A Self-Replicating Universal Turing Machine: From von Neumann's Dream to New Embryonic Circuits

Héctor Fabio Restrepo, Daniel Mange, and **Moshe Sipper**

Logic Systems Laboratory, Swiss Federal Institute of Technology
CH - 1015 Lausanne, Switzerland
E-mail: {name.surname}@epfl.ch

Abstract

Borrowing inspiration from von Neumann's dream and the idea of a true cellular automaton, we describe a multicellular universal Turing machine implementation with self-replication and self-repair capabilities. This implementation was made possible thanks to a new "multicellular" automaton developed as part of the Embryonics (embryonic electronics) project. This new automaton, in which every artificial cell contains a complete copy of the genome, is endowed with self-replication and self-repair capabilities. With these properties and by using a modified version of the W-machine, it was possible to realize the mapping of the universal Turing machine onto our multicellular array.

Keywords: self-replication, self-repair, universal Turing machine, cellular automata, Embryonics.

Introduction

The Embryonics (embryonic electronics) project is inspired by the basic processes of molecular biology and by the embryonic development of living beings. By adopting three fundamental features of biology — multicellular organization, cellular division, and cellular differentiation — and by transposing them onto the two-dimensional world of integrated circuits in silicon, we show that properties of the living world, such as self-replication and self-repair, can also be attained in artificial objects (integrated circuits).

Our goal in this paper is to present self-replicating machines exhibiting universal computation, i.e., universal Turing machines. We demonstrate that the dream of von Neumann, the self-replication of such a machine, can be realized in actual hardware thanks to the Embryonics architecture.

In the next section we present a brief reminder of specialized and universal Turing machines. We then survey classical self-replicating automata and loops. The following section introduces the Embryonics architecture based on a multicellular array of cells and describes the implementation of a self-replicating specialized Turing machine. We next present the architecture of an ideal and of an actual universal Turing machine able to self-replicate. A discussion of our results follows in the final section.

Turing machines

In the 1930's, before the advent of digital computers, several mathematicians began to think about what it means to be able to compute a function. The theory of Turing machines was the response to this question. It is important to mention that Alonzo Church and Alan Turing independently arrived at equivalent conclusions; their common definition was: *A function is computable if it can be computed by a Turing machine.*

Turing machines were conceived by Alan Turing in his historic paper, "*On Computable Numbers, with an Application to the Entscheidungsproblem*" [22], which was his response to the Entscheidungsproblem, posed by the German mathematician David Hilbert. Hilbert asked if there existed, in principle, any definite method which could be applied to determine the truth of any mathematical question.

Turing machines are one of the earliest and most intuitive ways to render precise the notion of effective computability. This is now the foundation of the modern theory of computation and computability.

Specialized Turing machines

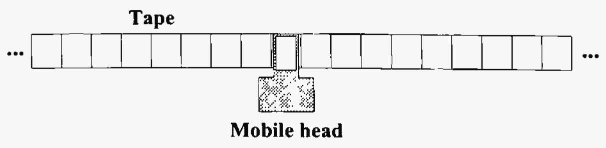

Figure 1: *A specialized Turing machine.*

A specialized Turing machine (Figure 1) is a finite-state machine (*the program*) controlling a mobile head, which operates on a tape (*the data*). The tape, composed of a sequence of squares, contains a string of symbols. The head is situated, at each moment, on some square of the tape and has to carry out three operations to complete a step of the computation. These operations are:

1. reading the square of the tape being scanned;
2. writing on the scanned square;
3. moving the head to an adjacent square.

A Turing machine can be described by three functions f_1, f_2, f_3:

$$Q^+ = f_1(Q, S) \quad (1)$$

$$S^+ = f_2(Q, S) \quad (2)$$

$$D^+ = f_3(Q, S) \quad (3)$$

where Q and S are, respectively, the current internal state and the current input symbol, and where Q^+, S^+, and D^+ are, respectively, the next internal state, the next input symbol, and the direction of the head's next move [13].

The universal Turing machine

Turing had the further idea of the universal Turing machine (UTM), capable of simulating the operation of any specialized Turing machine, and gave an exact description of such a UTM in his paper [22].

A universal Turing machine, U, is a Turing machine with the property of being able to read the description (on its tape) of any other Turing machine, T, and to carry out correctly (one step at a time) what T would have done. The necessary components of the machine U are a finite-state machine (the program of U) controlling a mobile head, which operates on a tape; the data on the tape describe completely the machine T to be simulated (the data of T and the program of T, i.e., the three functions Q^+, S^+, and D^+ describing T).

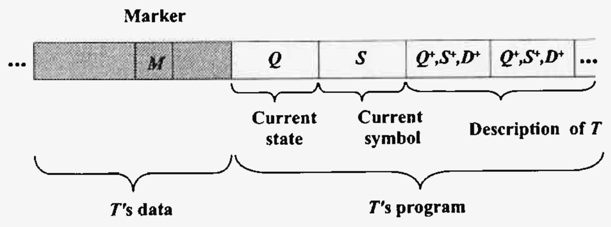

Figure 2: *Universal Turing machine's tape, describing specialized machine T.*

Figure 2 shows the organization of U's tape. To the left is a semi-infinite region, which contains the data of T's tape. Somewhere in this region is a marker M indicating where T's head is currently located. The next region contains the current internal state Q and the current input symbol S of T. The third region is used to record the description of T, i.e., the three functions Q^+, S^+, and D^+ for each combination of Q and S.

