Bio-inspired Computing Machines with Self-repair Mechanisms

André Stauffer, Daniel Mange, and Gianluca Tempesti

Ecole polytechnique fédérale de Lausanne (EPFL),
Logic Systems Laboratory, CH-1015 Lausanne, Switzerland
andre.stauffer@epfl.ch

Abstract. Developmental biology requires three principles of organization characteristic of living organisms: multicellular architecture, cellular division, and cellular differentiation. Implemented in silicon according to these principles, new computing machines become able to grow, to self-replicate, and to self-repair. The introduction of a new algorithm for cellular division, the so-called Tom Thumb algorithm, necessitates new self-repair mechanisms of structural configuration, functional configuration, microscopic cicatrization, and macroscopic regeneration. The details of these mechanisms constitutes the core of this paper.

1 Introduction

The *Embryonics* project (for *embryonic electronics*) [2] aims at creating radically new computing machines inspired by Nature and able to grow, to self-repair, and to self-replicate [4] [8] [1]. The embryonic development of living creatures fascinates engineers who dream of designing computing machines mimicking living organisms in a completely different environment, the two-dimensional world of silicon. Our Embryonics project aims at creating such machines which, starting from a one-dimensional blueprint, an "artificial genome", will be able to grow and give birth to computers endowed, as their living models, with original properties such as self-repair and self-replication. These properties are highly desirable for "extreme" applications of computer engineering (space exploration, radioactive environments, avionics, etc.) and, more importantly, are indispensable for the design of the future nanoscale electronic components whose characteristics will be very close to those of living organisms [3]. In conclusion, the challenge to be met is to make perfect systems out of imperfect components.

Borrowing three principles of organization (multicellular architecture, cellular division, and cellular differentiation) from living organisms, we have already shown [5] how it is possible to grow an artificial organism in silicon thanks to two algorithms: an algorithm for cellular differentiation, based on coordinate calculation, and an algorithm for cellular division, the Tom Thumb algorithm [6].

The goal of this paper is to perform the Tom Thumb algorithm many times so that self-repair could be introduced and make it possible the cicatrization (microscopic self-repair) and the regeneration (macroscopic self-repair) of the original organism. Starting with a very simple artificial organism with only three cells, the "SOS" acronym (for Save Our Souls), Section 2 will recall the definitions of cicatrization and regeneration. We will then introduce the growth processes, based on the Tom Thumb algorithm, leading to the structural configuration (Section 3) and functional configuration (Section 4), as well as to the cicatrization mechanism (Section 5) and regeneration mechanism (Section 6) of the artificial cell. A brief conclusion (Section 7) summarizes the whole procedure.

2 Cloning, Cicatrization and Regeneration

Even if our final goal is the development of complex machines, in order to illustrate the basic mechanisms of self-repair we shall use an extremely simplified example, the display of the acronym "SOS". The machine that displays the acronym can be considered as a one-dimensional artificial organism, *Acronymus elegans*, composed of four cells (Fig. 1a). Each cell is identified by a X coordinate, ranging from 1 to 4. For coordinate values $X = 1$ and $X = 3$, the cell should implement the S character, for $X = 2$, it should implement the O character, while for $X = 4$ it should implement a *spare* or *totipotent cell.* A totipotent cell (in this example, a cell capable of displaying either the S or the O character) comprises $4 \times 6 = 24$ molecules (Fig. 1b), 21 of which are invariant, one displays the S character, and two display the O character. An incrementer—an adder of one modulo 4—is embedded in the final organism; this incrementer implements the X coordinate calculation according to the following cycle: $X = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$.

The *self-replication* (or *cloning*) of a multicellular artificial organism (the "SOS" acronym, for example), i.e. the production of one or several copies of the original, rests on two assumptions: (1) There exists a sufficient number of spare totipotent cells in the array (at least four in our "SOS" example) to contain one copy of the additional organism. (2) The calculation of the coordinates produces a cycle $X = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ implying $X{+} = (X + 1) \bmod 4$. As the same pattern of coordinates produces the same pattern of genes (in our example, the same alphabetical pattern "SOS"), self-replication of an organism can be easily accomplished if the incrementer, embedded in each totipotent cell, counts modulo n, thus producing several occurrences of the basic pattern of coordinates. Given a sufficiently large space, the self-replication of the organism can be repeated for any number of specimens in the X and/or Y axes.

