Self-Evolution in a Constructive Binary String System

Peter Dittrich[1] and Wolfgang Banzhaf[2]

Dept. of Computer Science, University of Dortmund [3]
D-44221 Dortmund (Germany)

- DRAFT April 20, 1998-

Abstract

We examine the qualitative dynamics of a catalytic self-organizing reaction system of binary strings that is inspired by the chemical information processing metaphor. A string is interpreted in two different ways: either (i) as raw data or (ii) as a machine which is able to process another string as data in order to produce a third one. This paper focuses on the phenomena of evolution whose appearance is notable because no explicit mutation, recombination or artificial selection operators are introduced. We call the system self-evolving because every variation is performed by the objects themselves in their machine form.

Keywords: artificial chemistry, autocatalytic reaction system, molecular computing, prebiotic evolution, self-organization, self-programming

[1]e-mail: dittrich@@LS11.informatik.uni-dortmund.de
[2]e-mail: banzhaf@@LS11.informatik.uni-dortmund.de
[3]URL: http://LS11-www.informatik.uni-dortmund.de

1 Introduction

In recent years computer simulations have been used to study the phenomena of life, not by simulating life as-it-is (weak AL) but by instantiating life as-it-could be (strong AL) [22]. AL systems have been used, e.g. for studying questions concerning the emergence and quality of organizations [26, 13, 19], the process of diversification [27], the origin of replicators [25], or morphological evolution [21].

An important property of most strong AL systems is that they contain the ability for self-reference. For instance Ray's Tierra organisms are able to read, copy and modify their own code [27]. In Fontana's algorithmic chemistry every object is a character string able to process other objects by using the lambda-calculus which maps the character string into an (active) function [13]. The dualism inherent in those systems can be traced back to Gödel [15] who defined a mapping of mathematical statements into natural numbers that allowed self-reference, to Turing's universal machine [31], and to von Neumann's stored program computer [7].

The system discussed here is inspired by a chemical reaction dynamics, where molecules collide and interact to create new molecules forming metabolic networks. This contribution tries to give insight into the origin and evolution of these networks [2, 3]. Related work [4, 13, 20, 23, 30] focuses on the static structure of the emerged networks. Here, we will concentrate on dynamic phenomena, especially on the emergence of prebiotic evolution [2, 17]. Evolutionary phenomena can be observed, although no explicit fitness, mutation, recombination, or selection operators are used. Variations are only performed by the objects (molecules, binary strings) themselves when they act in machine form.

1.1 Self-organization and Self-Evolution

It seems worthwhile to start our considerations with a short note on self-organization and what we mean by "self-evolution" in the context of this paper. **Self-organization** is a process where a system is organized while the components directing this process are part of the system. This kind of view of "self-organization" implies: 1.) Every organization phenomenon becomes a result of self-organization provided we enlarge the system boundaries. 2.) Self-organization need not to be directed, e.g. towards higher complexity.

When transforming this linguistic abstract description of self-organization into the language of computer science it seems to be straight forward to represent an organizer as a program and the data a program is working on as the target for organization. In order to close the loop inherent in self-organization we need a mapping from data to machines or vice versa, yielding the dualism of the system components. The information residing in the system should either act as *active* machines or should be processed as *passive* data.

The organization process can also be expressed in the terms of mathematics as the application of (active) operators to (passive) data. An example for this approach is [4] where the operands are binary strings. Operators are formed out of binary strings by "folding" them into a matrix.

The term **self-evolution** should refer to an evolutionary process within a population system where the components responsible for the evolutionary behavior are (only) the individuals of the population system itself. Every variation is carried out by the individuals. Selection pressure is generated implicitly through interaction among the individuals and not by external agents.

The system must not contain a component which can be identified as a fitness function or global operators performing selection or variation (e.g. cross-over).

The outline of this paper is as follows: Section 2 gives a general introduction to algorithmic reaction systems. In Section 3 we briefly describe the special reaction mechanisms used for this contribution. Section 4 summarizes methods for investigation and visualization. In Section 5 we discuss four qualitative types of dynamic self-organization phenomena by showing typical simulation results, namely (1) extinction, (2) emergence of an organization structure, (3) exploration/innovation, and (4) evolution.

2 Algorithmic Reaction Systems

The system used for the experiments in this paper is in general called an **artificial chemistry** or **algorithmic reaction system**, because the interaction between its elements is specified by an algorithm. It consists of the following three components:

(1) a set of objects S: These objects may be abstract symbols [32], character sequences [20], lambda-expressions [13], binary strings [4, 30], numbers [5] or proofs [14]. Here, we use binary strings with a constant length of 32 bits, thus, $S = \{0,1\}^{32}$. In a basic setting, the soup has no topological structure so that its state can be noted as a concentration vector.

(2) a collision or reaction rule: The collision rule defines the interaction among two objects $s_1, s_2 \in S$ which may lead to the production of a new object $s_3 \in S$. This is denoted as:

$$s_1 + s_2 \Longrightarrow s_3$$

Note that the operator "+" is *not* commutative, in general.

(3) an algorithm simulating the reacation vessel: In this contribution we use the following algorithm that simulates a well-stirred tank reactor with mass-action kinetics which assures that the probability of a collision is proportional to the product of the concentration of the colliding objects (Eq. 1, below). The algorithm can also be found with minor modifications in [4, 13, 20, 30].

Reactor algorithm

1. Select two objects s_1, s_2 from the soup randomly, without removing them.
2. If there exists a reaction $s_1 + s_2 \Longrightarrow s_3$ and the filter condition $f(s_1, s_2, s_3)$ holds, replace a randomly selected object of the soup by s_3.

The replaced objects form the **dilution flux** of the system. The **filter** f can be used to block lethal objects and to introduce elastic collisions easily. The term **collision** refers to one execution of the two steps of the reactor algorithm. A collision is called **elastic** if no product s_3 is inserted into the soup. So, a collision of two objects s_1, s_2 is elastic, if no product is defined by the reaction rule or if the filter condition prohibits the insertion of the product. An object is said to be **lethal** if it is able to replicate in an unproportionally large number in almost any ensemble configuration [4].

The interaction scheme of the algorithm may be written as a chemical reaction equation [13]:

$$s_1 + s_2 + X \longrightarrow s_1 + s_2 + \begin{cases} \frac{1}{2}s_3 & \text{with } s_1 + s_2 \Longrightarrow s_3 \\ \frac{1}{2}s_3' & \text{with } s_2 + s_1 \Longrightarrow s_3' \end{cases}$$

In other words, s_1 and s_2 are not consumed but act as catalysts of the reaction. The **raw material** X is used to balance the equation. It does not appear explicitly in our system and could be interpreted as computational resources like processing time or memory [27]. Reaction systems exhibiting this kind of reaction scheme are able to form **hypercyclic** organizations [10].

In order to measure the running time of the algorithm we call M iterations (collisions) a **generation**, where M is the soup or reactor size. Using "generations" rather than "number of iterations" allows an easier comparison of runs with different soup sizes.