Self-replication: A brief survey

Self-replicating automata

The early history of the theory of self-replicating machines is basically the history of von Neumann's thinking on the matter [5, 18, 23]. Von Neumann's cellular automaton [23], as well as all the machines described in this paper, is based on the following general hypotheses:

- the automaton deals exclusively with the flow of information; the physical material (usually a silicon substrate) and the energy (power supply) are given a priori;
- the physical space is two-dimensional and as large as desired;
- the physical space is *homogeneous*, that is comprised of identical *molecules*, all of which have the same internal architecture and the same connections with their neighbors; only the *state* of a molecule (the combination of the values in its memories) and its position can distinguish it from its neighbors;
- replication is considered as a special case of growth: this process involves the creation of an identical organism by duplicating the genetic material of a mother entity onto a daughter one, thereby creating an exact clone.

To avoid conflicts with biological definitions, we do not use the term "cell" to indicate the parts of a cellular automaton, opting rather for the term "molecule". (In biological terms, a "cell" can be defined as the smallest part of a living being which carries the complete blueprint of the being, that is the being's *genome.*)

The molecule of von Neumann's automaton is a finite-state machine with 29 states. The future state of a molecule depends on the present state of the molecule itself and of its four cardinal neighbors (north, east, south, west). The exhaustive definition of the future state, the *transition table*, thus contains $29^5 = 20,511,149$ lines.

In his historic work [23], von Neumann showed that a possible *configuration* (a set of molecules in a given state) of his automaton can implement a universal constructor (Uconst) endowed with the following three properties:

1. universal construction (Figure 3);
2. self-replication of the universal constructor (Figure 4);
3. self-replication of a universal computer (Ucomp), i.e., a universal Turing machine (Figure 5).

According to the biological definition of a cell, it can be stated that von Neumann's automaton is a unicellular organism: its genome is composed of the description of the universal constructor and computer D(Uconst + Ucomp) written in the memory M (Figure 5); as each molecule of this description needs five molecules of the genome [23], it can be estimated that the genome is composed of approximately five times the number of molecules of the

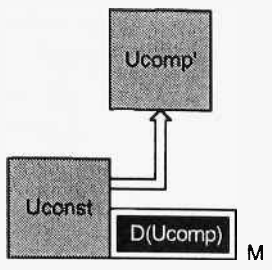

Figure 3: *Universal construction of von Neumann's automaton: a possible configuration can implement a universal constructor Uconst. Then, given the description D(Ucomp) of any one machine Ucomp, including a universal Turing machine, the universal constructor can build a specimen of this machine (Ucomp') in the molecular space.*

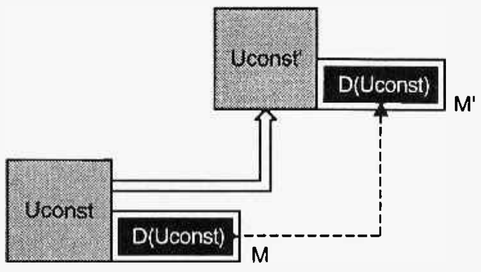

Figure 4: *Self-replication of the universal constructor: given the description D(Uconst) of the constructor itself, it is then possible to build a copy of the constructor in the molecular space: the constructor interprets first the description D(Uconst) to build a copy Uconst' whose memory M' is empty (translation process), and then copies the description D(Uconst) from the original memory M to the new memory M' (transcription process).*

universal constructor and computer.

In summary:

- the dimensions of von Neumann's automaton are substantial (on the order of 200,000 molecules) [6]; it has thus never been physically implemented and has been simulated only partially [4,16,17];
- the automaton implements the self-replication of a universal computer (a universal Turing machine).

Though von Neumann and his successors Burks [3,23], Thatcher [3], Lee [8], Codd [4], Banks [2], and Nourai and Kashef [14] demonstrated the theoretical possibility of realizing self-replicating automata with universal calculation, a practical implementation requires a markedly different approach. It was finally Langton, in 1984, who initiated a second stage in self-replication research.

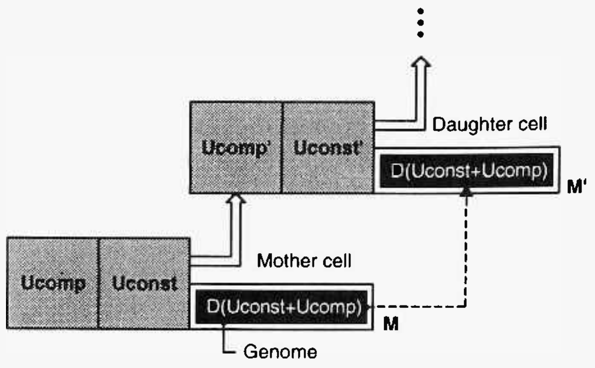

Figure 5: *Self-replication of a universal computer: by attaching to the constructor a universal computer Ucomp (a universal Turing machine), and by placing the description D(Uconst + Ucomp) in the original memory M, the universal constructor produces a copy of itself (Uconst') and a copy of the universal computer (Ucomp') through the mechanism described in Figure 4 (interpretation and then duplication of the description D).*

Self-replicating loops

In order to construct a self-replicating automaton simpler than von Neumann's, Langton [7] adopted more liberal criteria. He dropped the condition that the self-replicating unit must be capable of universal construction and computation. Langton's mechanism is based on an extremely simple configuration in Codd's automaton [4] called the periodic emitter, itself derived from the periodic pulser organ in von Neumann's automaton [23]. The *molecule* of Langton's automaton is a finite state machine with only 8 states. The future state, as with von Neumann's automaton, depends on the present state of the molecule itself and its four cardinal neighbors. The exhaustive definition of the future state, the transition table, contains only 219 lines, a very small subset of the theoretically possible $8^5 = 262,144$ lines (thanks to the use of default rules and symmetry assumptions).