In order to implement the self-repair of the organism, we decided to use one or several spare cells to the right of the original organism (Fig. 1a). The existence of a fault is detected by a KILL signal (Fig. 1c) which is calculated in each cell by a built-in self-test mechanism realized at the molecular level (see below). The state KILL=1 identifies the faulty cell, and the entire column of all cells to which the faulty cell belongs is considered faulty and is deactivated (column $X = 2$ in Fig. 1a; in this simple example, the column of cells is reduced to a single cell). All the functions (X coordinate and gene) of the cells to the right of the column $X = 1$ are shifted by one column to the right. Obviously, this process requires as many spare columns to the right of the array as there are faulty cells or columns

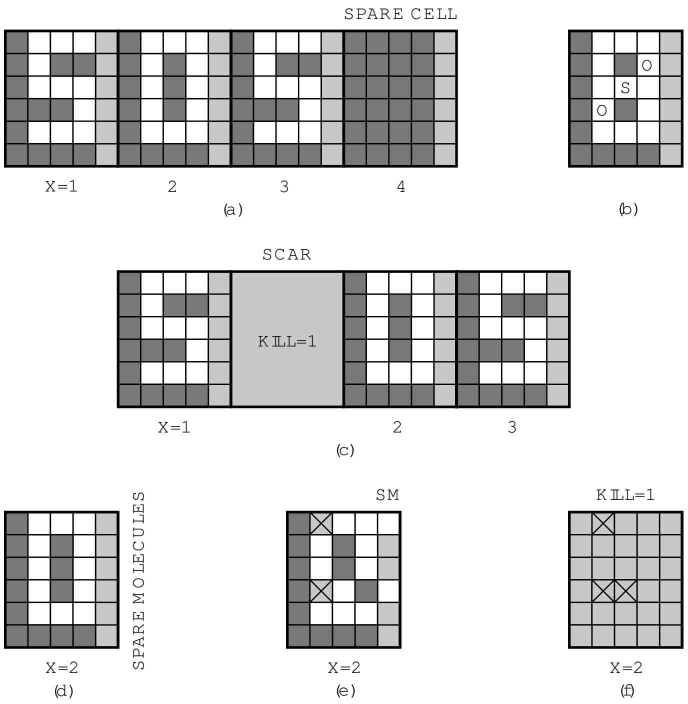

Fig. 1. Self-repair of *Acronymus elegans*, the acronym "SOS" (for Save Our Souls). (a) One-dimensional organism composed of four cells; a spare totipotent cell takes place at the right of the organism, with coordinate $X = 4$. (b) Totipotent cell. (c) The faulty cell ($X = 2$) and all the cells to the right of the faulty cell ($X = 3$) are shifted by one column to the right; the price to pay is an empty space, or a "scar". (d) Initial configuration of the healthy cell displaying O ($X = 2$). (e) Self-repaired cell ($X = 2$) with 2 faulty molecules; the price to pay is a deformation of the original O character; SM=spare molecule. (f) Faulty cell ($X = 2$) with 2 faulty molecules in the same row (KILL=1).

to repair (one spare column tolerating one faulty cell, in the example of Fig. 1a). It also implies that the cell needs to be able to bypass the faulty column and to divert to the right all the required signals (such as the artificial genome and the X coordinates). Given a sufficient number of cells, it is obviously possible to combine self-repair in the X direction and self-replication in both the X and Y directions.

The introduction in each cell of one or several columns of spare molecules (Fig. 1d), defined by a specific structural configuration, and the automatic detection of faulty molecules (by a built-in self-test mechanism which constantly compares two copies of the same molecule) allows cellular self-repair: each faulty molecule is deactivated, isolated from the network, and replaced by the nearest right molecule, which will itself be replaced by the nearest right molecule, and so on until a spare molecule (SM) is reached (Fig. 1e). The number of faulty molecules handled by the cellular self-repair mechanism is necessarily limited:

in the example of Fig. 1d, we tolerate at most one faulty molecule per row. If more than one molecule is faulty in one or more rows (Fig. 1f), cellular self-repair is impossible, in which case a global KILL=1 is generated to activate the organismic self-repair described above (Fig. 1c).