2.1 A model for the algorithmic reactor

In a setting corresponding to a well-stirred tank reactor the state of the system can be described by a concentration vector $\mathbf{x} = (x_1, \ldots, x_n)$ with $x_1 + \cdots + x_n = 1$ and $x_i > 0$. x_i is the concentration of string type s_i. For a large and constant soup size the behavior of the reactor algorithm is modeled by the **catalytic network equation** [29]:

$$\frac{dx_k}{dt} = \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_{ij}^k x_i x_j - x_k \sum_{i,j,k=1}^{n} \alpha_{ij}^k x_i x_j \qquad k = 1, \ldots, n \tag{1}$$

with second order rate constant α_{ij}^k for the reaction $i + j \overset{\alpha_{ij}^k}{\Longrightarrow} k$. Here the rate constants become

$$\alpha_{ij}^k = \begin{cases} 1 & \text{if } s_1 + s_2 \Longrightarrow s_3 \text{ and the filter condition } f(s_1, s_2, s_3) \text{ holds,} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Equation 1 becomes the famous **replicator equation** [17, 29] if $\forall i, j, k \in \{1, \ldots, n\} : \alpha_{ij}^k > 0 \Longrightarrow i = k \vee j = k$ holds.

Simulating systems with a large number n of different objects becomes difficult. For the special case of only a small fraction of all possible objects being present in the reactor, a technique called **meta dynamics** is used elsewhere, where the ODE System (Eq. 1) is treated dynamically and updated under certain conditions [2, 12, 20]. For example, equations are removed when the corresponding concentrations fall below a given threshold. The advantage of this method is that a potentially infinite number of objects can be handled by still using an ODE framework. Problems arise, however, if the diversity (e.g. the number of different objects in the reactor) becomes high, as will be the case here.

3 Special 32-bit Reaction Mechanisms

We will now briefly describe the special reaction mechanisms used for this contribution, namely the AND reaction and the automata reaction.

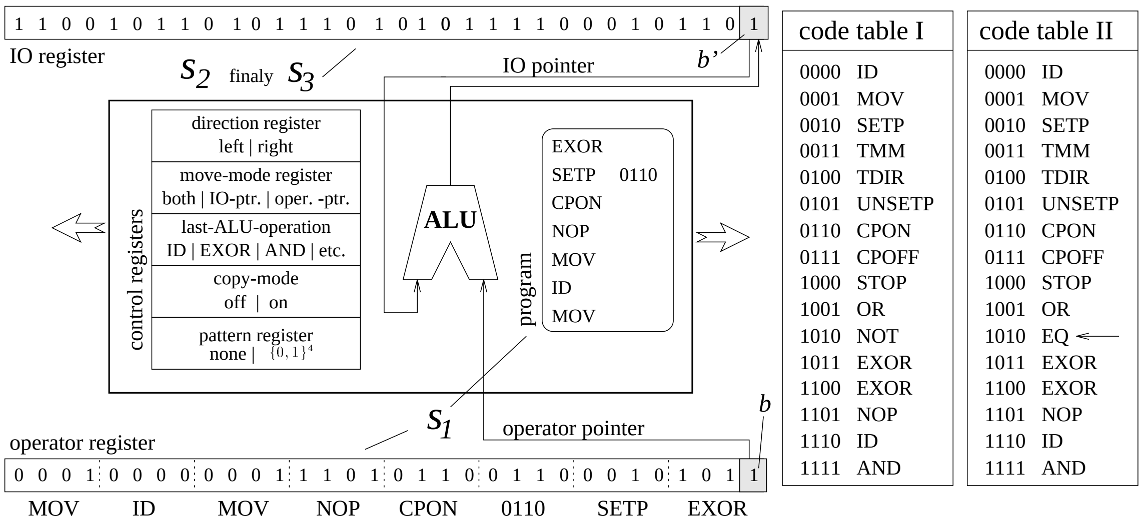


Figure 1: *Automaton, resulting from folding s_1. It carries out the reaction $s_1 + s_2 \Longrightarrow s_3$. s_1 is written into the operator register and specifies the program. The IO register is initialized with s_2 and contains the result s_3 after running the program.*

3.1 AND reaction

The AND reaction is simply the bitwise logic AND of strings s_1 and s_2.

$$\forall i \in \{0, 31\} : s_3^{(i)} = s_1^{(i)} \wedge s_2^{(i)}$$

where $s^{(i)} \in \{0, 1\}$ denotes the i-th bit of string s. This reaction is used for test and reference, because its behavior can be easily described and understood.

3.2 Automata reaction

The **automata reaction** is based on a finite state automaton which is a mixture of a Turing-machine and a register machine. It has also been inspired by the Typogenetics of Hofstadter [18].

The **automata reaction** instantiates a deterministic reaction $s_1 + s_2 \Longrightarrow s_3$, where $s_1, s_2, s_3 \in \{0, 1\}^{32}$. In order to calculate the product s_3, string s_1 is "folded" into an automaton A_{s_1}, which receives s_2 as input. The construction of A_{s_1} ensures that the automaton will halt after a bounded finite number of steps. Because A_{s_1} is a deterministic finite automaton, the automata reaction defines a functions $\{0, 1\}^{32} \times \{0, 1\}^{32} \longrightarrow \{0, 1\}^{32}$.

Figure 1 shows the structure of the automaton. It contains two 32-bit registers, the **IO register** and the **operator register**. At the beginning operator string s_1 is written into the operator register and operand s_2 into the IO register. The program is generated from s_1 by simply mapping successive 4-bit segments into instructions. Note that two slightly different mappings are used (Fig. 1, right). The resulting program is executed sequentially, starting with the first instruction. The automaton halts after the last instruction has been executed or in case of a STOP instruction. There are no control statements for loops or jumps in the instruction set.

Each 32-bit register has a pointer, referring to a bit location. The **IO pointer**, referring to a bit b' in the IO register and the **operator pointer**, referring to a bit b in the operator register, respectively. For each collision both pointers are initialized with position 0, as shown in Fig. 1. Bit b and b' are inputs to the **ALU** which is able to perform logic operations on two bits. The ALU result is stored at the IO pointer location, therefore replacing b'.

The execution of a logic instruction (ID, NOT, OR, AND, EXOR or EQ) is divided into two steps: 1.) The ALU processes bit b and b', and replaces b' by its output. 2.) The pointers are moved one bit left or right, according to the **direction register**. The **move-mode register** determines whether only the IO pointer, only the operator pointer, or both pointers are moved. The move-direction and the move-mode can be toggled by TDIR and TMM, respectively.

The MOV instruction is the most complicated one. Its execution depends on all control registers. If the pattern register is set to *none* and the copy-mode is switched off, the pointers are moved one bit according to the direction and move-mode register. This is equal to the second step of the execution of a logic instruction. If the pattern register is set to a pattern (a 4-bit bit string) the pointers are moved until the pattern is found in the operator register (max. 32 steps). If the copy-mode is switched on, the last ALU operation is executed before each step. Thus, one MOV instruction may trigger many ALU operations. The pattern register is set by the 8-bit instruction SETP `<pattern>` and cleared by UNSETP. The copy-mode is switched on and off by CPON and CPOFF, respectively.

