ELSEVIER

Robotics and Autonomous Systems 22 (1997) 35–58

Robotics and Autonomous Systems

Von Neumann revisited: A Turing machine with self-repair and self-reproduction properties

Daniel Mange *, Dominik Madon, André Stauffer, Gianluca Tempesti

Logic Systems Laboratory, Swiss Federal Institute of Technology, INN-Ecublens, CH 1015 Lausanne, Switzerland

Abstract

The growth and the operation of all living beings are directed through the interpretation, in each of their cells, of a chemical program, the DNA. This program, called *genome*, is the blueprint of the organism and consists of a sequence of four discrete characters: A, C, G, and T. This process is the source of inspiration for the Embryonics (embryological electronics) project, whose final objective is the conception of very large scale integrated circuits endowed with properties usually associated with the living world: self-repair (cicatrization) and self-reproduction. Within this framework, we will present a new family of coarse-grained field-programmable gate arrays. Each cell is a binary decision machine whose microprogram represents the genome, and each part of the microprogram is a gene whose execution depends on the physical position of the cell in the array, i.e. on its coordinates. The considerable redundancy introduced by the presence of a genome in each cell has significant advantages: self-reproduction (the automatic production of one or more copies of the original organism) and self-repair (the automatic repair of one or more faulty cells) become relatively simple operations. Both self-reproduction and self-repair will be illustrated by a classical example of a special-purpose Turing machine: a parenthesis checker. Even if the described system seems exceedingly complex, we believe that computer architectures inspired by molecular biology will allow the development of new FPGAs endowed with quasi-biological properties extremely useful in environments where human intervention is necessarily limited (nuclear plants, space applications, etc.).

Keywords: Von Neumann; Turing machine; Self repair; Self reproduction

1. Introduction

1.1. Towards embryonics

A human being consists of approximately 60 trillion (60×10^{12}) cells. At each instant, in each of these 60 trillion cells, the *genome*, a ribbon of 2 billion characters, is decoded to produce the proteins needed for the survival of the organism. This genome contains the ensemble of the genetic inheritance of the individual and, at the same time, the instructions for both the construction and the operation of the organism. The parallel execution of 60 trillion genomes in as many cells occurs ceaselessly from the conception to the death of the individual. Faults are rare and, in the majority of cases, successfully detected and repaired.

This process is remarkable for its complexity and its precision. Moreover, it relies on completely discrete processes: the chemical structure of DNA (the chemical substrate of the genome) is a sequence of four bases, usually designated with the letters A (adenine), C (cytosine), G (guanine), and T (thymine). Each group of three bases is decoded in the cell to

* Corresponding author. Tel.: 4121 693 26 39; fax: 4121 693 37 05; e-mail: mange@di.epfl.ch.

PII S0921-8890(97)00015-8

produce a particular amino acid, a future constituent of the final protein (thus, the triplet ACG will produce threonine [32]).

Our research is inspired by the basic processes of molecular biology [26]. By adopting certain features of cellular organization, and by transposing them to the two-dimensional world of integrated circuits on silicon, we will show that properties unique to the living world, such as self-reproduction and self-repair, can also be applied to artificial objects (integrated circuits).

To the best of our knowledge, the word *embryonics* was coined by de Garis [6] to mean *embryological electronics*. Our work constitutes a first concrete approach to embryonics and aims at the realization of a new family of coarse-grained field-programmable gate arrays (FPGAs) endowed with quasi-biological properties.

1.2. Objectives and contents

Two main stages characterize the development of self-reproducing automata [27]:

- Von Neumann and his successors (Burks, Thatcher, Lee, Codd, Banks, etc.) developed universal computing automata, that is, self-reproducing automata capable of simulating a universal Turing machine; unfortunately, the complexity of these automata is such that no physical implementation has yet been possible, and only partial simulations have been realized.
- Langton and his successors (Byl, Reggia et al.) developed self-reproducing automata which are much simpler and which have been simulated in their entirety; these machines, however, lack any computing ability and are capable exclusively of self-reproduction.

The purpose of our research is to propose a third approach, called Embryonics. Borrowing three features (multicellular organization, cellular differentiation, and cellular division) from living organisms, we introduce the architecture of a new cellular automaton, complex enough for universal computation, but simple enough for a physical implementation through the use of commercially available digital integrated circuits. The method of construction, self-reproduction, and self-repair of an artificial organism will be illustrated through the example of a specialized Turing machine, and will lead to a realistic physical implementation. We will conclude by suggesting to interested biologists to analyze a first specimen of an artificial genome.

Our research aims at reaching two main objectives:

- A technical objective, i.e. the conception and implementation of high-complexity integrated circuits endowed with quasi-biological properties (self-repair, self-reproduction). These two properties, characteristic of the living world, seem particularly desirable for very complex artificial systems meant for hostile (nuclear plants) or inaccessible (space) environments. Self-reproduction allows the complete reconstruction of the original device in case of a major fault, while self-repair allows a partial reconstruction in case of a minor fault.
- A scientific objective, i.e. the establishment of the logical conditions which must be satisfied by a cellular automaton endowed with self-reproduction and universal computation capability.

Section 2 recalls the two main stages in the development of self-reproducing automata and recapitulates the main results obtained by von Neumann, Langton, and their successors.

Section 3 introduces the fundamental features of Embryonics, a project based on three features, which roughly duplicate the process of cellular development: multicellular organization, cellular differentiation, and cellular division.

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 mechanism, we will use a microprogram to calculate first each gene of the artificial organism, then its coordinates, and, finally, its complete genome. Section 4 applies this method to the example of a specialized Turing machine, a parenthesis checker.

In Section 5 we present a summary of the main features of the MICROTREE cell, the basic unit of our new coarse-grained programmable logic array. This cell is based on a binary decision machine which executes microprograms of up to 1024 instructions. The instructions are essentially tests and assignments (for the calculation of the genes, of the coordinates, and of the genome) and configuration directives (for the programming of the busses connecting each cell to its immediate neighbors). The MICROTREE cell is then embedded into a demonstration module, the BIODULE 601. Finally, we will show that the specialized Turing machine, implemented with an array of

BIODULES 601, is capable of self-repair and self-reproduction.

In Section 6, we conclude by demonstrating that universal computation, that is the capability of realizing, repairing, and reproducing a universal Turing machine, can be verified with our multicellular automaton despite some limitations due to the actual hardware implementation.