Langton proposed a configuration in the form of a loop (Figure 6), with a constructing arm (pointing to the north in the left loop and to the east in the right loop) and a replication program, or genome, which turns counterclockwise. After 151 clock periods, the left loop (the mother loop) produces a daughter loop, thus obtaining the self-replication of Langton's loop.

Referring again to biological definitions, we observe that Langton's self-replicating loop is a unicellular organism; its genome, defined in Figure 6, comprises 28 molecules and is a subset of the complete loop which includes 94 molecules.

In summary:

- the size of Langton's loop is perfectly reasonable, since it requires 94 molecules, thus allowing complete simu-

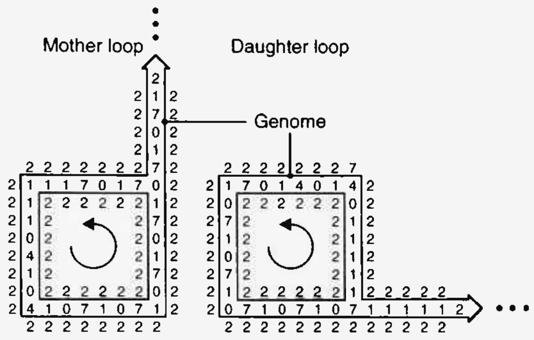

Figure 6: *In Langton's self-replicating loop, the genome, which turns counterclockwise, is characterized by the sequence, read clockwise: 170 170 170 170 170 170 140 140 1111. The signals "1" are ignored, the signals "70" cause the extension of the constructing arm by one molecule, while the signals "40", repeated twice, cause the arm to turn* 90° *counterclockwise. After 151 clock periods, the left loop (the mother loop) produces a daughter loop, thus obtaining the self-replication of Langton's loop. The genome is both interpreted (construction of a copy at the end of the constructing arm: translation process) and copied (duplication at the junction of arm and loop: transcription process).*

lation;

- there is no universal construction or calculation: the loop does nothing but replicate itself; comparing Figure 4 and Figure 6 reveals that Langton's self-replicating loop represents a special case of von Neumann's self-replication; the loop is a non-universal constructor, capable of building, on the basis of its genome, a single type of machine: itself.

Self-replicating loops with computing capabilities

The loops of the previous section exhibit only rudimentary computing and constructing capabilities, their sole functionality being that of self-replication. Lately, new attempts have been made to redesign Langton's loop in order to embed calculation capabilities. Tempesti's loop [19] is a self-replicating automaton which preserves some of the more interesting features of Langton's loop (in particular, it preserves the structure based on a square loop to dynamically store information, and the concept of a constructing arm); nevertheless, Tempesti introduced important modifications to Langton's design. Tempesti's loop attaches an executable program that is duplicated and executed in each of the copies (Figure 7), a process demonstrated for a simple program that writes out (after the loop's replication) LSL, acronym of the Logic Systems Laboratory [21].

Self-replicating loops with universal computing capabilities

Perrier et al.'s self-replicating loop [15] exibits universal computational capabilities (Figure 8). The system consists of three parts, loop, program, and data, all of which are replicated, followed by the program's execution on the given data. In the figure, P and D denote states belonging to the set of program states and to the set of data states, respectively.

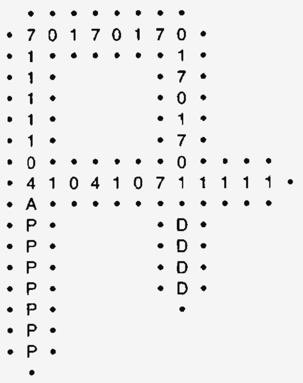

Figure 8: *Perrier et al.'s self-replicating loop. P denotes a state belonging to the set of program states. D denotes a state belonging to the set of data states. A is a state which indicates the position in the program.*

The universal computational model chosen for this work was the W-machine, introduced by Hao Wang [24] and named for him by Lee [9], who explored its relation with finite automata. A W-machine is like a Turing machine with two symbols $S = 0$ and $S = 1$, save that its operation at each time step is guided not by the three functions f_1, f_2, f_3 of a state table but by an instruction from the following list [1]:

PRINT 0, PRINT 1, MOVE DOWN, MOVE UP, IF 1 THEN (*n*) ELSE (*next*), STOP

The complete program for a Turing machine is a finite ordered list of instructions (a program) equivalent to the state table. After execution of an instruction of the first four types, control is automatically transferred to the next instruction. The conditional jump transfers control to the n-th instruction if the square under scan is a 1 symbol, otherwise it transfers control to the next instruction.