Salamanders, starfish, polyps and zebra fish can regenerate new heads, limbs, internal organs or other body parts if the originals are lost or damaged. The prolific properties of planarian worms make them an ideal starting point for investigating regeneration. A flatworm contains dormant stem cells distributed throughout its body. When damage occurs, stem cells near the injury rely on signals from neighboring damaged tissues to work out their location, and hence what repairs are needed [7]. According to Wolpert [10], regeneration is closely related to embryonic development; many aspects of regeneration seem related to embryonic regulation and can be considered in terms of replacing the positional values of cells that have been lost. Coming back to the Embryonics project, one may consider that the first step of repair, i.e. cellular self-repair where faulty molecules are replaced by spare molecules, can be regarded as a kind of *artificial cicatrization*, while the second step of repair, i.e. organismic self-repair where faulty cells are replaced by spare totipotent cells, is the equivalent of *artificial regeneration*.

3 Structural Configuration

For a better understanding of the *structural growth process*, performing the *structural configuration* by applying the Tom Thumb algorithm, we will give up the "SOS" acronym example, yet too complex, and we will start with a minimal self-repair cell. This cell is made up of six molecules organized as an array of two rows by three columns, one column (two molecules) being dedicated to self-repair (SM=spare molecule) (Fig. 2a). Each molecule is now able to store in its five memory positions five characters of the artificial genome, and the whole cell embeds 30 such characters. The structural genome G1 for the minimal self-repair cell is organized as a string of twelve characters, i.e. two characters for each molecule in the cell, moving anticlockwise by one character at each time step ($t = 0, 1, 2, ...$). The characters composing the alphabet of our structural genome are detailed in Fig. 3a. They are either *empty data*, *flag data* (from "north connect flag" to "north connect and branch activate flag") or *structural data*; structural data combine two pieces of information: a position information or *type* (from "top type" to "top-left type") and a state information or *mode* (from "living mode" to "dead mode"). Flag data will be used for constructing the various paths between the molecules, while structural data are indispensable, in a first step, for locating the *living* (normal) and *spare* molecules. Furthermore, each character is given a status and will eventually be *mobile data*, indefinitely moving around the cell, or *fixed data*, definitely trapped in a memory position of a molecule (Fig. 3b).

At each time step, a character of the structural genome is shifted from right to left and simultaneously stored in the lower leftmost molecule (Fig. 2a). The