Table 1 shows the operation of the automaton for the following example:

$$s_1 = 0001\,0000\,0001\,1101\,0110\,0110\,0010\,1011$$
$$s_2 = 1100\,1011\,0101\,1101\,0101\,1110\,0010\,1101$$
$$s_3 = 1100\,1011\,0101\,1101\,0101\,0110\,0000\,0110$$

In hex notation, this reads: $101d662b + cb5d5c2d \Longrightarrow cb5d5606$. Table 2 shows more examples in hex notation that should give an impression of the static properties of the automata reaction. The first noticeable property is that the structure of the product s_3 is similar to its "parents" s_1, s_2. This indicates that there is a correlation between s_1, s_2 and s_3 which is a prerequisite for evolution. Secondly, variations of string sequences appear usually at the edges of a sequence, mostly on the right side and sometimes on the left. Modification of the core bits only is very seldom. This is due to the fact that the pointers are always initialized to bit position 0 and the direction register is always set to *left*. Passive replication[4] is common. About 30% of all randomly generated strings are passive replicators. Active self-replication[5] is rare. About 0.004% randomly generated sequences are active replicators. As we will demonstrate later, the system evolves towards active replication, provided the soup is large enough for an explorative or evolutionary behavior.

4 Investigation and Visualization of System Behavior

Designing a system that shows behavior of enormous complexity is surprisingly easy. But to visualize and to explain the behavior in detail is much harder. Methods for investigation can be divided into macroscopic, mesoscopic and microscopic approaches.

[4] A string s_1 is called **passive replicator**, if $\forall s_2 \in S : s_1 + s_2 \Longrightarrow s_2$.
[5] A string is called **active self-replicator**, if $\forall s_2 \in S : s_1 + s_2 \Longrightarrow s_1$

s_1	command	comment
1011	EXOR	The exclusive-or product of b' (referred to by the IO pointer) and b (referred to by the operator pointer) is written at the position of b'. Therefore, the 0. bit of the IO register becomes 0. Then both pointers are moved one bit position to the left.
0010 0110	SETP 0110	The pattern register is set to 0110. If the pattern is set, a subsequent MOV operation will move the pointers until the pattern is found in the operator register (max. 32 steps).
0110	CPON	The copy-mode is switched on. Every time the pointers are moved the ALU operation is performed (here, EXOR).
1101	NOP	No operation. The registers are not modified.
0001	MOV	The pointers are moved to the left until the pattern 0110 is found in the operator register, which happens at the 8. bit position. Because the copy-mode is activated (by CPON), the last ALU operation (here: EXOR) is executed for every bit passed.
0000	ID	The 8. bit of the operator register is copied into the IO register, and both pointers are moved one step to the left.
0001	MOV	Both pointer are moved to the next appearance of 0110, namely to the 12. bit position. This time the ID operation is executed at each step.

Table 1: Example for the automata reaction.

No	s_1 operator	s_2 operand	s_3 product	comment
1	`101d662b`	`cb5d5c2d`	`cb5d5606`	typical random reaction
2	`f82710fb`	`73fbc890`	`73fbc891`	typical random reaction
3	`98ac35f1`	`4c22bc78`	`4c22bc7c`	typical random reaction
4	`c1126a24`	`f9c8a25e`	`c1c8a25e`	typical random reaction with TDIR
5	`9a716b98`	`9878fa02`	`9878fa02`	passive replication
6	`9a716b98`	`00000000`	`00000000`	passive replication
7	`1e1ca260`	`078f21ff`	`1e1ca260`	active replication
8	`1e1ca260`	`00000000`	`1e1ca260`	active replication

Table 2: *Examples for the automata reaction.*

A **macroscopic method** observes the whole system, by mapping its high-dimensional state into a low-dimensional space, where every component of the system should be considered. An example from thermodynamics is "temperature". A **mesoscopic method** considers only a small subset of the systems components. It still accumulates many microscopic events into low dimensional values (e.g., the concentration of a specific substance in a region). **Microscopic methods** are visualizing elementary details of the system for instance the history of a specific object or memory cell.

Here, we shall apply only macroscopic methods, which are explained in the following. The soup (population) $P = \{s_1, \ldots, s_M\}$ is defined as a multi set on S:

Diversity: Various measures for the diversity of a population exist. In ecology, diversity is often defined as the number of different groups or taxa. The dynamic behavior of this diversity depends on the definition of the grouping. A fine grained grouping results in a diversity fluctuating over time. This is due to the fact that the number of individuals belonging to one group is not considered. Here, we use a simple, fine grained grouping, where every possible genotype forms a group. Then, the resulting (absolute) diversity Div_{abs} becomes the number of different string types in P:

$$Div_{abs}(P) = |\{s | s \in P\}|.$$

To get a relative value $Div(P)$ in the range $]0, 1]$ the absolute diversity is divided by the soup size M. In the future, the term "diversity" refers to the **relative diversity**

$$Div(P) = \frac{Div_{abs}(P)}{M}.$$

Distance distribution complexity (DCC): One problem of the diversity measure mentioned before is that the distance between different groups is not taken into account. For example an ecology with 10 horses and 10 zebras would have the same diversity as an ecology with 10 horses and 10 amoeba. The DDC, recently introduced by Kim [21], is able to distinguish these cases: Given a discrete distance measure $D : P \times P \to \mathbb{N}$ the **distance distribution** is defined here, as the relative frequency of the distance value $d \in \mathbb{N}$:

$$f(d) = \frac{2}{M(M-1)} |\{(s_i, s_j) | s_i, s_j \in P, i < j, D(s_i, s_j) = d\}|.$$

The **distance distribution complexity** (DDC) is then defined as the Shannon entropy of the distribution of distance values:

$$DDC(P) := -\sum_d f(d) \log(f(d)).$$

For the following analysis D is defined as the Hamming-distance. This simple and fast measure is sufficient, because no shifting and no insertion or deletion of bits is taking place. Note, a high DDC indicates a high diversity and vice versa, but the DDC can increase even if the diversity decreases (e.g. Fig. 2).

Productivity: The **productivity** $Prod(P)$ is the probability, that a collision of two strings is reactive. If the productivity is zero, no new string is produced and inserted into the soup. To measure the productivity during a simulation, the number of iterations in which a string s_3 is inserted into the soup is counted for M steps. The result is divided by M. If the soup size M is constant the productivity is equal to the dilution flux.

Innovativity: The innovativity is the probability that a collision produces an object that is new, with respect to a given time window $[t_1, t_2]$. Let P_t be the soup after the t-th iteration of the reactor algorithm. Then the **(total) innovativity** is defined for the time window $[0, t]$ as

$$Inn(P_t) = p(s_3 \text{ is inserted into the soup} \quad \text{and} \quad s_3 \notin \bigcup_{t'=0}^{t} P_{t'}).$$

The total innovativity has been measured by counting the number of completely new strings that are inserted into the soup during M iterations of the reactor algorithm. The result is divided by M.

Reaction table: Given a set of objects $A = \{s_1, \ldots, s_{n_a}\}$ then the corresponding reaction table $RT : \{1, \ldots, n_a\}^2 \to \{1, \ldots, n_a, -, *\}$ is defined as:

$$RT(i,j) = \begin{cases} - & \text{if the collision of } s_i, s_j \text{ is elastic,} \\ k & \text{if } s_i + s_j => s_k \text{ and } s_k \in A, \\ * & \text{if } s_i + s_j => s_k \text{ and } s_k \notin A. \end{cases}$$

The term "the collision of s_i, s_j is elastic" refers to step 2 of the reactor algorithm, where s_i, s_j are selected but no product is inserted into the soup. For the reaction tables shown in this paper, the set of objects is sorted according to their concentration: $\forall s_i, s_j \in A : [s_i] \geq [s_j] \iff i \geq j$.