Adding functionality to Langton's loop is, in fact, not possible without major alterations. Perrier et al. developed a relatively complex automaton, in which a two-tape Turing machine was appended to Langton's loop. This automaton exploits Langton's loop as a sort of carrier: the first function of Perrier's loop is to allow Langton's loop to build a copy of itself. The main function

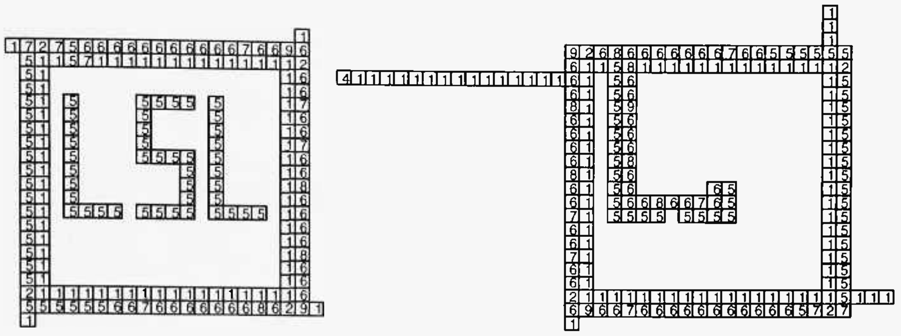

Figure 7: *Tempesti's self-replicating loop has an attached executable program that is duplicated and executed in each copy.*

of the offspring is to determine the location of the copy of the Turing machine. Once the new loop is ready, a messenger runs back to the parent loop and starts to duplicate the Turing machine, a process completely disjoint from the operation of the loop. When the copy is finished, the same messenger activates the Turing machine in the parent loop (the machine had to be inert during the replication process in order to obtain a perfect copy). The process is then repeated in each offspring until the space is filled [15].

The automaton thus becomes a self-replicating universal Turing machine, a powerful construct which is unfortunately handicapped by its complexity: in order to implement a Turing machine, the automaton requires a very considerable number of additional states (63), as well as a large number of additional transition rules. This complexity, while still relatively minor compared to von Neumann's universal constructor, is nevertheless too high to be considered for a hardware application. So once again, adapting Langton's loop to fit our requirements proved too complex to be efficient [21].

Self-replication of specialized Turing machines on a multicellular array: The Embryonics approach

Arbib [1] was the first to suggest a true "cellular" automaton, in which every cell contains a complete copy of the genome, and a hierarchical organization, where each cell is itself decomposed into smaller and regular parts, the "molecules"; unlike all previous realizations, this new architecture is a true "multicellular" artificial organism.

This key idea was the basis of the Embryonics (embryonic electronics) project, under development by Mange and his colleagues since 1993, whose ultimate objective is the construction of large-scale integrated circuits, exhibiting properties such as growth, self-repair (healing), and self-replication, found up until now only in living beings [11, 12, 20].

Embryonics features

Essentially, Embryonics is a modified automata-based approach in which three biologically inspired principles are employed: multicellular organization, cellular differentiation, and cellular division. According to the *multicellular organization* feature, the artificial organism is divided into a finite number of cells (Figure 9), where each cell realizes a unique function, described by a subprogram called the *gene* of the cell.

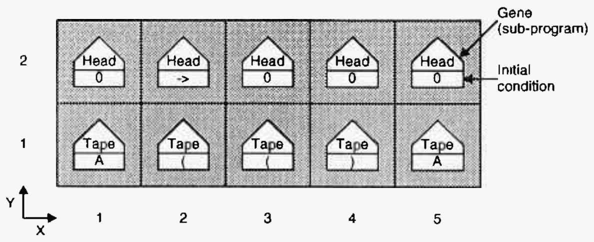

Figure 9: *Multicellular organization of a specialized Turing machine, a parenthesis checker.*

We will confine ourselves to a simple example of a two-dimensional artificial organism (Figure 9): a specialized Turing machine, a parenthesis checker [13], implemented with ten cells and featuring two distinct genes, the *tape gene* and the *head gene*. Each cell is associated with some *initial condition*. In our example the head cells are distinguished by the initial values "0" and "→", the tape cells by "A", "(", and ")" values.

Let us call *genome* the set of all the genes of an artificial organism, where each gene is a sub-program characterized by a set of instructions, by an initial condition, and by a position (its coordinates X, Y). Figure 9 then shows the genome of our Turing machine, with the corresponding horizontal (X) and vertical (Y) coordinates. Let then each cell contain the entire genome (Figure 10): depending on its position in the array, i.e., its place in the organism, each cell can interpret the genome and

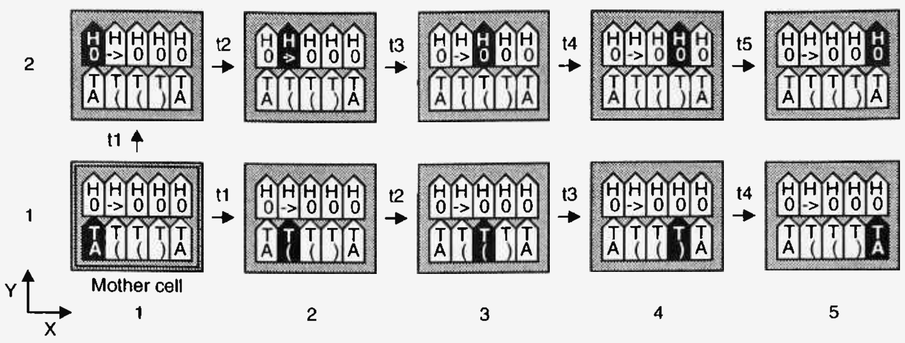

Figure 10: *Cellular differentiation and cellular division of a specialized Turing machine, the parenthesis checker; $t_1...t_5$: five successive divisions.*

extract and execute the gene (with its initial condition) which configures it. According to the *cellular differentiation* feature, it can interpret any gene of the genome (including the initial condition), given the proper coordinates.