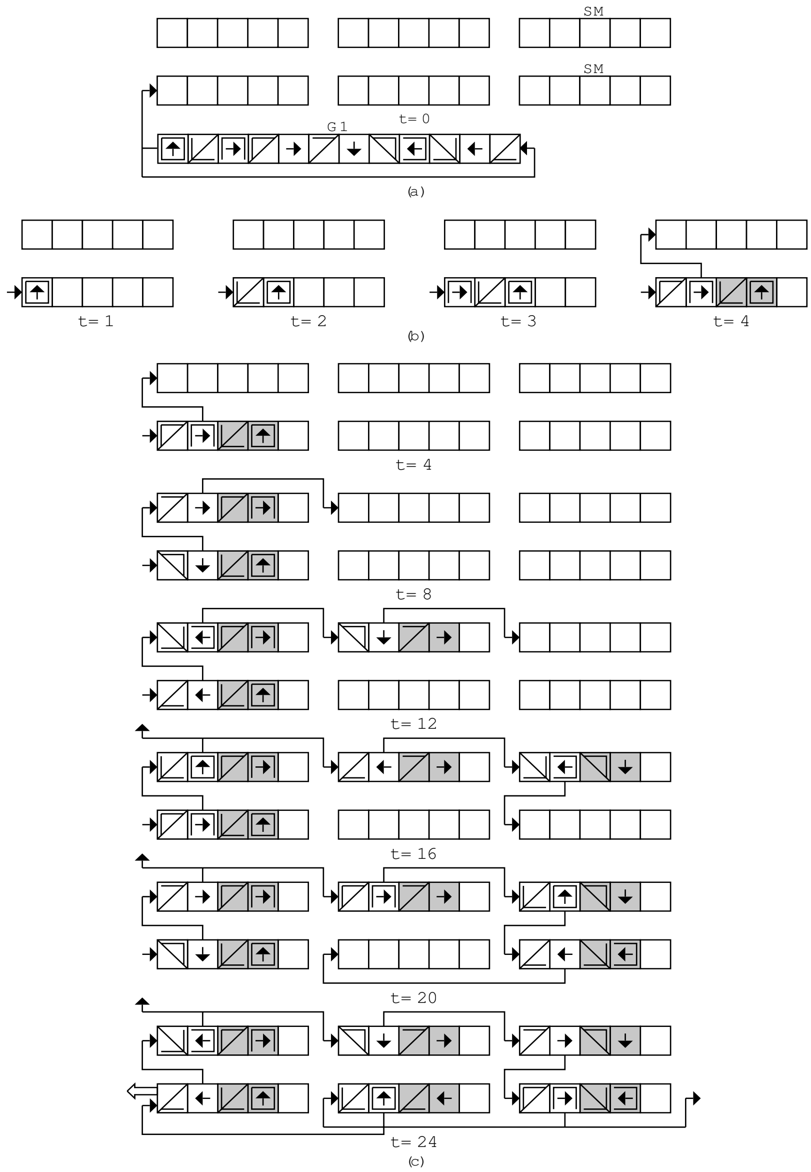

Fig. 2. The minimal self-repair cell made up of six molecules. (a) At time $t = 0$, the cell is empty; the structural genome G1 is ready to enter the first molecule. (b) Four steps of the Tom Thumb algorithm; $t = 1...3$: shift of the genome by one position to the right; $t = 4$: the genome is shifted by one position to the right, while the two rightmost characters are trapped in the molecule (fixed data in gray) and a connection is established to the north. (c) State of the connections between the molecules after every four steps of the structural growth process; at time $t = 24$, when the path is closed, the lowermost molecule of the first column delivers a close signal to the nearest left neighbor cell.

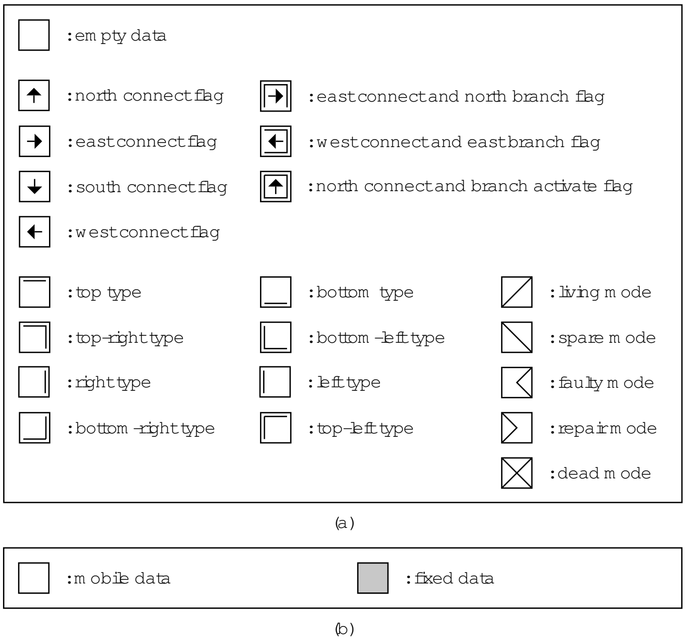

Fig. 3. The characters forming the alphabet of an artificial genome. (a) Graphical representations of the characters which are divided in 3 major classes: empty data, flag data describing the paths between the molecules of the cell, and structural data describing both the position (type) and the state (mode) of each molecule. (b) Graphical representation of the status of each character defining mobile or fixed data.

construction of the cell, i.e. storing the fixed data and defining the paths for mobile data, depends on two major patterns (Fig. 2b): (1) If the five, four or three rightmost memory positions of a molecule are empty (blank squares), the characters are shifted by one position to the right ($t = 0, 1, 2$). (2) If the two rightmost memory positions are empty ($t = 3$), the characters are shifted by one position to the right; in this situation, the two rightmost characters are trapped in the molecule (fixed data), and a new connection is established from the second leftmost position toward the northern, eastern, southern or western molecule, depending on the fixed flag information (for $t = 3$: “north connect and branch activate flag”, and the connection is toward the northern molecule). At time $t = 24$, the connection path between the molecules is closed and 24 characters, i.e. twice the contents of the structural genome, have been stored in 24 memory positions of the cell (Fig. 2c). The lowermost molecule of the first column delivers a *close signal* to the nearest left neighbor cell, while twelve characters are fixed data, defining the structure of the final cell, and the twelve remaining ones are mobile data, composing a copy of the structural genome. Mobile data are ready for starting the growth of other cells, in both northern and eastern directions.