2. Historical survey

2.1. *Von Neumann's self-reproducing cellular automaton*

The early history of the theory of self-reproducing machines is basically the history of John von Neumann's thinking on the matter [8]. Von Neumann's cellular automaton [31], as well as all the machines described in this article, is based on the following general hypotheses:

(a) 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;
(b) the physical space is two-dimensional and as large as desired;
(c) the physical space is *homogeneous*, that is comprised by identical *elements*, all of which have the same internal architecture and the same connections with their neighbors; only the *state* of an element (the combination of the values in its memories) can distinguish it from its neighbors;
(d) the reproduction is asexual: the daughter automaton is identical to the mother automaton.

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 "element"; in fact, 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 element of von Neumann's automaton is a finite state machine with 29 states. The future state of an element depends on the present state of the element 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 [31], von Neumann successively showed that a possible *configuration* (a set of elements in a given state) of his automaton can implement a *universal constructor* (Uconst) endowed of the three following properties: constructional universality (Fig. 1(a)), self-reproduction of the universal constructor (Fig. 1(b)), and self-reproduction of a universal calculator (Ucomp) (Fig. 1(c)).

According to the biological definition of a cell, it can be stated that:

(a) 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 (Fig. 1(c)); as each element of this description needs five elements of the genome [31], it can be estimated that the genome is composed of approximately five times the number of elements of the universal constructor and computer;
(b) the dimensions of von Neumann's automaton are substantial (in the order of 200 000 elements) [12]; it has thus never been physically implemented and has been simulated only partially [5,28];
(c) the automaton implements the self-reproduction of a universal computer (a universal Turing machine).

If von Neumann and his successors Burks [3,31], Thatcher [3], Lee [14], Codd [5], Banks [2], Nourai and Kashef [24] demonstrated the theoretical possibility of realizing self-reproducing automata with universal calculation, a practical implementation requires a sharply different approach. It was finally C. Langton, who opened a second stage in this field of research in 1984.

2.2. *Langton's self-reproducing loop*

In order to construct a self-reproducing automaton simpler than this of von Neumann, Langton [13] adopted more liberal criteria. He dropped the condition that the self-reproducing unit must be capable of universal construction and computation. Langton's mechanism is based on an extremely simple configuration in Codd's automaton [5] called the periodic emitter, itself derived from the periodic pulser organ in von Neumann's automaton [31]. The *element* of Langton's automaton is a finite state machine with only eight states. The future state, as for von

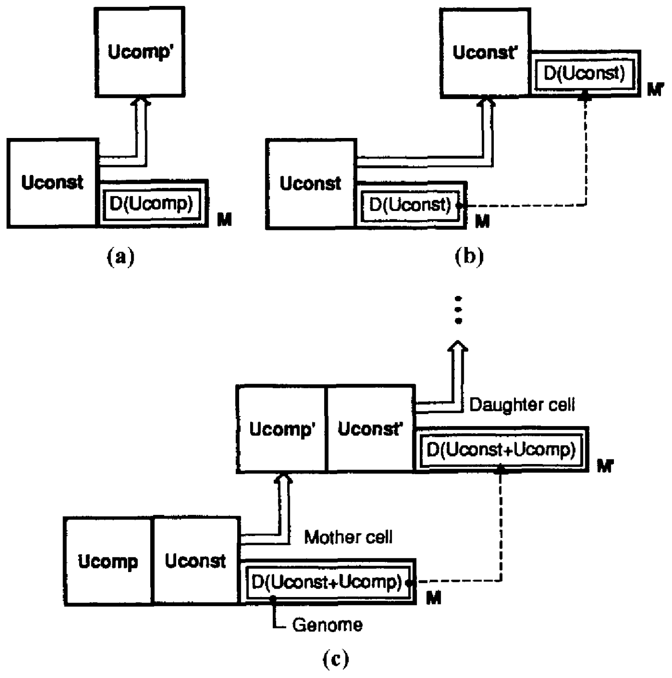


Fig. 1. The three properties of von Neumann's automaton. (a) Constructional universality: a possible configuration can implement a universal constructor Uconst. Then, given the description D(Ucomp) of any one machine UComp, including a universal Turing machine, the universal constructor can build a specimen of this machine (Ucomp') in the cellular space. (b) Self-reproduction 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 cellular 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). (c) Self-reproduction of a universal calculator: 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 above (interpretation and then duplication of the description D).

Neumann's automaton, depends on the present state of the element itself and of its four cardinal neighbors. The exhaustive definition of the future state, the transition table, comprises only 219 lines, a very small subset of the theoretically possible $8^5 = 262\,144$ lines.

Langton proposes a configuration in the form of a loop (Fig. 2), endowed notably with a constructing arm (pointing to the north in the left loop and to the east in the right loop) and of 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-reproduction of Langton's loop.

Again referring to biological definitions, we can observe that:

(a) Langton's self-reproducing loop is a unicellular organism; its genome, defined in Fig. 2, requires 28 elements and is a subset of the complete loop which requires 94 elements;
(b) the size of Langton's loop is perfectly reasonable, since it requires 94 elements, thus allowing complete simulation;
(c) there is no universal construction nor calculation: the loop does nothing but reproduce itself; comparing Figs. 1(b) and 2 reveals that Langton's self-reproducing loop represents a special case of von Neumann's self-reproduction of a universal constructor; the loop is a non-universal constructor, capable of building, on the basis of its genome, a single type of machine: itself.

More recently, Byl [4] proposed a simplified version of Langton's automaton. Last, but not least, Reggia et al. [27] discovered that having a sheath surrounding the data paths of the genome (the "2" signals in Fig. 2)

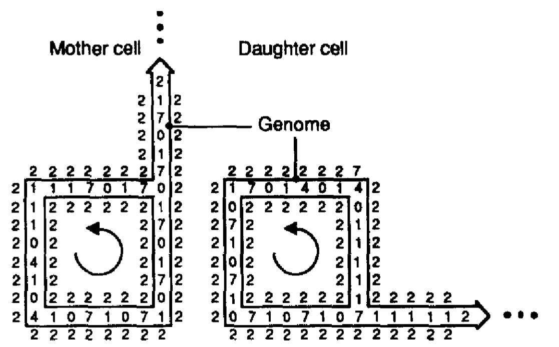


Fig. 2. Langton's self-reproducing loop. The genome, which turns counterclockwise, is characterized by the sequence, read clockwise: 170 170 170 170 170 170 140 140 1111. The signals "1" are ignored, the signals "70" cause the extension of the constructing arm by one element, while the signals "40", 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-reproduction 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).

was not essential, and that its removal led to smaller self-reproducing structures which also have simpler transition functions. Moreover, they found that relaxing the strong symmetry requirement consistently led to transition functions that required fewer rules than the corresponding strong symmetry version.