At startup, the mother cell or *zygote* (Figure 10), arbitrarily defined as having the coordinates $X, Y = 1, 1$, holds the one and only copy of the genome. At time t_1, according to the *cellular division* feature, the genome of the mother cell is copied into the two neighboring (daughter) cells to the north and to the east. The process then continues until the two-dimensional space is completely programmed. In our example, the furthest cell is programmed at time t_5.

In all living beings, the string of characters which makes up the DNA is executed sequentially by a chemical processor, the *ribosome*. Drawing inspiration from this biological mechanism, we will use a microprogram to compute first the coordinates of the artificial organism, then the initial conditions of each cell, the tape gene and the head gene, and finally the complete genome. The calculation of this microprogram is detailed in [10, 12]. Its software implementation requires basically two kinds of instructions: a *test instruction* (**if** VAR **else** $LABEL$), and an *assignment instruction* (**do** $X = DATA$).

Each artificial cell (called MICTREE for "microinstruction tree") is implemented as an element of a new kind of coarse-grained programmable logic network, which is realized on a special field-programmable gate array (FPGA) circuit. This artificial cell consists basically of a *binary decision machine*, executing the above-mentioned instructions, a random access memory, storing the microprogram of the genome, and several programmable connections linking the cell to its four immediate neighbors (to the north, east, south, and west) [11, 12].

Self-repair and self-replication

In order to demonstrate *self-repair*, we added two spare cells in each row, to the right of the original Turing machine, all identified by the same horizontal coordinate ($X = 6$ in Figure 11). The spare cells may be used not only for self-repair, but also for the example of a Turing machine necessitating growth of the tape of arbitrary, but finite, length.

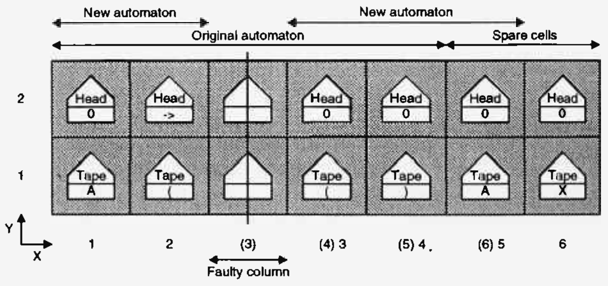

Figure 11: *Self-repair of a 10-cell parenthesis checker in a 14-cell array.*

The existence of a fault is detected by a $KILL$ signal which is calculated in each artificial cell by a built-in self-test realized at the FPGA level. The state $KILL = 1$ identifies the faulty cell and the entire column to which the faulty cell belongs is considered faulty, and is deactivated (column $X = 3$ in Figure 11). All the functions of the artificial cells to the right of the column $X = 2$ 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 columns to repair (there are two spare columns in the example of Figure 11). It also implies that the artificial cell has the capability of bypassing the faulty column and shifting to the right all or part of the original cellular array. During such a process, the actual values calculated by the cells are destroyed and the whole calculation should be restarted.

The *self-replication* of an artificial organism rests on two hypotheses:

- there exist a sufficient number of spare cells (unused cells at the upper side of the array, at least ten for our

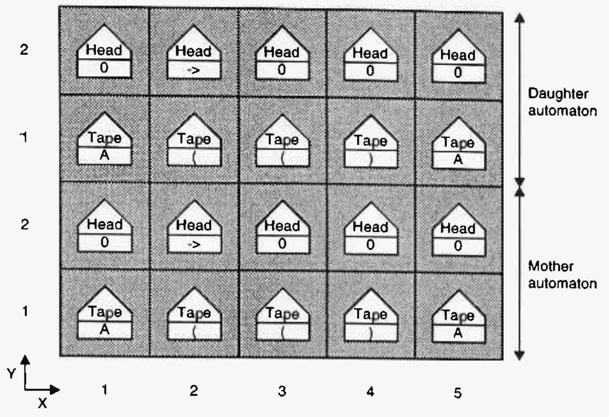

Figure 12: *Self-replication of a 10-cell parenthesis checker in a 20-cell array.*

example);

- the calculation of the coordinates produces a cycle at the cellular level ($Y = 1 \to 2 \to 1$ in Figure 12).

As the same pattern of coordinates produces the same pattern of genes (with the initial conditions), self-replication can be easily accomplished if the microprogram of the genome, associated with the homogeneous network of cells, produces several occurrences of the basic pattern of coordinates ($Y = 1 \to 2$ in Figure 9). In our example, repetition of the vertical coordinate pattern, i.e., the production of the pattern $Y = 1 \to 2 \to 1 \to 2$ (Figure 12), produces one copy, the *daughter automaton*, of the original *mother automaton*. Given a sufficiently large space, the self-replication process can be repeated for any number of specimens in the Y axis (remember that the X axis is reserved for self-repair and/or for a possible growth of the Turing machine).