5 Qualitative Types of Dynamic Self-Organization Phenomena

In this section we show experimental results to discuss four qualitative types of self-organization processes. In all experiments the objects are binary strings of fixed size 32 bit. The soup is always initialized with random strings, resulting in an initial diversity of approximately 1. The phenomena will be illustrated by visualizing "typical" runs, because averaging over a series of runs with the same parameter set but different random numbers is not feasible.

5.1 Extinction

Figure 2 shows the behavior of the AND reaction. The diversity drops quickly to $1/M$ (only one string type left) and the DDC to zero, indicating that the soup is exploited by a single type, the lethal string 00000000. It is able to replicate with every other string and is furthermore produced through many reaction pathways, e.g. $f120000 + 0004711a \Longrightarrow 00000000$.

At the outset the DDC increases, because intermediate products appear which are characterized by a small number of ones (e.g. 00000200), resulting in a higher complexity of the distribution of distances during generation 0-2 (Fig. 2, right).

Extinction is a typical phenomenon where lethal strings appear likely [4]. The resulting organization is so simple that further examination is normally not warranted. Extinction can be circumvented by filtering lethal objects [4] or by prohibiting replication reactions [13] (Eq. 3, below).

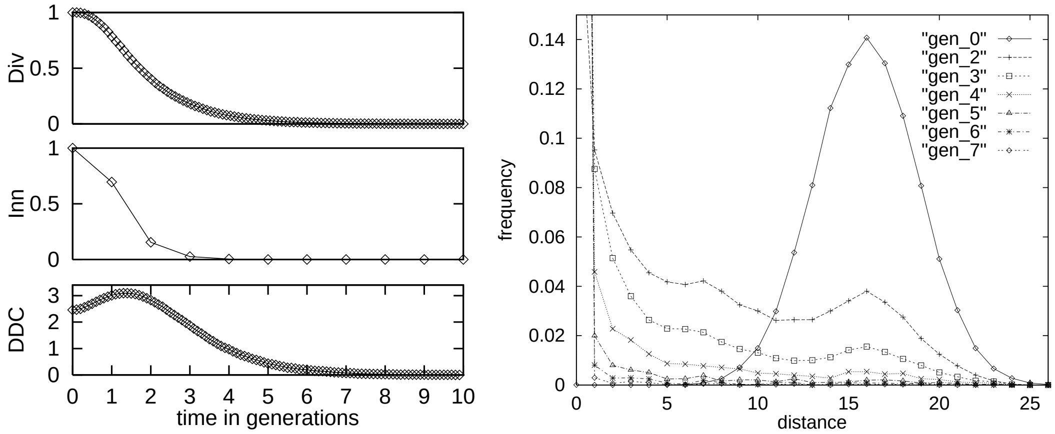


Figure 2: *Example for extinction. Parameters: AND reaction, no filter condition,* $M = 10^4$*. Diversity and DDC drop to zero because the system is exploited by the destructor* 00000000 *which remains as the only surviving string. Right: Distance distribution for the generation 0-7.*

5.2 Emergence of an Organization

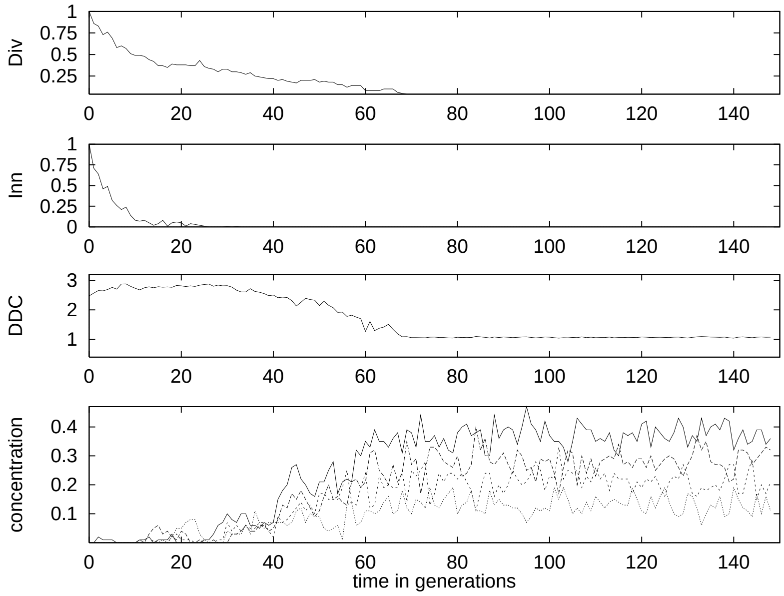
Figure 3 shows the emergence of a simple organization comprising four different string types, now in a simulation of the automata reaction in a soup of size $M = 100$. Diversity and innovativity drop continuously until a closed self-maintaining [6] organization dominates the soup with only a few distinct objects. Through drift the diversity of this organization is slowly reduced until an organization remains, which does not contain a closed, self-maintaining subset.

5.3 Exploration and innovation

If the soup size is increased to $M = 10^4$ a new qualitative phenomenon appears. Figure 4 shows a typical simulation of the automata reaction with instruction table 1 and without filter condition. After a quick drop of the diversity at the beginning (gen. 0-2) the system enters an **explorative phase** (gen. 2-40) which is characterized by high innovativity, productivity and diversity. Many new objects are generated constantly and there are no string types with exceptionally high concentrations. The high innovativity indicates that the sequence space is intensively explored.

The explorative phase comes to a sudden end, when string types appear (around gen. 40) that are replicating with every other string (active replicators), thus quickly dominating the soup. The appearance of active replicators may be called an innovation which leads – in this case – to an end of the explorative phase. The process that follows after gen. 40 is similar to the extinction process shown in Fig. 3. The system becomes simple and enters a stable phase where the organization structure is reduced only through undirected drift, with every string being an active replicator. The qualitative behavior is different from the AND reaction and the process leading to the final string is more gradual and continuous.

[6] A set $\mathcal{A}$ of object species is **closed** if every interaction within $\mathcal{A}$ produces only objects already in $\mathcal{A}$. A set $\mathcal{A}$ of object species is **self-maintaining** if every object is produced by at least one interaction within $\mathcal{A}$ [13].

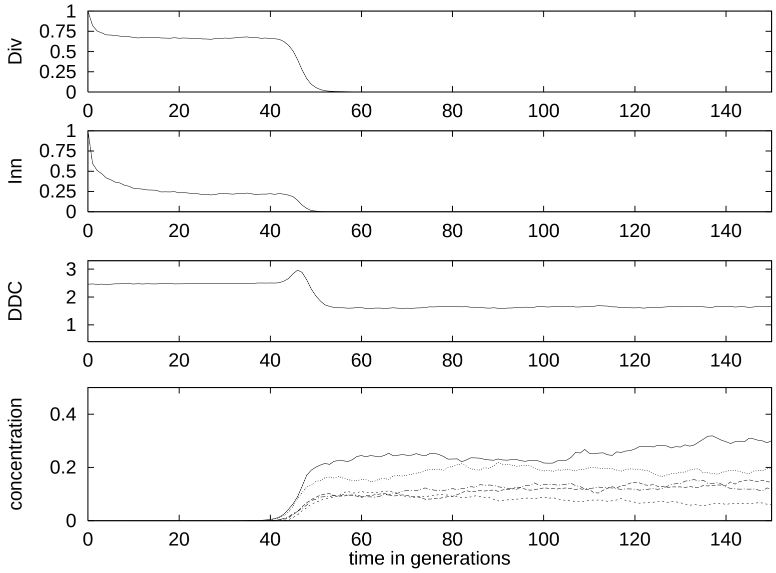


Reaction table at generation 140

No	String	0	1	2	3
0	7240a7ef	2	1	2	1
1	7240a7ea	0	0	0	0
2	7240a7eb	3	0	3	0
3	7240a7ee	1	1	1	1