These last months, new attempts have been made to redesign Langton's loop in order to embed some calculation possibilities [30], or even some kind of Turing machine [25].

In the same vein, but with a completely different methodology, Morris [23] used the concept of *typogenetics*, first introduced by Hofstadter [11], to reproduce strings of characters analogous to those of DNA in a one-dimensional environment. In all these constructions (Langton's, Byl's, Reggia's two-dimensional, and Morris' one-dimensional), the initial configurations "do nothing but propagate" [23].

3. The foundations of embryonics

Our final objective is the realization of a computing machine offering at the same time the original properties of von Neumann's automaton (constructional universality, self-reproduction of a universal calculator) and the simplicity of Langton's loop (reasonable size, allowing not only a complete software simulation, but also a physical realization through existing digital integrated circuits). This objective can be approached by introducing a new architecture, a *multicellular automaton*, roughly derived from the structure of multicellular living beings, and based on the following three characteristics: multicellular organization, cellular differentiation, and cellular division. The proposed automaton meets all the general hypotheses described above for von Neumann's automaton. The element is a true "cell", according to the biological definition: it contains a random access memory (RAM), which stores the microprogram of the complete genome. This microprogram is executed by a small processor, a binary decision machine analogous to the ribosome of the living cell. The microprogram itself is decomposed in sub-programs which are equivalent to the different parts of the genome, i.e. the genes.

3.1. First feature: Multicellular organization

The first feature is *multicellular organization*: the artificial organism is divided into a finite number of cells (Fig. 3(a)), where each cell realizes a unique function, described by a sub-program called the *gene* of the cell. The same organism can contain multiple cells of the same kind (in the same way as a living being can contain a large number of cells with the same function: nervous cells, skin cells liver cells, etc.).

In this presentation, for clarity's sake, we will confine ourselves to a simple example of a two-dimensional artificial organism (Fig. 3(a)): a *specialized Turing machine*, a *parenthesis checker* [22], implemented with 10 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.

The design of these genes will be analyzed in detail later: let us simply state here that the gene is part of the global program of the cell, the genome, and that the execution of a particular gene, as well as the determination of the value of the initial condition, depend only on the position of the cell in the whole array, that is, on its coordinates.

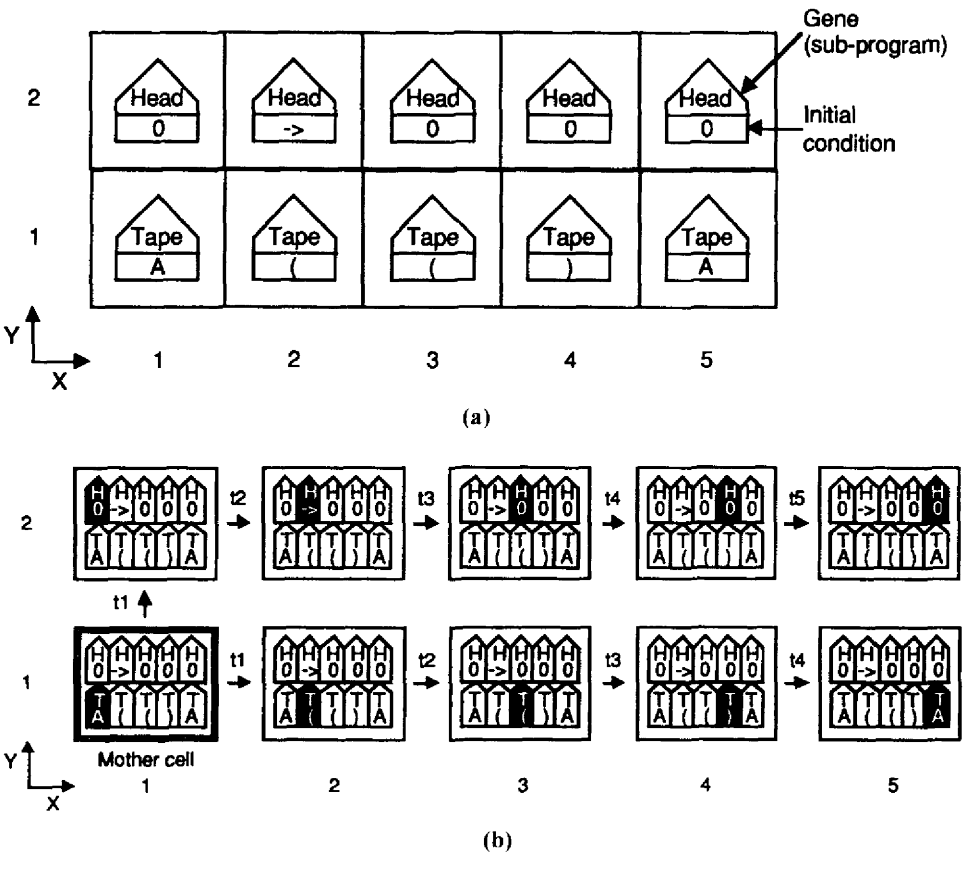


Fig. 3. The three characteristics of the new multicellular automaton: example of a specialized Turing machine, a parenthesis checker. (a) Multicellular organization. (b) Cellular differentiation and cellular division; $t1, \ldots, t5$: five successive divisions.

3.2. Second feature: Cellular differentiation

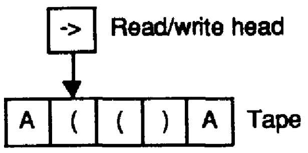
Let us call *genome* the set of all the genes of an artificial organism, where each gene is a sub-program characterized by a set of instructions, by an initial condition, and by a position (its coordinates X, Y). Fig. 3(a) then shows the genome of our Turing machine, with the corresponding horizontal (X) and vertical (Y) coordinates. Let then each cell contain the entire genome (Fig. 3(b)): depending on its position in the array, i.e., its place in the organism, each cell can interpret the genome and extract and execute the gene (with its initial condition) which configures it.

In summary, storing the whole genome in each cell makes the cell universal: it can realize any gene of the genome (including the initial condition), given the proper coordinates.

3.3. Third feature: cellular division

At startup, the mother cell or *zygote* (Fig. 3(b)), arbitrarily defined as having the coordinate $X, Y = 1, 1$, holds the one and only copy of the genome. At time $t1$, 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 $t5$.