With a sufficient number of cells, it is obviously possible to combine self-repair (or growth) toward the X direction and self-replication toward the Y direction.

Self-replication of a universal Turing machine on a multicellular array

The preceding section presented a self-replicating two-dimensional artificial organism implementing a specialized Turing machine, the parenthesis checker, which was made of ten MICTREE artificial cells. By using the same type of cells we now show how it is possible to design and build a universal Turing machine (UTM) with self-replication capabilities.

Multicellular architecture of a universal Turing machine

Conventional universal Turing machines [13] consist of a finite but arbitrarily long tape, and a single read/write mobile head controlled by a finite-state machine, which is itself described on the tape (Figure 2). In order to implement a universal Turing machine in an array of MICTREE artificial cells, we made three fundamental architectural choices (Figure 13):

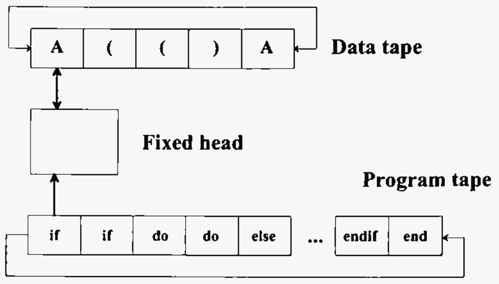

Figure 13: *Universal Turing machine architecture for the parenthesis checker example.*

1. The read/write head is fixed; the tapes are therefore mobile.
2. The data of the given application (the specialized Turing machine to be simulated) are placed in a mobile tape, the *data tape*; this tape can shift right, shift left, or not at all.
3. The finite-state machine for the given application is translated into a very simple program written in a language called PICOPASCAL[1]; each instruction of this program takes place in a square of a second mobile tape, the *program tape*; this tape just needs to shift left.

The fixed head, which is in fact an interpreter of the PICOPASCAL language, has to continuously execute cycles consisting of four operations:

1. reading and decoding an instruction on the program tape;
2. reading a symbol on the data tape;
3. interpreting the current instruction, and writing a new symbol on the current square of the data tape;
4. shifting the data tape (left, right, or none) and the program tape (left).

An application: A binary counter

In order to test our UTM implementation, we used, as a simple but non-trivial example, a binary counter [13], a machine which writes out the binary numbers 1, 10, 11, 100, etc. The counter's state table (Figure 14) has two internal states ($Q \in \{0 \to, 1 \leftarrow\}$) and two input states ($S \in \{0, 1\}$), S being the value of the current square read

[1]PICOPASCAL, itself derived from NANOPASCAL [12], is a minimal subset of PASCAL; the transformation of a state table into such a program is directly inspired by the W-machine with the major advantage of avoiding the jumps necessitated by the **IF 1 THEN** (n) **ELSE** ($next$) instructions.

on the data tape. Depending on the present internal state Q and the present input state S, the specialized Turing machine will:

1. write a new binary value $S^+(0,1)$ on the current square of the data tape;
2. move its tape to the right ($Q^+ = 0 \rightarrow$) or to the left ($Q^+ = 1 \leftarrow$), which is equivalent to moving the data tape to the left or to the right, respectively;
3. go to the next state $Q^+(0 \rightarrow, 1 \leftarrow)$.

$Q+,S+$	$S=0$	$S=1$
$0\rightarrow$	$0\rightarrow,0$	$1\leftarrow,1$
$1\leftarrow$	$0\rightarrow,1$	$1\leftarrow,0$
Q		

Figure 14: *State table of the binary counter.*

The PICOPASCAL program equivalent to the state table (Figure 14) is given in Figure 15.

```
ADDR   DATA   PROGRAM

 00     5     if (Q)
 01     5       if (S)
 02     A         do 0     (S)
 03     9         do 1<-   (Q)
 04     4       else
 05     B         do 1     (S)
 06     8         do 0->   (Q)
 07     6       endif
 08     4     else
 09     5       if (S)
 0A     B         do 1     (S)
 0B     9         do 1<-   (Q)
 0C     4       else
 0D     A         do 0     (S)
 0E     8         do 0->   (Q)
 0F     6       endif
 10     6     endif
 11     2     end
```

Figure 15: *PICOPASCAL program equivalent to the state table of Figure 14.*

An ideal architecture for the universal Turing machine

A universal Turing machine architecture is ideal in the sense that it is able to deal with applications of any complexity, characterized by:

1. a finite, but arbitrarily long data tape;
2. a read/write head able to interpret a PICOPASCAL program of any complexity;
3. a finite, but arbitrarily long program tape.

It must be pointed out that, for any application, the program tape and the read/write head (the PICOPASCAL interpreter) are always characterized by finite dimensions; only the data tape can be as long as desired, as is the case for the binary counter.