In order to grow an artificial organism in both horizontal and vertical directions, the mother cell should be able to trigger the construction of two daughter cells, northward and eastward. At time $t = 15$ (Fig. 4a), we observe a pattern of characters which is able to start the construction of the northward daughter cell; the upper leftmost molecule is characterized by two specific flags, i.e. a fixed flag indicating a north branch and the branch activation flag. The new path to the northward daughter cell will start from the second leftmost memory position, at time $t = 16$. At time $t = 21$, another particular pattern of characters will start the construction of the eastward daughter cell; the lower rightmost molecule is characterized by two specific flags, i.e. a fixed flag indicating an east branch, and the branch activation flag (Fig. 4b). The new path to the eastward daughter cell will start from the second leftmost memory position at time $t = 22$.

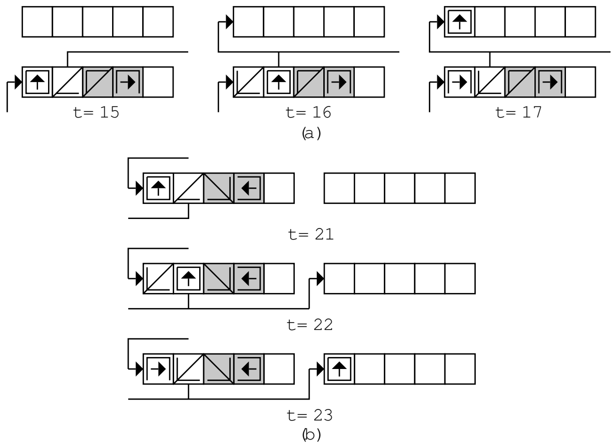

Fig. 4. Artificial cell division. (a) North directed branch starting the growth of a northward neighboring cell. (b) East directed branch starting the growth of an eastward neighboring cell.

The final cell shown in Fig. 2c will now start a *load process*, triggered by the close signal delivered by its nearest right neighbor cell ($t = i$), when this has finished its own structural growth process. A *load signal* will then propagate east-west and bottom-up according to Fig. 5 ($t = i+1$ to $i+3$). At each step, this signal will transform the corresponding molecule in two ways: (1) The structural data in the third position are shifted in the fifth, i.e. rightmost position, of the molecule memory. (2) The four leftmost memory positions are cleared (i.e. empty data). At the end of this process ($t = i + 4$), we finally obtain an homogeneous tissue of molecules, with four memory positions empty, and the fifth filled with the structural data, defining both the boundaries of the cell and the position of its living and spare molecules. This tissue is ready for being configured by a second artificial genome, the functional genome.

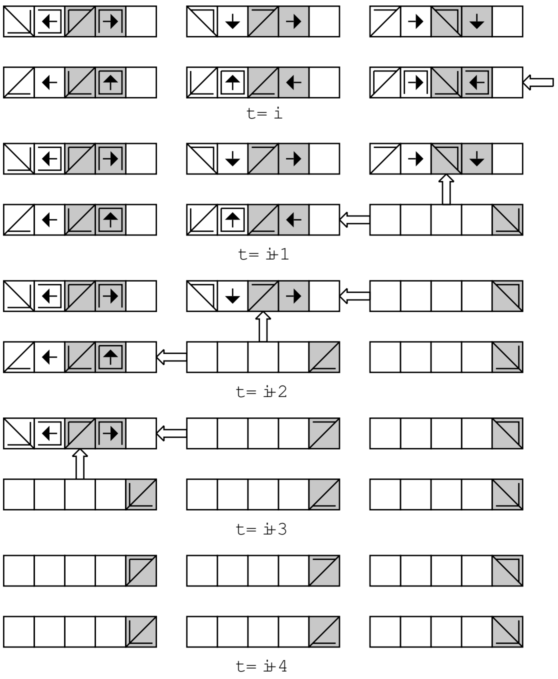

Fig. 5. Triggered by the the close signal of the nearest right neighbor cell ($t = i$), the load process stores the molecular types and modes of the artificial cell in the rightmost memory position of the molecules while the four leftmost ones are cleared ($t = i + 1$ to $i + 4$)

4 Functional Configuration

The goal of *functional configuration* is to store in the homogeneous tissue which already contains structural data (Fig. 5, $t = i + 4$) the functional data needed by the specifications of the current application. This configuration is a *functional growth process*, performed by applying the Tom Thumb algorithm with the following conditions: only the molecules in the "living mode" participate to the growth process, while the molecules in the "spare mode" are simply bypassed. The final cell is made up of four "living mode" molecules organized as an array of two rows by two columns (Fig. 6b, $t = 16$), while one row of two "spare mode" molecules are bypassed. Each molecule is now able to store in its four empty memory positions four characters of the functional genome G2 (Fig. 6a), which is implemented as a string of eight characters. These characters are either *empty data*, *flag data* (Fig. 3a) or *functional data*; functional data are indispensable for defining the final specifications of the cell under construction.