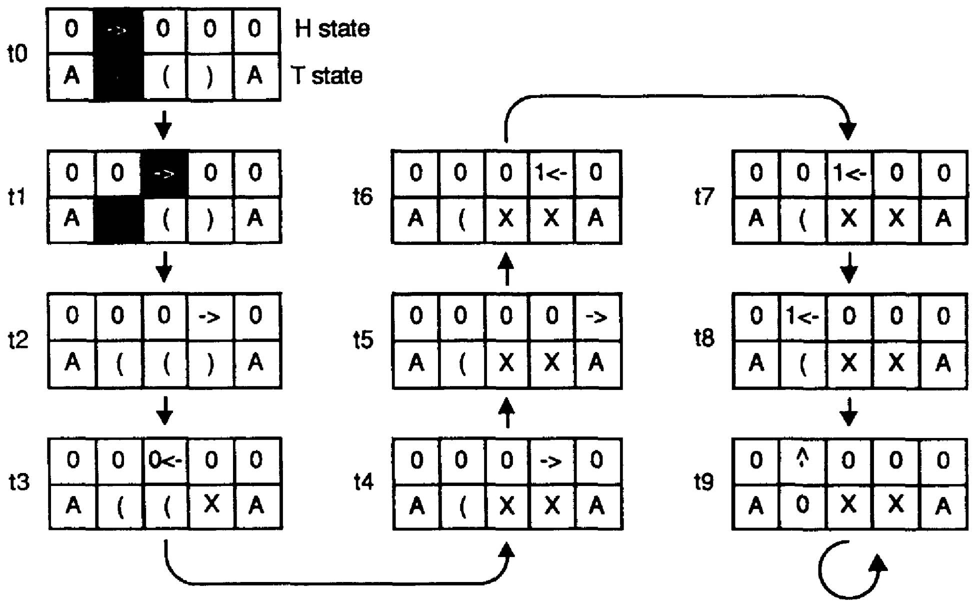
As in the embryological development of living multicellular beings, each mother cell produces a maximum of two daughter cells. It should be noted that, while the biological division of a mother cell in one or two daughter cells occurs to the detriment of the mother cell, which disappears, the artificial system which we propose allows it to survive.



(a)

H+,T+	T X	(	A	)	0	1
->	->,X	->,(	1<-,A	0<-,X	-,-	-,-
0<-	0<-,X	->,X	^,0	0<-,)	-,-	-,-
1<-	1<-,X	^,0	^,1	-,-	-,-	-,-
^	-,-	-,-	-,-	-,-	^,0	^,1
H						

(b)



(c)

Fig. 4. A parenthesis checker: (a) general overview; (b) state table: H: head state, T: tape state; (c) successive H and T states during a 10-step sequence ($t0, \ldots, t9$).

4. An example of Turing machine: A parenthesis checker

4.1. Description

According to Minsky [22, pp. 121–123], the problem is to decide whether a sequence of left (open) and right (closed) parentheses is well-formed. A good procedure for checking parentheses consists in searching to the right for a right (closed) parenthesis, then searching to the left for its mate (open), and removing both. This procedure is repeated until no more pairs are found. If any unmatched symbols remain, the expression is not well-formed, and conversely.

A specialized Turing machine for checking parentheses consists of a tape, decomposed in squares, and a finite-state machine with a read/write head. For checking the expression "(()", for example, we prepare the tape in the form of Fig. 4(a), where the beginning and end of the expression are marked by "A" symbols.

The read/write head starts by reading the first left parenthesis (Fig. 4(a)); the initial state of the finite

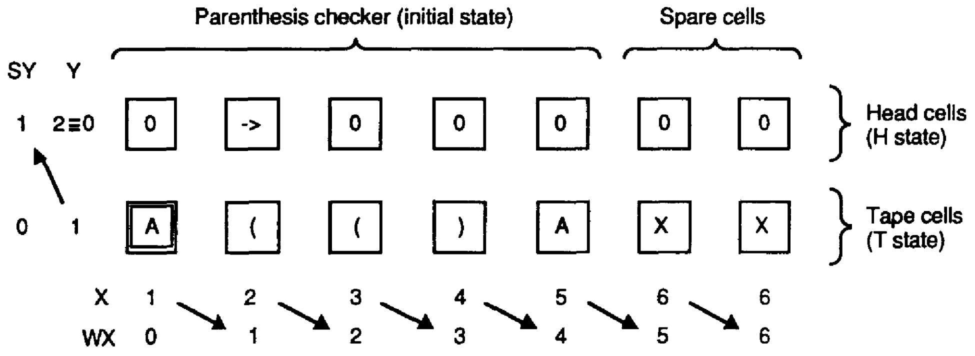


Fig. 5. 10-cell ($X = 1, \dots, 5$) parenthesis checker with four spare cells ($X = 6$); *WX*: horizontal coordinate of the western neighboring cell; *SY*: vertical coordinate of the southern neighboring cell.

state-machine (Fig. 4(b)) is the "→" state. For each couple H,T (*head state, tape state*), the state table of Fig. 4(b) produces a new couple H+, T+ giving the next head state H+, belonging to the set {→, 0 ←, 1 ←, ↑} and the next tape state T+, belonging to the set {X, (, A,), 0,1}.

The initial couple H,T (Fig. 4(c)): time *t*0) is defined in our example as H,T = →, (and produces a next couple H+,T+ = →,(. The result of this computation may be illustrated as follows (Fig. 4(c): time *t*1):

- the next head state H+ is "→"; the read/write head moves one square to the right and displays this state in the new square;
- the next tape state T+ is "("; the square of the tape read by the head at time *t*0 is unchanged.

For the moment, we assume that the read/write head can move and has, at a given time, one single position; all the squares with H = 0 are, therefore, without any physical significance. This process continues until time *t*9, when no more changes are possible. After going through the head states "→" (move to the right), "0 ←" (move to the left, 1st type), "1 ←" (move to the left, 2nd type), the finite-state machine reaches finally the "↑" state (hold). During this process, the read/write head has replaced on the tape all the left "(" and right ")" parentheses by the symbol "X". As the expression "(()" is not well-formed, the last symbol written on the tape by the head is "0". Conversely, if the expression were well-formed, the last symbol would be "1".

4.2. Cellular implementation

Conventional Turing machines consist of a tape, not infinite, but as long as desired, and a single read/write mobile head controlled by a finite-state machine.