String	Program
1111	AND
1110	ID
0111	CPOFF
1010	NOT
0000	ID
0100	TDIR
0010	SETP 0111
0111	

Figure 3: *Emergence of an organization. Parameters: Automata reaction with code table 1, $M = 100$. no filter condition, soup seeded with M random strings. For the four surviving strings the reaction table and their concentration over time is given.*



Reaction table at generation 280

No	String	0	1	2		7
0	1e1ca260	0	0	0		0
1	1e1ca261	1	1	1		1
2	1e1ca264	2	2	2		2
7	1011a261	7	7	7		7

String	Program
0000	ID
0100	CPON
0010	SETP 1010
1010	
1100	EXOR
0001	MOV
1110	ID
0001	MOV

Figure 4: *Exploration and innovation. During the explorative phase (generation 2 - 40) the diversity is constantly high and many totally new string types are produced. Parameters: soup size $M = 10^4$, automata reaction with table 1, no filter condition f, soup seeded with M random strings.*

5.4 Evolution

This section describes experiments which showed evolutionary behavior in the absence of external variation operators (i.e. mutation) and the absence of explicit selection[7] and fitness function. The following setup has been used for the experiments in this section:

(a) Soup size $M = 10^5$ or $M = 10^6$. These large soup sizes are used to get an explorative behavior and to reduce the effects of the non-deterministic components of the reactor algorithm.

(b) Elastic collisions introduced through the filter condition

$$f_1(s_1, s_2, s_3) = (s_1 \neq s_3 \wedge s_2 \neq s_3). \tag{3}$$

By using this filter condition the exact replication is disabled. Every collision that normally produces a string identical to one of the reactants is considered to be elastic [13].

(c) Reaction mechanism: automata reaction with code table 2. Code table 2 does not contain a NOT operation. Without a NOT operation it is harder to construct a self-maintaining organization of strings under filter condition f_1.

(d) The soup is seeded with random strings from $\{0,1\}^{32}$, resulting in an initial diversity approximately equal to one.

Figure 5 shows the short-time behavior[8]. As in Fig. 4 an explorative phase with very high diversity can be observed (generation $0 - 110$). A lot of completely new strings are produced. The productivity (fraction of collisions that are not elastic) is similar to the productivity of a soup of randomly generated strings.

This changes when a self-maintaining though not closed organization emerges (generation $110 - 200$). This does not happen as fast as in Fig. 4 . Here, the process takes a much longer time and seems to be more complex. Many different "species" are competing for space. Over a short period of time new strings are generated which increase their concentrations and are replaced by "better" ones. Figure 7 shows this early phase for some representative strings of a run with soup size $M = 10^6$.

Figure 6 shows the long time behavior. After the first early evolutionary phase (gen $110 - 200$) an organization has emerged whose gene diversity is massively reduced. It consists of approximately 50000 different string types where only about 3000 are present at a specific time. This organization still generates completely new strings but at a much lower frequency than the soup in the explorative phase (generation $0 - 110$).

The productivity in Fig. 6 (generation $500 - 7000$) shows, that even this (over a long period stable) organization is able to develop further. The character of this evolutionary process is different from that shown in Fig. 7 . There, evolution is taking place on the binary string (better: quasi-species) level. Here, the core set of strings is not changed. The organization is enhanced by adding/absorbing "useful" new strings which are "invented".

[7] The selection of two objects in the reactor algorithm is not selection in the sense of evolution theory. The random selection simulates only random collisions among objects.

[8] Although this is called "short-time behavior", a rather long time is shown compared to Fig. 2.

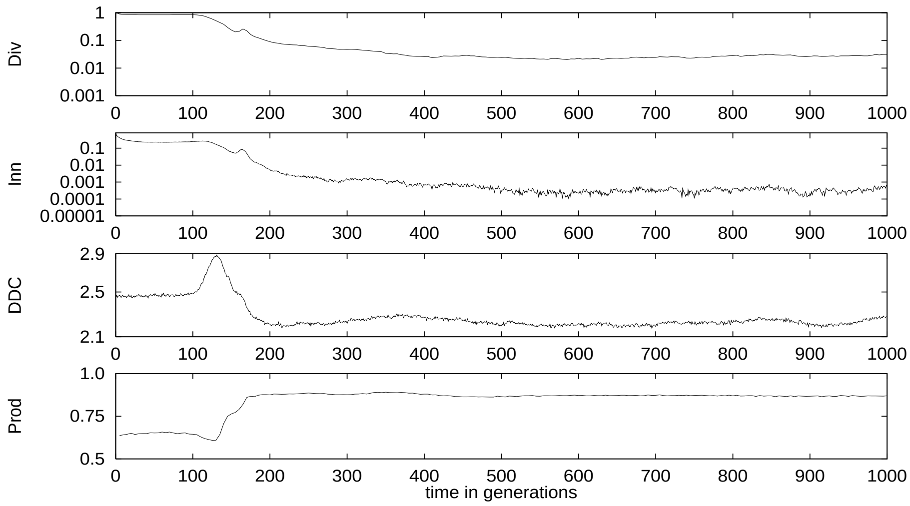


Figure 5: *Short-time evolutionary behavior. Parameters: soup size $M = 10^5$, automata reaction with table 2, filter condition f_1, soup seeded with M random strings.*

The phenomenon that long quasi-stable periods are interrupted by rapid changes is often referred to as punctuated equilibrium [11]. It has been observed in natural as well as in artificial systems (e.g. DNA world [28] and multi-agent systems [33]).

For a more detailed analysis of the organization structure the reaction table can be used. Figure 8 and 9 show a part of the reaction table of the 90 most frequent strings at different stages of the evolutionary process. They demonstrate how the coupling between strings becomes closer and the amount of elastic collisions is reduced.

An interesting detail is, that a cross-over mechanism has emerged. Figure 10 shows an example where 8 "cross-overing" strings form a closed organization. The organization is so small and closed, because the cross-over is performed at a specific point.

6 Discussion

Our experiments demonstrate that complex forms of evolution can take place in systems without any explicit variation operator, like mutation or recombination, and without any explicit (artificial) selection.

The setup used here is one of the simplest ways of letting machines acting on each other. We think that an important concept concerning natural evolution is that the way evolution is carried out is itself under the influence of evolution, a concept which is called **meta-evolution**. Meta-evolution has long been applied in evolutionary algorithms. For example, in evolution strategies (ES) parameters are used which determine the variance and covariance of a generalized n-dimensional normal distribution for mutations. The strategy parameters themselves are adapted during the optimization process. Only the operators used for adaptation are fixed. In our system, the variation of an object alters the way how other objects are modified, because the object will subsequently interact with others. We can interpret such a system as having many levels of meta-evolution, maybe a potentially infinite number of levels.