The Tom Thumb algorithm is executed according to the rules of structural configuration (Section 3) if we consider that the fifth position of each molecule is empty. At time $t = 16$ (Fig. 6b), 16 characters, i.e. twice the contents of the

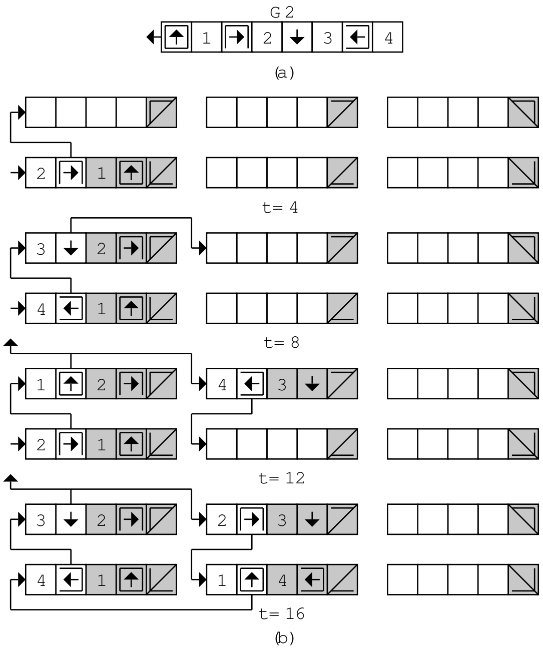

Fig. 6. Functional configuration of the "living mode" molecules. (a) Artificial genome G2. (b) State of the functional growth process after every four steps.

functional genome G2, have been stored in the 16 memory positions of the "living mode" molecules of the cell. Eight characters are fixed data, forming the phenotype of the final cell, and the eight remaining ones are mobile data, composing a copy of the original genome G2, i.e. the genotype. Both *translation* (i.e. construction of the cell) and *transcription* (i.e. copy of the genetic information) have been therefore achieved.

5 Cicatrization Mechanism

Fig. 6b, at time $t = 16$, shows the normal behavior of a healthy minimal cell, i.e. a cell without any faulty molecule.

A molecule is considered as *faulty*, or in the "faulty mode" (Fig. 3a), if some built-in self-test, not described in this paper, detects a lethal malfunction. Starting with the normal behavior of Fig. 6b ($t = 16$), we suppose that two molecules are suddenly in the "faulty mode" (Fig. 7, $t = i$): (1) The lowermost molecule in the first column, which was in previously in the "living mode". (2) The uppermost molecule in the third column, which was before in the "spare mode". While there is no change for the uppermost molecule, which is just no more able to play the role of a "living mode" molecule, the lowermost one triggers the following *cicatrization mechanism*, made up of a *repair process* followed by a *reset process* (Fig. 7).

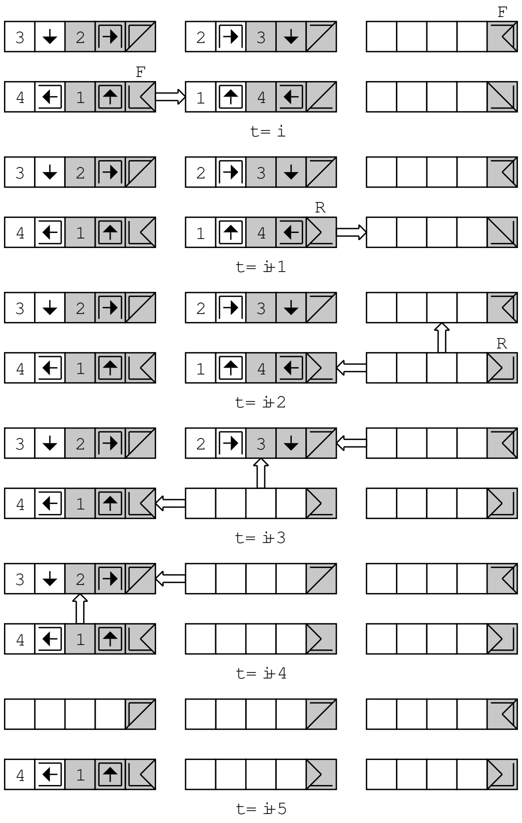

Fig. 7. Cicatrization mechanism performed as a repair process ($t = i + 1$ and $i + 2$) followed by a reset process ($t = i + 3$ to $i + 5$); at the start ($t = i$), two molecules are supposed in the "faulty mode" (F); two molecules are given the "repair mode" (R) at time $t = i + 1$ and $i + 2$