In order to obtain a cellular implementation for our parenthesis checker which would be compatible with the homogeneous cellular space defined above, we will realize our Turing machine as follows (Fig. 5):

- The cellular space is divided in two rows, identified by the vertical coordinate Y ($Y = 1$ or 2), and by N columns, identified by the horizontal coordinate X ($X = 1, \dots, N$). In our example, defined by Fig. 4(a), $N = 5$. In order to demonstrate self-repair, we have decided to add two spare cells in each row, to the right of the Turing machine, all identified by the same horizontal coordinate $N + 1$ ($X = 6$ in our example); for technical reasons, each cell also contains the coordinates WX of its western neighbor and SY of its southern neighbor. By definition, the mother cell always has the coordinates $X, Y = 11$, i.e. $WX, SY = 00$.
- The upper row, identified by $Y = 2$, realizes the moving read/write head; all the cells of this row are in a new *quiescent head state* (H = 0), except for one cell which implements the moving head with a non-quiescent state (H = "→", "0 ←", "1 ←" or "↑").
- The lower row, identified by $Y = 1$, realizes the conventional tape. At the start, all the cells of this row are in one of the three tape states T = "A", "(" or ")"; the spare cells are in the *quiescent tape*

state (T = X). At the end of the computation process, all the cells in the "(" and ")" states are replaced by "X" state, except for the unique cell giving the final result (T = 1 for a well-formed expression, T = 0 in the contrary).

In conclusion, each cell of the upper row (the head cell) must contain the complete information for calculating the next head state H+, while each cell of the lower row (the tape cell) must contain the whole information for calculating the next state T+.

It must finally be pointed out that the spare cells, at the right of the cellular space, may be used not only for self-repair, as in our example, but also for other examples of Turing machines necessitating a growth of the tape of any, but non-infinite, length.

4.3. Computing the coordinates

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 local horizontal coordinate (X) of a given cell is computed as a function of the horizontal coordinate of its western neighbor (WX). If we represent the coordinate WX in its binary form $WX2:0$, the specifications of Fig. 5 allow us to derive directly the Karnaugh map for X (Fig. 6(a)). A "don't care" condition (Φ) is specified for the unused value $WX2:0 = 111$.

The use of Karnaugh map for simplifying *binary decision trees* [16,17] generates a tree with seven branches (Fig. 6(b)), each represented by a block in the map. Each test element of the tree, represented by a diamond and defined by a test variable, has a single input, a true output (test variable equal to 1), and a complemented output (test variable equal to 0), identified by a small circle. The leaf elements, represented as squares, define the output value of the given function (X in our example).

For a microprogrammed realization, the binary decision tree of Fig. 6(b), is the flowchart for the gene of X coordinate. The software implementation of this flowchart requires two kinds of instructions: a test instruction and an assignment instruction.

The *test instruction* is defined by an address ADR and a test variable VAR. Each instruction of this kind has one input and two outputs defining the address of the instruction to be executed next: ADR + 1 when VAR = 1 (increment) or ADR0 when VAR = 0. Thus, the *mnemonic expression* of a test instruction is:

IF VAR **ELSE** ADR0 (1)

or, in the more convenient format typical of the assembly language used below:

if VAR **else** LABEL (2)

where LABEL identifies a unique line in the program, and, therefore, a unique memory address.

The *non-conditional jump* is a particular case of the test instruction where the test variable is the logic constant 0. Its mnemonic expression is simply:

goto LABEL (3)

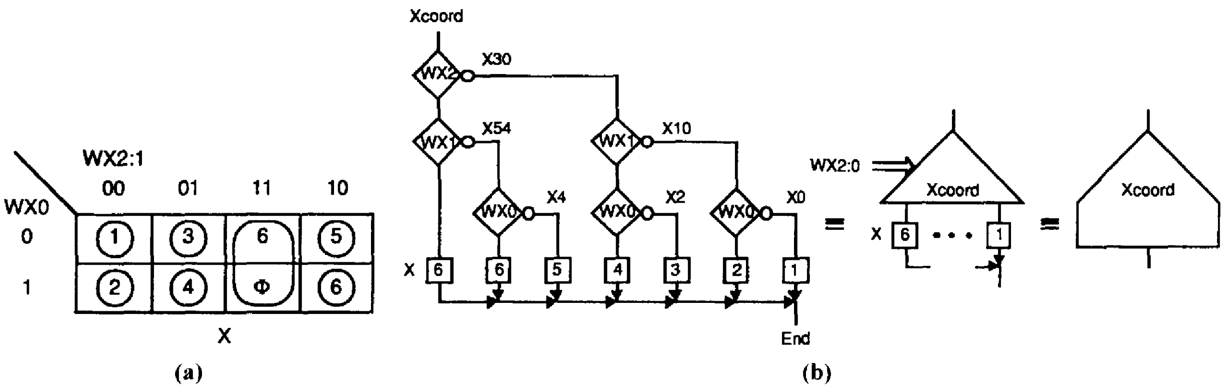
The *assignment instruction* is defined by an address ADR and a synchronous assignment $X \leftarrow$ DATA, where X specifies the X coordinate register and DATA an output state (a decimal constant). Each instruction of this kind has one input and one output defining the address ADR + 1 (increment) of the instruction to be executed next. The mnemonic expression of this assignment instruction is:

do X = DATA (4)

The sub-program, (or gene) **Xcoord** calculating the X coordinate can then be written using the mnemonic expressions (2)–(4), and the labels X30, X54, X4, X10, X2, X0 and End (Fig. 6(c)). The local vertical coordinate (Y) of a given cell is computed as a function of the vertical coordinate of its southern neighbor (SY). Unlike the organization of the rows, which requires the placement of spare cells (for self-repair and/or extension of the tape), we wish to design a cycle on the vertical coordinate in order to demonstrate self-reproduction (see Section 5.3). Such a cycle can be described by the expression $Y = 1 \rightarrow 2 \rightarrow 1$, which can be coded in the following binary form (Figs. 5 and 20):

$$Y = 1 \rightarrow 0 \rightarrow 1 \quad (5)$$

The trivial Karnaugh map of Fig. 7(a), therefore, defines Y as a function of $SY0$ (the unique binary



```
Xcoord:  if WX2 else X30
         if WX1 else X54
         do X =6
         goto End
X54:     if WX0 else X4
         do X =6
         goto End
X4:      do X =5
         goto End
X30:     if WX1 else X10
         if WX0 else X2
         do X =4
         goto End
X2:      do X =3
         goto End
X10:     if WX0 else X0
         do X =2
         goto End
X0:      do X =1
End:     ...
```

(c)

Fig. 6. Computing the horizontal X coordinate (sub-program **Xcoord**): (a) Karnaugh map; (b) binary decision tree and flowcharts; (c) assembly language sub-program.