An ideal architecture, embedding the current example, but compatible with any other application, is as follows (Figure 16):

1. The data tape, able to shift right, left, or hold, is folded on itself; the initial state is defined by $QL1 : 0, QC, QR0 : 1 = 00100$, where QL are the squares to the left of the central square QC, and QR are squares to the right of QC; the data tape is able to grow to the left of $QC(QL2, QL3, ...)$ and to the right of $QC(QR2, QR3, ...)$.
2. The fixed read/write head, which is not detailed here, is basically composed of a state register Q,S (storing the current values of internal and input states Q,S, respectively, with an initial state $Q,S = 01$) and a stack $ST1 : 3$ characterized by a 1-out-of-3 code (one-hot encoding). At the start of the execution of the PICOPASCAL program (Figure 15, i.e., in address $ADDR = 00$), the stack is in an initial state $ST1 : 3 = 100$; roughly speaking, each **if** instruction will involve a PUSH operation, each **endif** a POP operation, and each **else** a LOAD operation. When $ST1 = 1$, the **do** instructions are executable. The main characteristic of the stack is its scalability: for any program exhibiting n nested **if** instructions, the stack is organized as a $n+1$ square shift register. Both the $ST1 : 3$ stack and the Q, S register are able to grow to accommodate more complex applications.
3. The program tape is folded on itself; it is able to grow to accommodate more complex applications.

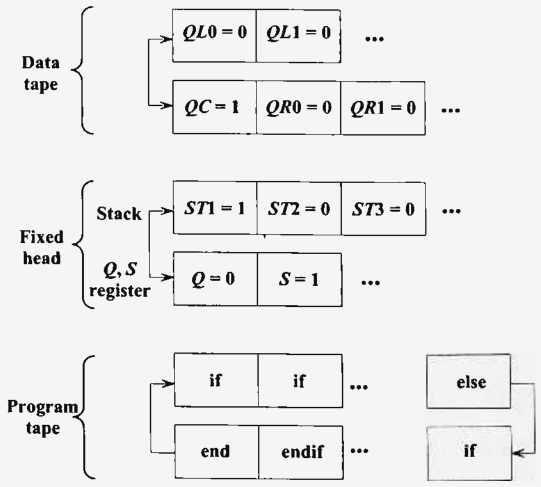

Figure 16: *UTM's ideal architecture.*

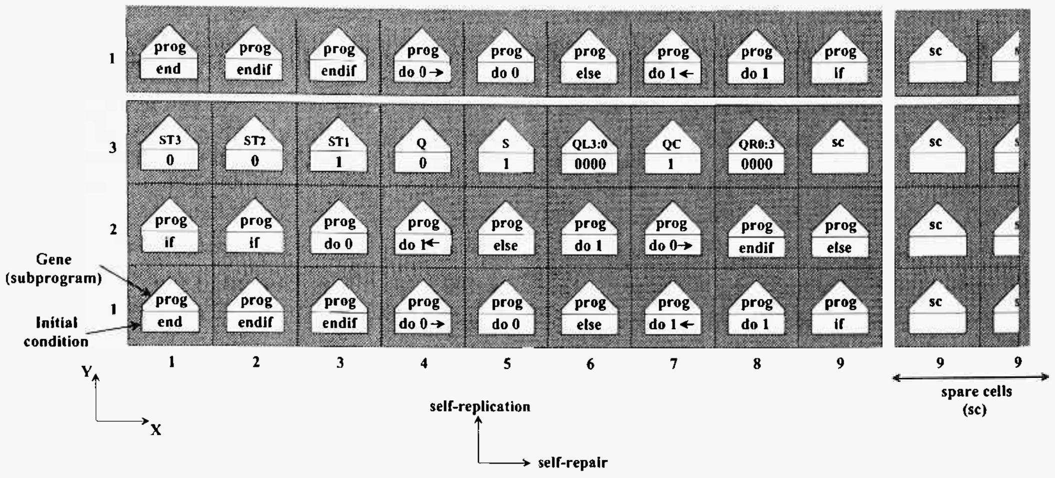

Figure 17: *UTM's actual implementation for the binary counter example on a multicellular array of 27 MICTREE cells.*

An actual implementation of the UTM for the binary counter example

In order to implement the binary counter application with a limited number of MICTREE artificial cells, we have somewhat relaxed the characteristics of the ideal architecture described earlier. Our final architecture is made up of three rows ($Y = 1, 2, 3$) and nine columns ($X = 1...9$) organized as follows (Figure 17):

- The 18 instructions of the PICOPASCAL program (Figure 15) take place in the program tape, using the two lower rows ($Y = 1, 2$) of the array.
- The read/write head is composed of a $ST1 : 3$ stack and the Q,S register ($X = 1...5, Y = 3$), while the data tape is implemented thanks to three cells ($X = 6...8, Y = 3$) displaying 9 bits $QL3 : 0, QC, QR0 : 3$.

In order to demonstrate self-repair, we added spare cells in each row, at the right-hand side of the UTM, all identified by the same horizontal coordinate ($X = 9$ in Figure 17). As previously mentioned, more cells may be used not only for self-repair, but also for a UTM necessitating a growth of the tape of arbitrary, but finite, length.

Self-replication rests on two hypotheses:

- there exist a sufficient number of spare cells (unused cells at the upper side of the array, at least $3 \times 9 = 27$ for our example);
- the calculation of the coordinates produces a cycle at the cellular level (in our example: $Y = 1 \rightarrow 2 \rightarrow 3 \rightarrow 1$).