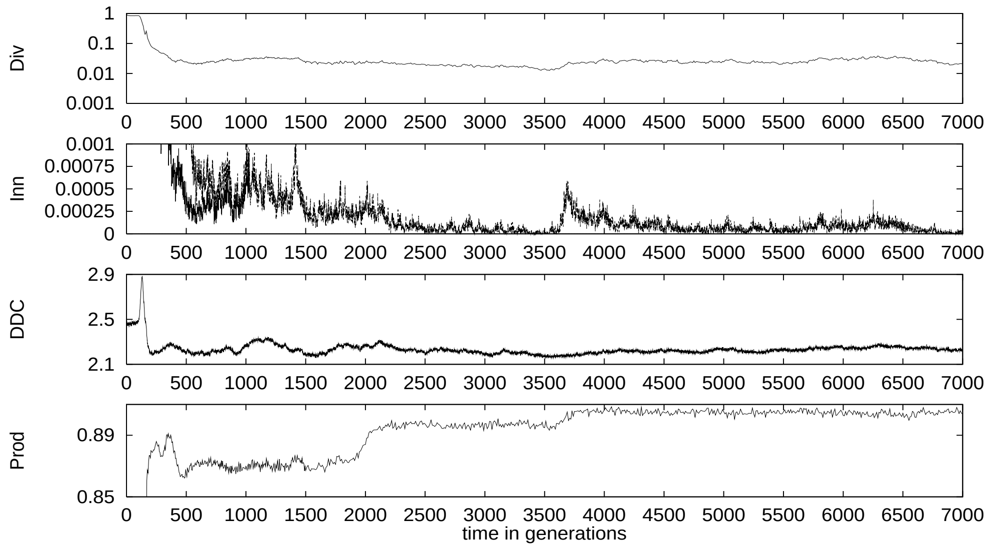


Figure 6: *Long-time evolutionary behavior. Parameters: soup size $M = 10^5$, automata reaction with table 2, filter condition f_1, soup seeded with M random strings. Same run as in figure 5*

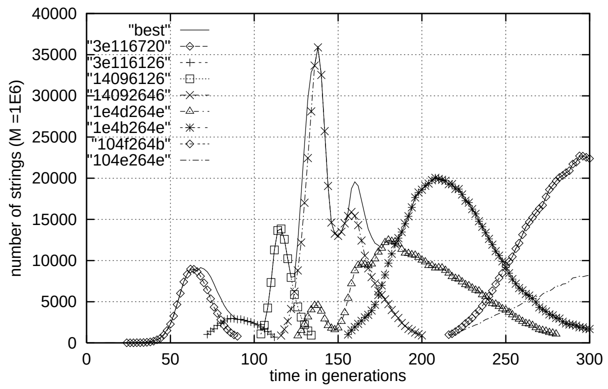


Figure 7: *Early evolutionary behavior. The concentration of some representative strings are shown. Parameters: soup size $M = 10^6$, automata reaction with table 2, filter condition f_1, soup seeded with M random strings.*

generation 75

No.	String	0	1	2	3	4	5	6	7	88	89
0	8140b6fa	29	*	*	*	*	*	*	*	*	*
1	140da6e8	-	-	-	-	-	-	-	-	-	-
2	a625ffb3	29	22	-	*	*	-	-	*	-	-
3	140ca6f2	-	14	-	-	14	14	14	14	14	14
4	77ae526e	-	*	*	-	-	*	*	-	*	*
5	c46b12a1	*	*	*	*	*	*	*	*	*	*
6	4a8c05c9	*	*	*	*	*	*	*	*	*	*
7	223be8ae	-	-	*	-	-	*	*	-	82	*
88	7ecbf629	*	*	*	*	*	*	*	*	*	*
89	70507121	*	-	*	*	*	-	-	*	-	-

generation 110

No.	String	0	1	2	3	4	5	6	7	88	89
0	c11a260b	-	-	-	-	-	-	-	-	-	-
1	011a2614	02	-	-	-	-	02	02	02	02	02
2	011a2615	-	-	-	01	01	-	-	-	-	-
3	c11a260e	-	-	-	-	-	-	-	-	-	-
4	c11a260a	-	00	-	00	00	-	-	-	-	-
5	8211e267	27	28	28	27	27	-	28	27	28	27
6	1211e267	23	-	-	23	23	23	-	23	-	23
7	d211e267	-	06	06	-	-	23	-	-	*	-
88	211a260d	49	-	-	49	49	49	-	49	-	49
89	c611e267	-	*	*	-	-	*	*	-	*	-

Figure 8: *Left:Reaction table during the explorative phase (generation 75, Fig. 5). The objects are loosely coupled. Many reaction pathways leave the reaction table (denoted by "*"). Right: Reaction table shortly after the explorative phase (generation 110, Fig. 5). The objects are coupled more closely. Still many reaction pathways leave the reaction table. "-" means an elastic collision (no reaction product).*

generation 3400

No.	String	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	67	68	69
0	104e264b	01	-	03	02	05	04	07	06	01	-	03	02	01	-	02	-	03	-	64	65	-
1	104e264a	-	00	03	02	05	04	07	06	-	00	03	02	-	00	02	00	03	00	64	65	00
2	104f264b	01	00	03	-	05	04	07	06	01	00	03	-	01	00	-	00	03	00	64	65	00
3	104f264a	01	00	-	02	05	04	07	06	01	00	-	02	01	00	02	00	-	00	64	65	00
4	140e264b	-	00	-	02	-	-	-	06	00	00	02	02	00	00	02	00	02	00	-	65	00
5	140e264a	01	-	03	-	-	-	07	-	01	01	03	03	01	01	03	01	03	01	64	-	01
6	140f264b	-	00	-	02	-	04	-	-	00	00	02	02	00	00	02	00	02	00	-	65	00
7	140f264a	01	-	03	-	05	-	-	-	01	01	03	03	01	01	03	01	03	01	64	-	01
8	104e246b	09	-	11	10	30	24	31	28	09	-	11	10	09	-	10	-	11	-	*	*	-
9	104e246a	-	08	11	10	30	24	31	28	-	08	11	10	-	08	10	08	11	08	*	*	08
10	104f246b	09	08	11	-	30	24	31	28	09	08	11	-	09	08	-	08	11	08	*	*	08
11	104f246a	09	08	-	10	30	24	31	28	09	08	-	10	09	08	10	08	-	08	*	*	08
12	104e206b	-	00	-	02	00	00	02	02	-	08	-	10	-	-	16	21	-	18	00	02	25
13	104e206a	01	-	03	-	01	01	03	03	09	-	11	-	-	-	-	-	14	-	01	03	23
14	104f206a	01	-	03	-	01	01	03	03	09	-	11	-	13	-	-	-	-	-	01	03	23
15	104e261a	01	-	03	-	01	01	03	03	09	-	11	-	13	-	-	-	14	-	01	03	23
16	104f206b	-	00	-	02	00	00	02	02	-	08	-	10	-	12	-	21	-	18	00	02	25
17	104e260a	01	-	03	-	01	01	03	03	09	-	11	-	13	-	-	-	14	-	01	03	23
67	1046264b	01	00	03	02	05	04	07	06	*	*	*	*	*	*	*	00	*	00	64	65	00
68	10af264a	59	59	-	-	59	59	-	-	*	*	*	*	*	*	*	*	*	*	59	-	*
69	104e265e	01	-	03	-	01	01	03	03	09	-	11	-	13	-	-	-	14	-	01	03	-