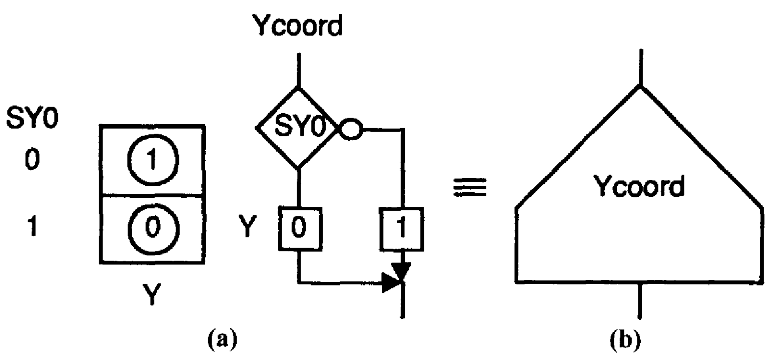


Fig. 7. Computing the vertical Y coordinate (sub-program **Ycoord**): (a) Karnaugh map; (b) binary decision tree and flowcharts.

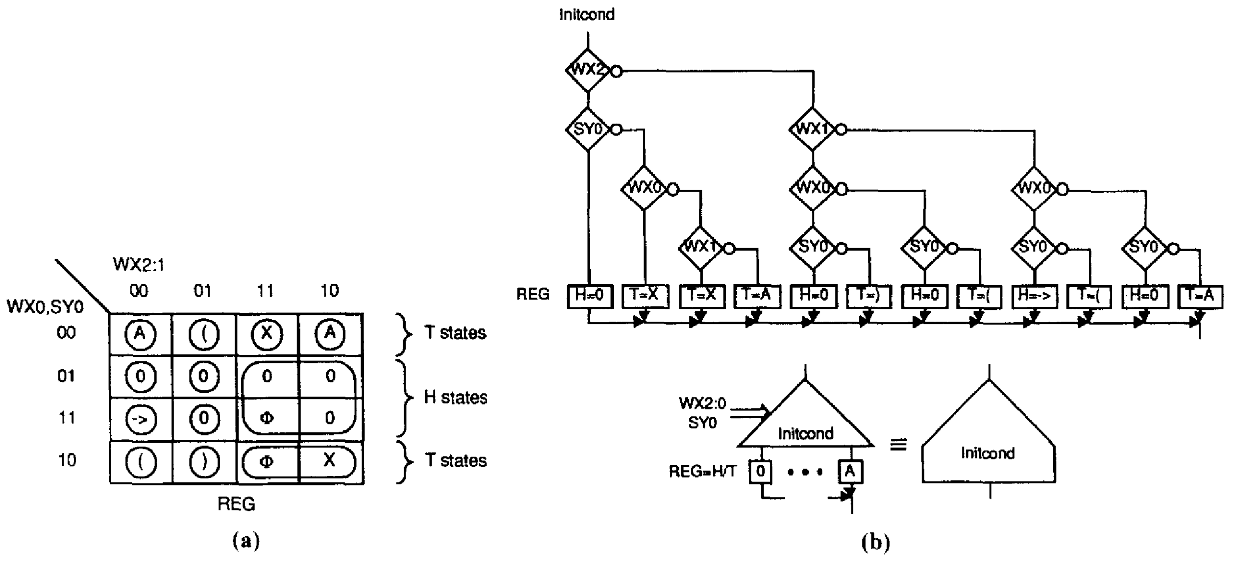


Fig. 8. Computing the initial conditions (sub-program **Initcond**): (a) Karnaugh map; (b) binary decision tree and flowcharts.

variable describing SY) and leads to the final binary decision tree and flowchart of Fig. 7(b). For the vertical coordinate Y, we define a new but symmetrical assignment instruction whose mnemonic expression is:

do Y = DATA (6)

The sub-program (or gene) **Ycoord** can then be expressed using the mnemonics (2), (3) and (6).

4.4. Computing the initial conditions

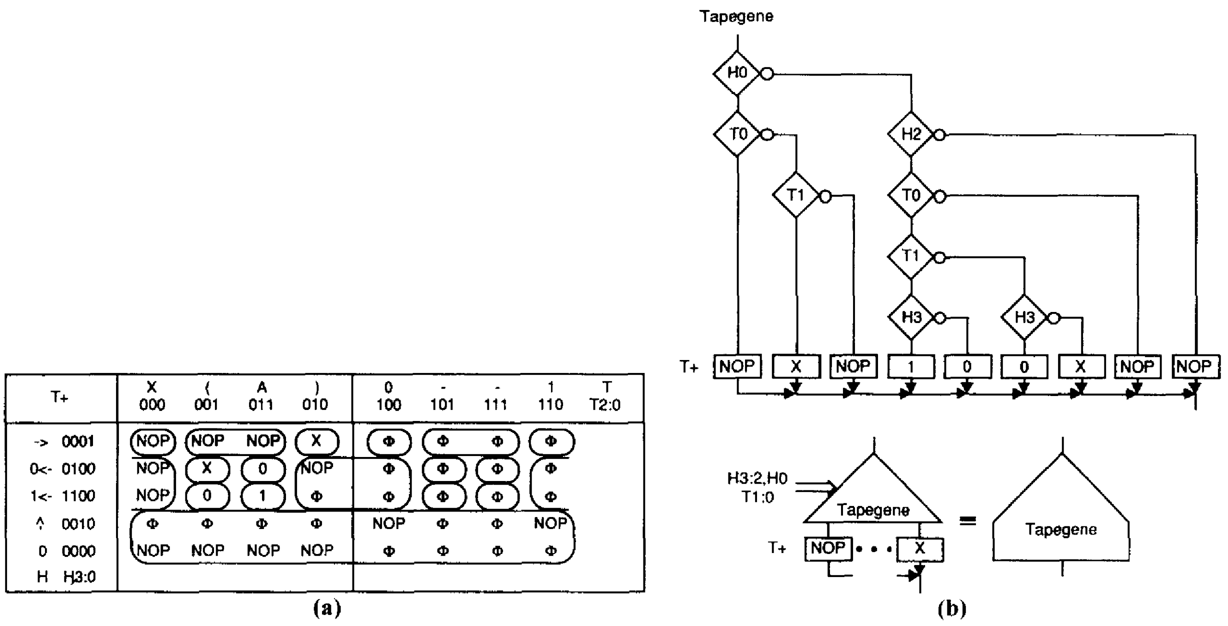
The difficulty of setting the cellular space in an initial configuration is generally underestimated. Starting self-reproduction with the historical von Neumann's automaton [31] would require the assignment of a given state out of 29 to approximately 200 000 elements. In our case, computing the initial conditions, i.e. setting each cell of the cellular space in a given head state H or type state T according to Fig. 5, is a part of the complete microprogram. In other words, it is a gene of the complete genome.