- In a first step, a *repair signal* sent by the lowermost cell transforms the first nearest right "living mode" neighbor in the "repair mode" ($t = i + 1$).
- In a second step, this signal continues to propagate eastward and transforms the next nearest right "spare mode" neighbor in the "repair mode" ($t = i{+}2$).
- The repair signal having converted a "spare mode" molecule into a "repair mode" one, a *reset signal* comes back in the opposite direction in order to clear the four first memory positions of each "living mode" molecule ($t = i{+}3$ and $i + 4$).
- In the last step ($t = i{+}5$), the reset signal erases the contents of the memory positions of the last molecule, the uppermost molecule in the first column.

In this mechanism, the contents of the four memory positions of any "faulty mode" molecule is not erased, as these molecules are obviously not able to perform any function.

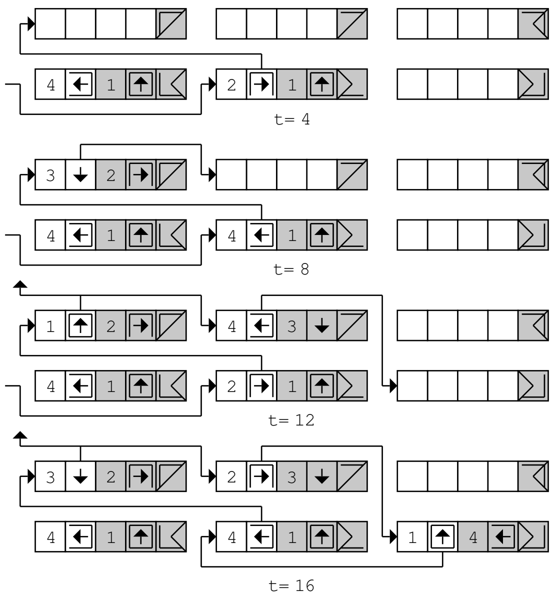

Fig. 8. Functional reconfiguration, based on the artificial genome G2, following the cicatrization mechanism; state of the functional growth process after every four steps

We finally obtain a new topological array containing four healthy molecules ("living mode" and "repair mode"), which is able to be reconfigured in order to emulate the original minimal cell, in its rectangular implementation (Fig. 6b, $t = 16$). A functional reconfiguration, exactly similar to that of Section 4, is launched. The preceding functional genome G2 (Fig. 6a) is introduced into the trapezoidal array of Fig. 7 ($t = i + 5$) and produces a new version of the original minimal cell (Fig. 8, $t = 16$).

6 Regeneration Mechanism

In the general discussion about cicatrization (Section 2), we have already pointed out that only one faulty molecule was tolerated between two spare molecules. A second faulty molecule in the same row will trigger the death of the whole cell, and the start of a regeneration mechanism with recalculation of the X coordinate. Fig. 9 ($t = i$) illustrates such a case: the cicatrized cell of Fig. 8 ($t = 16$) is given a new faulty molecule ("faulty mode") in its uppermost molecule of the second column. The *regeneration mechanism*, made up of a *repair process* and a *kill process*, takes place as follows (Fig. 9):

– In a first step, the new faulty cell sends a *repair signal* eastward, in order to look for a "spare mode" molecule, able to replace it ($t = i$).
– In a second step, the supposed "spare mode" molecule, which is in fact a "faulty mode" one, enters the lethal "dead mode" and triggers *kill signals* westward and southward ($t = i + 1$).