Figure 9: *Reaction table of a converged soup (generation 3400, Fig. 5). The objects are coupled very closely. Only a few reaction pathways are leaving the reaction table. The objects are very similar.*

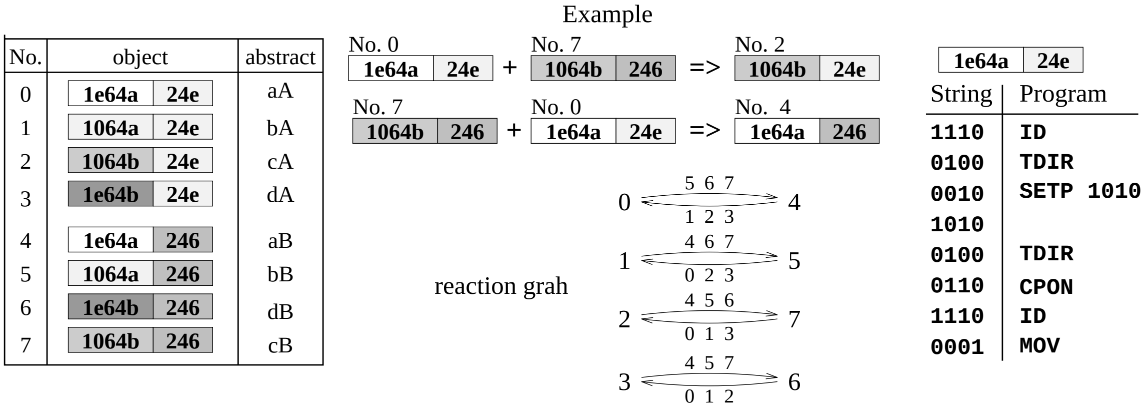


Figure 10: *Emergence of cross-over. The strings are able to insert a part of their sequence into the sequence of an arbitrary string. The result is a cross-over of both. E.g.* $1e64a24e + 00000000 \Longrightarrow 0000024e$.

Bagley, Farmer and Fontana have studied the emergence and evolution of autocatalytic metabolic networks by using a constructive artificial chemistry, where the molecules are character sequences and the reactions are concatenation and cleavage [2]. They have shown how small, spontaneous fluctuations can be amplified by an autocatalytic network, possibly leading to a modification of the entire network. A sequence of such modifications has been identified as an evolutionary process. The spontaneous fluctuations are explicitly induced by the meta-dynamical algorithm which can be interpreted as explicit mutation.

In contrast to meta-dynamical ODE approach in [2], we have used a discrete soup of objects. The diversity in the kinetic parameters is low. Only two different values for rate constants are needed (e.g. 0 and 1, see Eq. 2). There are only catalyzed reactions. No spontaneous reactions are needed to generate an evolutionary behavior. The evolutionary steps are not explicitly initiated by the simulation algorithm. The algorithm for simulating the kinetics is much simpler than the stochastic metadynamics proposed in [2].

Disadvantages of our approach are that (1) the number of molecules inside the soup is finite and bounded by the memory size, (2) reactions with rate constants that differ in several orders of magnitude can not be simulated easily, and (3) the computational effort is much higher in case of low diversity. Advantages are that (1) the number of different objects inside the soup can be very high (e.g. 10^6 here vs. 75 types in [2]), (2) the time evolution is not influenced or disturbed by the algorithm simulating the chemical dynamics, and (3) implementation is much simpler.

7 Summary and Future Work

We have extended our earlier work by introducing a reaction mechanism inspired by techniques of computer science It has been demonstrated, that the building blocks used in computer science to describe hardware elements and functions can be easily used to construct self-organizing and even evolving systems.

For exploring generalizations about evolving systems the high speed at which the reactions are carried out is helpful, because it allows interactive and large experiments. The disadvantages are the (so far) fixed length of strings and that the formalization of the reaction mechanism is more complicated than using lambda-calculus [13] or matrix multiplication [4].

The complex behavior shown here, results from a simple, ad hoc hand-written interaction mechanism working on small objects in an environment with no spatial structure. Introducing larger, variable-length objects and topological structures, allowing the formation of niches, will result in an increase in complexity. Then we have to face the danger, that in order to grasp a world we do not understand, we create a system we do not understand [6, 24]. To circumvent this, powerful analyzation and visualization methods are required, that should be general (like the DDC), so that they can be applied to different systems [9].

Reaction systems – like the system discussed here – are able to perform computations [5]. For example a chemical reaction system has been used to instantiate an artificial neural network [16] or to solve a variant of the Hamilton-path problem [1]. So far, the reaction pathways have been chosen carefully by hand. The integration of self-evolution into molecular computing systems is a promising goal for future research.

Acknowledgments

This project is supported by the DFG (Deutsche Forschungsgemeinschaft), grant Ba 1042/2-1 and Ba 1042/2-2. We also thank Helge Baier, Peter Nordin and Jochen Triesch for many stimulating discussions and Ulrich Hermes for his outstanding technical support.

APPENDIX

The Appendix contains a description for all registers and instructions of the automata reaction.

7.0.1 The control registers

In addition to the two 32 bit registers the automaton has 5 smaller **control registers**. The control registers are initialized with the first value given in the following description:

direction r_D — $r_D \in \{\text{left}, \text{right}\}$
The direction register determines the direction the pointers are moving. The operation TDIR toggles the direction.

move-mode r_M — $r_M \in \{\text{both}, \text{IOpointer}, \text{operator-pointer}\}$
This register controls which pointer is moved in case of a pointer movement. The operation TMM toggles the movement mode.

last-ALU-operation r_{ALU} — $r_{ALU} \in \{ID, NOT, AND, OR, EXOR, EQ\}$
This register stores the last ALU operation. If the copy mode is switched on, the operation stored in r_{ALU} will be executed before each movement step.

copy-mode r_C — $r_C \in \{\text{off}, \text{on}\}$
If the copy mode is switched on, the last ALU operation stored in r_{ALU} is executed before each movement step of the IO pointer and the operator-pointer.

pattern r_P — $r_P \in \{\text{none}\} \cup \{0,1\}^4$
With "SETP **p**" this register will be set to the pattern **p** ($\mathbf{p} \in \{0,1\}^4$). The operation UNSETP clears the pattern by setting r_P to none. If the pattern is set, each MOV operation will move the pointers unless the pattern is found in the operator register (max. 32 steps). If only the IO pointer is moved the pattern is searched in the IO register. If $r_P = \text{none}$ the pointers are moved one step.

7.0.2 Arithmetic and logic instructions

During an arithmetic or logic operation bit b and b' are processed by the ALU (Fig. 1). The result overwrites b'. Then the pointers are moved accordingly to the direction register r_D (left or right) and the the move-mode register r_M (both, IO pointer only or operator pointer only).

ID identity: $b' := b$, $r_{ALU} := ID$.

NOT negation: $b' := \neg b$, $r_{ALU} := NOT$.

AND logic and: $b' := b \wedge b'$, $r_{ALU} := AND$.

OR logic or: $b' := b \vee b'$, $r_{ALU} := OR$.

EXOR exclusive or: $b' := b \neq b'$, $r_{ALU} := EXOR$.

EQ equality: $b' := b = b'$, $r_{ALU} := EQ$.

After the execution of a logic instruction, register r_{ALU} is set to the operation executed. It is used only during the execution of the MOV instruction, if the copy mode is switched on ($r_C = on$).