The methodology is roughly the same as that used above for computing the coordinates. The initial state of each cell of the cellular space (Fig. 5) is expressed as a function of the horizontal coordinate of its western neighbor (WX) and of the vertical coordinate of its southern neighbor (SY). If we represent the coordinate WX in its binary form $WX2:0$ and SY as $SY0$, the specifications of Fig. 5 allow us to derive directly the Karnaugh map for REG (Fig. 8(a)), i.e. the state register of each cell (REG = H for head cells, REG = T for tape cells). A "don't care" or Φ-condition is specified for the unused value $WX2:0 = 111$. The 12 blocks of the Karnaugh map generate a simplified binary decision tree with 12 branches, which in turn generates the flowchart of the **Initcond** sub-program (Fig. 8(b)). This flowchart required the introduction of a new type of assignment instruction realizing a synchronous transfer REG ← DATA, where REG specifies the cell state register (H register or T register) and DATA an output state. The mnemonic expression of such an assignment instruction is simply:

do REG = DATA (7)

For technical reasons, the final binary coding of the H and T states will be chosen later (Section 4.5). To avoid any confusion between H and T states (for example H = → and T = (will be coded with the same 4 bit word 0001), we have kept the original H and T values in the assignment instructions of Fig. 8(b).

It must finally be pointed out that Figs. 8(a) and (b) illustrate the particular example of Fig. 5. For any other initial conditions related to the same parenthesis checker, this part of the microprogram must be redesigned to meet the new specifications.



T+	X 000	(001	A 011	) 010	0 100	- 101	- 111	1 110	T T2:0
-> 0001	NOP	NOP	NOP	X	Φ	Φ	Φ	Φ	
0<- 0100	NOP	X	0	NOP	Φ	Φ	Φ	Φ	
1<- 1100	NOP	0	1	Φ	Φ	Φ	Φ	Φ	
↑ 0010	Φ	Φ	Φ	Φ	NOP	Φ	Φ	NOP	
0 0000	NOP	NOP	NOP	NOP	Φ	Φ	Φ	Φ	
H H3:0									

Fig. 9. Tape gene computation (sub-program **Tapegene**): (a) Karnaugh map; (b) binary decision tree and flowcharts.

4.5. Computing the tape gene

Starting from the state table of Fig. 4(b), we wish to express the next tape state T+ as a function of the present tape state T and the present head state H. Using the Karnaugh map of Fig. 9(a), we first must add the new head state H = 0, which is the quiescent head state introduced above (Section 4.2) for the cellular realization of our Turing machine. The new Karnaugh map is then derived from the original state table according to the following additions and tranformations:

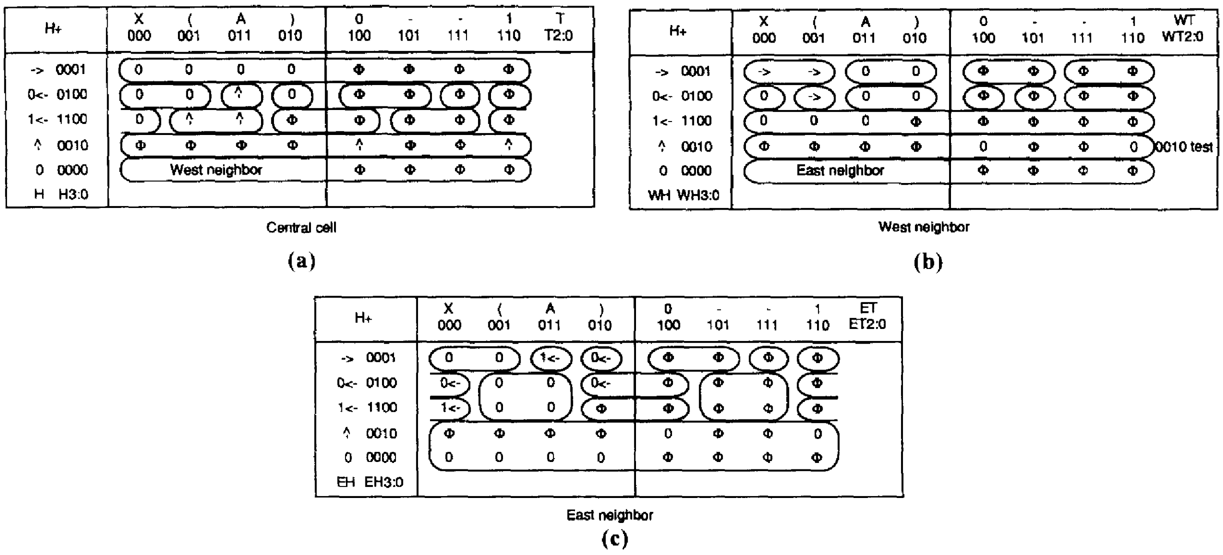
- The row for the head state H = 0 is completed. Examination of the sequence in Fig. 4(c) shows that T+ is a *neutral* or *no-operation state* (NOP) or a "don't care" condition (Φ).
- Values for T+ are borrowed from Fig. 4(b). When T+ = T, the next tape state T+ can be replaced by a NOP state; a dash is replaced by a "don't care" condition (Φ).
- A binary coding is proposed for both T and H states. This coding has been chosen in order to obtain the usual configuration for a Karnaugh map; this map is completely developed for the eight columns (eight T states) and partially for the five rows (in order to obtain simple boolean expressions, we introduce a non-minimal coding and use only a small subset of the 16 possible H3 : 0 states).

We therefore obtain a partial 7-variable Karnaugh map, which is convenient enough for our simplification needs. Recalling that the two half-maps (for T2 = 0 and T2 = 1) are adjacent, we obtain nine blocks in the Karnaugh map, a binary decision tree with nine branches and the corresponding flowchart (Fig. 9(b)). Writing the sub-program **Tapegene** does not require additional instructions beyond those defined above.