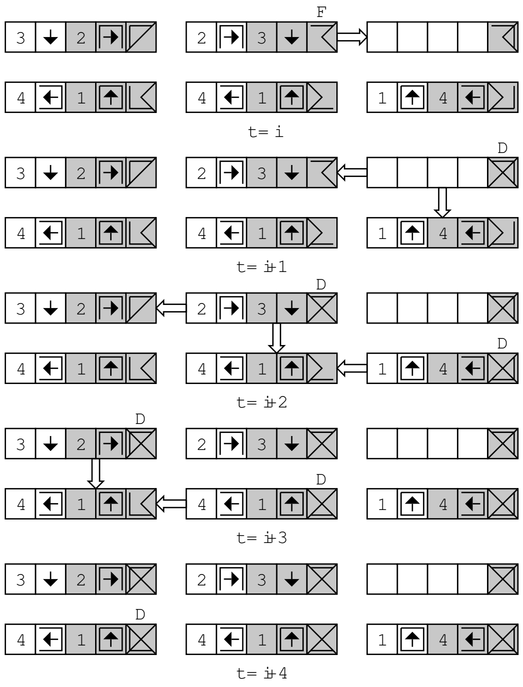

Fig. 9. Regeneration mechanism performed as a repair process ($t = i + 1$) followed by a kill process ($t = i + 2$ to $i + 4$); at the start ($t = i$), a new molecule is supposed in the "faulty mode" (F); all the molecules are then successively given the "dead mode" (D) from time $t = i + 1$ to $i + 4$

- In the three next steps, the other molecules of the array are given the "dead mode" ($t = i + 5$); the original minimal cell is dead, and a general KILL=1 signal is emitted so that the recalculation of the coordinates might take place.

7 Conclusion

While the Tom Thumb algorithm is a rather straightforward tool for designing self-repairing patterns of any complexity, its implementation in systems which are able to self-repair, involves a succession of four possible mechanisms:

- First, a structural configuration, defined by an artificial genome G1, is injected in a homogeneous array of Tom Thumb cellular automata in order to fix the boundaries of the cell ("top type", "bottom type", etc.) and the state of each molecule ("living mode" or "spare mode").
- Second, a functional configuration, described by an artificial genome G2, is injected in the sole "living mode" molecules of the previously defined array; the artificial organism under construction is then obtained.

– Third, a minor fault, one molecule in the “faulty mode”, is detected between two “spare mode” molecules; the cicatrization mechanism modifies the original array in order to replace the “faulty mode” molecule by a “repair mode” one; a reconfiguration process with the functional genome G2 concludes this mechanism.
– Fourth, a major fault may happen, with two molecules in the “faulty mode” between two “spare mode” molecules; the cell is no more able to self-repair, and the regeneration mechanism kills all the molecules in the cell, before generating a global KILL=1 signal triggering the recalculation of the X coordinate.

The detailed architecture for the data and signals cellular automaton (DSCA) which constitutes the basic molecule of the Tom Thumb algorithm has been described in [6], while the modifications of this DSCA for performing the self-repair mechanisms are presented in [9].

References

1. R. Canham and A. M. Tyrrell. An embryonic array with improved efficiency and fault tolerance. In J. Lohn et al., editor, *Proceedings of the NASA/DoD Conference on Evolvable Hardware (EH'03)*, pages 265–272. IEEE Computer Society, Los Alamitos, CA, 2003.
2. H. de Garis. Evolvable hardware: Genetic programming of a Darwin machine. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 441–449. Springer-Verlag, Heidelberg, 1993.
3. J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: opportunities for nanotechnology. *Science*, 280(5370):1716–1721, June 1998.
4. D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The Embryonics approach. *Proceedings of the IEEE*, 88(4):516–541, April 2000.
5. D. Mange, A. Stauffer, E. Petraglio, and G. Tempesti. Embryonics machines that divide and differentiate. In A.J. Ijspert, D. Mange, M. Murata, and S. Nishio, editors, *Biologically Inspired Approaches to Advanced Information Technology.* Proceedings of The First International Workshop Bio-ADIT 2004, Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2004.
6. D. Mange, A. Stauffer, E. Petraglio, and G. Tempesti. Self-replicating loop with universal construction. *Physica D*, 191(1-2):178–192, April 2004.
7. H. Pearson. The regeneration gap. Nature, 414(6):388–390, 2001.
8. L. Prodan, M. Udrescu, and M. Vladutin. Survivability of embryonic memories: Analysis and design principles. In R. S. Zebulum et al, editor, Proceedings of the NASA/DoD Conference on Evolvable Hardware (EH'04), pages 130–137. IEEE Computer Society, Los Alamitos, CA, 2004.
9. A. Stauffer, D. Mange, and G. Tempesti. Embryonic machines that grow, self- replicate and self-repair. In J. Lohn et al, editor, Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware (EH'05), pages 290–293. IEEE Computer Society, Los Alamitos, CA, 2005.
10. L. Wolpert. The Triumph of the Embryo. Oxford University Press, Oxford, 1993.