7.0.3 Movement and control instructions

MOV — Moves the pointers accordingly to the move-mode, direction and pattern register. If no pattern is set, ($r_P = \text{none}$) the pointers are moved one step (one bit). If a pattern is set, the pointers are moved unless the pattern is found (max. 32 steps). If $r_M = \text{IOpointer}$ the pattern is searched in the IO register, otherwise in the operator register. Again, the direction of the movement is given by the direction register r_D.
If the copy-mode is switched on ($r_C := \text{on}$) by CPON, the last ALU operation is executed before each step.

CPON — Switches the copy-mode on ($r_C := \text{on}$).

CPOFF Switches the copy-mode off ($r_C :=$ off).

SETP **p** Sets the pattern $\mathbf{p} \in \{0,1\}^4$ for the move operation ($r_P := \mathbf{p}$). This is the only operation with an argument.

UNSETP Clears the pattern ($r_P :=$ none).

TDIR Toggles the move direction for the pointers.

$$r_D := \begin{cases} \text{right} & \text{if } r_D = \text{left}, \\ \text{left} & \text{if } r_D = \text{right} \end{cases}$$

TMM Toggles the move-mode:

$$r_M := \begin{cases} \text{both} & \text{if } r_M = \text{operator-pointer}\ , \\ \text{IOpointer} & \text{if } r_M = \text{both}, \\ \text{operator-pointer} & \text{if } r_M = \text{io-pointer} \end{cases}$$

NOP No operation.

STOP Stops the automaton before the whole program is executed.

For a precise formal specification of the automata reaction the source code is available [8].

References

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 1994.

[2] R. J. Bagley, J. D. Farmer, and W. Fontana. Evolution of a metabolism. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Proceedings of the Workshop on Artificial Life (ALIFE '90)*, volume 5 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 141–158, Redwood City, CA, USA, Feb. 1992. Addison-Wesley.

[3] W. Banzhaf. Self-organisation in a system of binary strings. In R. Brooks and P. Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Cambridge, MA, 1994. MIT Press.

[4] W. Banzhaf. Self-organizing algorithms derived from RNA interactions. In W. Banzhaf and F. H. Eeckman, editors, *Evolution and Biocomputing*, volume 899 of *Lecture Notes in Computer Science*, pages 69–103. Springer, Berlin, 1995.

[5] W. Banzhaf, P. Dittrich, and H. Rauhe. Emergent computation by catalytic reactions. *Nanotechnology*, 7(1):307–314, 1996.

[6] R. Boyd and P. J. Richerson. *Culture and the Evolutionary Process*. University of Chicago Press, 1985.

[7] A. W. Burks, H. H. Goldstine, and J. von Neumann. Preliminary discussion of the logical design of electronic computing instrumen. Technical report, Report to the U.S. Army Ordonance Dept., 1946.

[8] P. Dittrich. ANSI C source code for the automata reaction. Internet server of the Chair of Systems Analysis, Dept. of Computer Science, University of Dortmund, 1997.

[9] P. Dittrich, J. Ziegler, and W. Banzhaf. Mesoscopic analysis of self-evolution in an artificial chemistry. Proc. of "Artificial Life VI", Los Angeles, June 26-29, 1998; C. Adami, R. Belew, H. Kitano, and C. Taylor, eds., MIT Press, in print, 1998.

[10] M. Eigen. Selforganisation of matter and the evolution of biological macromolecules. *Naturwissenschaften*, 58:465–526, 1971.

[11] N. Eldridge and S. J. Gould. Punctuated equilibria: An alternative to phyletic gradualism. In T. M. Schopf, editor, *Models in Paleobiology*, pages 82–115. Freeman, Cooper a. Co, San Francisco, 1972.

[12] J. D. Farmer, S. A. Kauffman, and N. H. Packard. Autocatalytic replication of polymers. *Physica D*, 22:50–67, 1986.

[13] W. Fontana. Algorithmic chemistry. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II: proceedings of the second artificial life workshop*, pages 159–209. Addison-Wesley, Redwood City, CA, 1992.

[14] W. Fontana and L. W. Buss. The barrier of objects: From dynamical systems to bounded organization. In J. Casti and A. Karlqvist, editors, *Boundaries and Barriers*, pages 56–116. Addison-Wesley, 1996.

[15] K. Gödel. über formal unentscheidbare Sätze der principia mathematica und verwandter systeme, i. *Monatshefte fuer Mathematik und Physik*, 38:173–198, 1931.

[16] A. Hjelmfelt and J. Ross. Chemical implementation and thermodynamics of collective neural networks. *Proc. Natl. Acad. Sci. USA*, 89:388–391, 1992.

[17] J. Hofbauer and K. Sigmund. *Dynamical Systems and the Theory of Evolution.* University Press, Cambridge UK, 1988.

[18] D. R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid.* Basic Books,Inc., New York, 1979.

[19] T. Ikegami and T. Hashimoto. Coevolution of machines and tapes. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Proceedings of the Third European Conference on Artificial Life : Advances in Artificial Life*, volume 929 of *LNAI*, pages 234–245. Springer Verlag, Berlin, 1995.

[20] S. A. Kauffman. *The Origins of Order.* Oxford University Press, 1993.

[21] J. T. Kim. Using distance distribution to measure complexity of populations. In D. Stein and L. Nadel, editors, *1993 Lectures in Complex Systems.* Addison-Wesley, Redwood City, CA, 1995.

[22] C. G. Langton. Artificial life. In C. G. Langton, editor, *Artificial Life*, SFI Studies in the Science of Complexity, pages 1–45. Santa Fe Institute, Los Alamos, New Mexico, Addison-Wesley, Redwood City, CA, 1989.

[23] M. W. Lugowski. Computational metabolism: Towards biological geometries for computing. In C. G. Langton, editor, *Artificial Life*, pages 341–368. Addison-Wesley, Redwood City, CA, 1989.

[24] J. Maynard-Smith. Byte-sized evolution. *Nature*, 355:772–773, 1992.

[25] A. N. Pargellis. The spontanous generation of digital "life". *Physica D*, 91:86–96, 1996.

[26] S. Rasmussen, C. Knudsen, R. Feldberg, and M. Hindsholm. The coreworld: Emergence and evolution of cooperative structures in a computational chemistry. *Physica D*, 42:111–194, 1990.

[27] T. S. Ray. An approach to the synthesis of life. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II: proceedings of the second artificial life workshop*, pages 371–408. Addison-Wesley, Redwood City, CA, 1992.

[28] M. R. Shen, M. A. Batzer, and P. L. Deininger. Evolution of the master alu gene(s). *J. Mol. Evol.*, 33(4):311–320, Oct. 1991.

[29] P. F. Stadler, W. Fontana, and J. H. Miller. Random catalytic reaction networks. *Physica D*, 63:378–392, November 1993.

[30] M. Thürk. *Ein Modell zur Selbstorganisation von Automatenalgorithmen zum Studium molekularer Evolution*. PhD thesis, University of Jena, naturwissenschaftliche Fakultät, 1993.

[31] A. M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math. Soc. (2)*, 42:230–265, 1935.

[32] F. Varela. On being autonomous: The lessons of natural history for systems theory. In G. Klir, editor, *Applied General Systems Research*, pages 77–84, 1978.

[33] M. Youssefmir and B. A. Huberman. Resource contention in multiagent systems. In V. Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 398–403, San Francisco, CA, 1995. MIT Press.