Automation Magazine, IEEE*, 14(1):71–78, 2007.

[MW08] Ross Mead and Jerry B Weinberg. 2-dimensional cellular automata approach for robot grid formations. In *AAAI*, pages 1818–1819, 2008.

[NB66] John von Neumann and Arthur W Burks. Theory of self-reproducing automata. 1966.

[ØKBL06] Esben Hallundbæk Østergaard, Kristian Kassow, Richard Beck, and Henrik Hautop Lund. Design of the atron lattice-based self-reconfigurable robot. *Autonomous Robots*, 21(2):165–183, 2006.

[PCK+06] Padmanabhan Pillai, Jason Campbell, Gautam Kedia, Shishir Moudgal, and Kaushik Sheth. A 3d fax machine based on claytronics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 4728–4735. IEEE, 2006.

[PLI07] Rolf Pfeifer, Max Lungarella, and Fumiya Iida. Self-organization, embodiment, and biologically inspired robotics. *science*, 318(5853):1088–1093, 2007.

[RS08] Michael Rubenstein and Wei-Min Shen. A scalable and distributed approach for self-assembly and self-healing of a differentiated shape. Nice, France, September 2008.

[RS09] Michael Rubenstein and Wei-Min Shen. Scalable self-assembly and self-repair in a collective of robots. St. Louis, Missouri, USA, October 2009.

[SSW02] Wei-Min Shen, Behnam Salemi, and Peter Will. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *Robotics and Automation, IEEE Transactions on*, 18(5):700–712, 2002.

[THS12] Luenin Barrios Teawon Han, Nadeesha Ranasinghe and Wei-Min Shen. An online gait adaptation with superbot in sloped terrains. Guangzhou, China, December 2012.

[Uni14] Carnegie Mellon University. Claytronics, 2014. URL: `http://www.cs.cmu.edu/~claytronics/`.

[Wol84] Stephen Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1):1–35, 1984.

[Wol98] Stephan Wolfram. Cellular automata as models of complexity. *Nonlinear Physics for Beginners: Fractals, Chaos, Solitons, Pattern Formation, Cellular Automata, Complex Systems*, 311:197, 1998.

[Wol14] Stephen Wolfram. Wolframmathworld, May 2014. URL: `http://mathworld.wolfram.com/`.