Given a sufficiently large space, the self-replication process can be repeated for any number of specimens in the Y axis. With a sufficient number of cells, it is obviously possible to combine self-repair (or growth) towards the X direction and self-replication towards the Y direction.

Discussion

In this paper we presented a new, true "multicellular" automaton, in which every cell contains a complete copy of the genome; we have shown that such a multicellular automaton is able to self-replicate and to self-repair.

We then showed that it is possible to embed a universal Turing machine in such a multicellular array, thus obtaining a self-replicating and self-repairing universal Turing machine.

The mapping of the universal Turing machine onto our multicellular array was made possible thanks to the introduction of a modified version of the W-machine, i.e., an interpreter of the PICOPASCAL language. We showed that an ideal architecture was able to deal with applications of any complexity, i.e., with a semi-infinite data tape. We also presented an actual implementation in which we relaxed somewhat the characteristics of the ideal architecture in order to use a limited number of MICTREE artificial cells. We slightly simplified our implementation by presenting the example of the binary counter in which the data are binary (in general we might have discrete values) and where the direction of the head's moves coincides with the internal state (in general functions Q^+ and D^+ are independent).

The property of universal construction raises issues of a different nature, since it requires (according to von

Neumann) that a MICTREE cell be able to implement organisms of any dimension. This challenge can be met by decomposing a cell into molecules and tailoring the structure of cells to the requirements of a given application [11].

Acknowledgments

This work was supported in part by grant 21-54113.98 from the Swiss National Science Foundation, by the Consorzio Ferrara Richerche, Università di Ferrara, Ferrara, Italy, and by the Leenaards Foundation, Lausanne, Switzerland. We thank Barry McMullin and the other two anonymous reviewers for their helpful remarks.

References

[1] M. A. Arbib. *Theories of Abstract Automata.* Prentice-Hall, Englewood Cliffs, N.J., 1969.

[2] E. R. Banks. Universality in Cellular Automata. In *IEEE 11th Annual Symposium on Switching and Automata Theory*, pages 194–215, Santa Monica, California, October 1970.

[3] A. Burks, editor. *Essays on Cellular Automata.* University of Illinois Press, Urbana, Illinois, 1970.

[4] E. F. Codd. *Cellular Automata.* Academic Press, New York, 1968.

[5] G. T. Herman. On Universal Computer-Constructors. *Information Processing Letters*, 2(3):61–64, August 1973.

[6] J. G. Kemeny. Man Viewed as a Machine. *Scientific American*, 192:58–68, April 1955.

[7] C. G. Langton. Self-reproduction in cellular automata. *Physica D*, 10:135–144, 1984.

[8] C. Lee. Synthesis of a Cellular Computer. In J. J. Tou, editor, *Applied Automata Theory*, pages 217–234. Academic Press, London, 1968.

[9] C. Y. Lee. Automata and Finite Automata. *Bell System Tech. Journal*, XXXIX:1267–95, 1960.

[10] D. Mange, D. Madon, A. Stauffer, and G. Tempesti. Von Neumann revisited: A Turing machine with self-repair and self-reproduction properties. *Robotics and Autonomous Systems*, 22(1):35–58, 1997.

[11] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Towards robust integrated circuits: The embryonics approach. *Proceedings of the IEEE*, April 2000. to appear.

[12] D. Mange and M. Tomassini, editors. *Bio-Inspired Computing Machines: Towards Novel Computational Architectures.* Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.

[13] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, New Jersey, 1967.

[14] F. Nourai and R. S. Kashef. A universal four-state cellular computer. *IEEE Transactions on Computers*, c-24(8):766–776, August 1975.

[15] J.-Y. Perrier, M. Sipper, and J. Zahnd. Toward a Viable, Self-Reproducing Universal Computer. *Physica D*, 97:335–352, 1996.

[16] U. Pesavento. An implementation of von Neumann's self-reproducing machine. *Artificial Life*, 2(4):337–354, 1995.

[17] J. Signorini. Complex Computing with Cellular Automata. In P. Manneville, N. Boccara, G. Y. Vichniac, and R. Bidaux, editors, *Cellular Automata and Modeling of Complex Physical Systems*, volume 46 of *Springer Proceedings in Physics*, pages 57–72. Springer-Verlag, Heidelberg, 1990.

[18] M. Sipper. Fifty Years of Research on Self-Replication: An Overview. In M. Sipper, G. Tempesti, D. Mange, and E. Sanchez, editors, *Artificial Life*, volume 4, pages 237–257, Cambridge, Massachusetts, 1998. The MIT Press.

[19] G. Tempesti. A New Self-Reproducing Cellular Automaton Capable of Construction and Computation. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *ECAL'95: Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 555–563, Heidelberg, 1995. Springer-Verlag.

[20] G. Tempesti, D. Mange, and A. Stauffer. Self-Replicating and Self-Repairing Multicellular Automata. *Artificial Life*, 4(3):259–282, 1998.

[21] Gianluca Tempesti. *A Self-Repairing Multiplexer-Based FPGA Inspired by Biological Processes.* PhD thesis, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1998.

[22] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Math. Soc.*, 42:230–265, 1936.

[23] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, Illinois, 1966. Edited and completed by A. W. Burks.

[24] H. Wang. A Variant to Turing's Theory of Computing Machines. *Journal of the ACM*, IV:63–92, 1957.