4.6. Computing the head gene

As illustrated in Fig. 4(c), we have to keep in mind that the head moves to the right if the next head state H+ is equal to "$\rightarrow$", and to the left if H+ is equal to "0 $\leftarrow$" or "1 $\leftarrow$". We therefore have to consider three different situations which will be combined to obtain the final microprogram calculating the next head state H+:

- If the next head state H+ is the hold state "↑", the head does not move and the cell keeps its "↑" state; we therefore rewrite all the occurrences of H+ = ↑ from the original state table of Fig. 4(b) in the new Karnaugh map of Fig. 10(a). If the next state H+



H+	X 000	(001	A 011	) 010	0 100	- 101	- 111	1 110	T T2:0
-> 0001	0	0	0	0	Φ	Φ	Φ	Φ	
0<- 0100	0	0	↑	0	Φ	Φ	Φ	Φ	
1<- 1100	0	↑	↑	Φ	Φ	Φ	Φ	Φ	
↑ 0010	Φ	Φ	Φ	Φ	↑	Φ	Φ	↑	
0 0000	West neighbor				Φ	Φ	Φ	Φ	
H H3:0									

Central cell

(a)

H+	X 000	(001	A 011	) 010	0 100	- 101	- 111	1 110	WT WT2:0
-> 0001	->	->	0	0	Φ	Φ	Φ	Φ	
0<- 0100	0	->	0	0	Φ	Φ	Φ	Φ	
1<- 1100	0	0	0	Φ	Φ	Φ	Φ	Φ	
↑ 0010	Φ	Φ	Φ	Φ	0	Φ	Φ	0	0010 test
0 0000	East neighbor				Φ	Φ	Φ	Φ	
WH WH3:0									

West neighbor

(b)

H+	X 000	(001	A 011	) 010	0 100	- 101	- 111	1 110	ET ET2:0
-> 0001	0	0	1<-	0<-	Φ	Φ	Φ	Φ	
0<- 0100	0<-	0	0	0<-	Φ	Φ	Φ	Φ	
1<- 1100	1<-	0	0	Φ	Φ	Φ	Φ	Φ	
↑ 0010	Φ	Φ	Φ	Φ	0	Φ	Φ	0	
0 0000	0	0	0	0	Φ	Φ	Φ	Φ	
EH EH3:0									

East neighbor

(c)

Fig. 10. Head gene computation (sub-program **Headgene**): (a) Karnaugh map for the central cell; (b) Karnaugh map for the West neighboring cell; (c) Karnaugh map for the East neighboring cell.

in Fig. 4(b) is "→", "0 ←" or "1 ←", the head moves (right or left) and therefore leaves the present square, and the next state H+ of the actual cell will be "0". All the dashes of Fig. 4(b) become "don't care" conditions (Φ) in Fig. 10(a). Finally, if the present H state is the quiescent "0" state, its next value H+ will depend on the next state of its two West and East neighbors (WH+ and EH+).

- A first test is then performed on the West neighbor. All occurrences of the next head state WH+ = → are rewritten from the original state table of Fig. 4(b), depending in that case on the WT2 : 0 and WH3 : 0 variables, in the new Karnaugh map of Fig. 10(b) which completes Fig. 10(a). Except for the Φ-conditions, all other next states WH+ produce the value H+ = 0 in this new map. Finally, if the West neighbor cell is itself in the quiescent state WH = 0, a last test is performed on the East neighbor.
- All the occurrences of the next head state EH+ = 0 ← or 1 ← are rewritten from the original state table in the new Karnaugh map of Fig. 10(c), which concludes the description of the next head state H+ of the central cell. Except for the Φ-conditions, all other next state EH+ produce the value H+ = 0.

The three Karnaugh maps of Figs. 10(a)–(c) may be simplified and produce seven blocks each, that is a binary decision tree with 21 branches (Fig. 11).

It is possible to introduce further, and more tedious simplifications. As an example, the block defined by WH3 : 0 = 0010 in Fig. 10(b) requires a test on the variable WH1 in the tree of Fig. 11. Such a test may be suppressed for the following reason:

- If WH3 : 0 = 0010, the West neighbor cell is in the hold state WH = ↑ and the next head state H+ will be 0.
- If this test is ignored, the microprogram will test the East neighbor cell which is obviously in a quiescent state (EH = 0), producing the same final result: H+ = 0 (remember that the read/write head has a single position on the tape).

If we suppress the above-mentioned test on the WH1 variable, the final binary decision tree, as well as the corresponding flowchart, consist of 20 branches (Fig. 11). Writing the sub-program **Headgene** does not require additional instructions beyond those defined above.

4.7. *Computing the genome*

Cellular differentiation occurs, in our example, only through the vertical coordinate which, for technical reasons, is computed as a function of the coordinate SY of the preceding cell (the southern neighbor). From

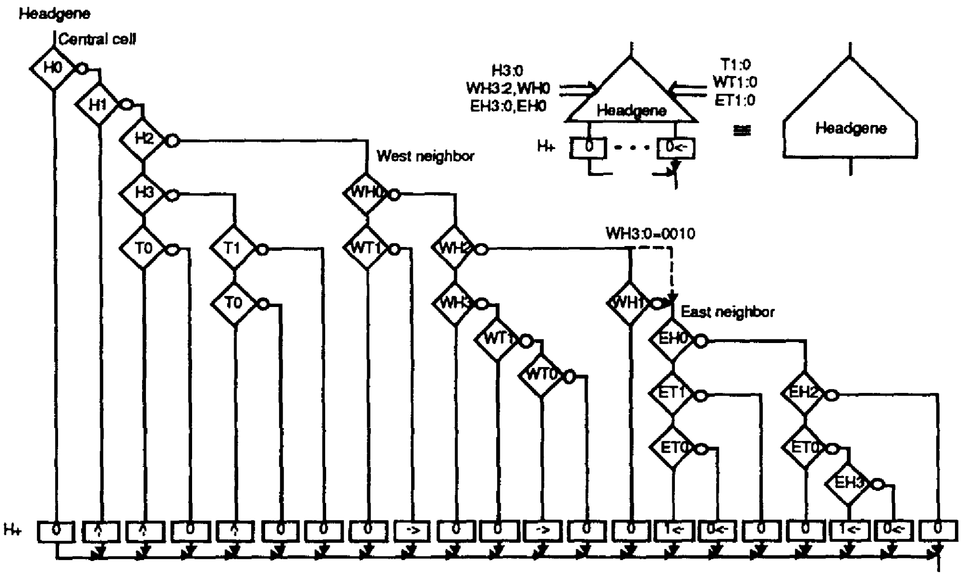


Fig. 11. Head gene computation: binary decision tree and flowcharts (sub-program **Headgene**).

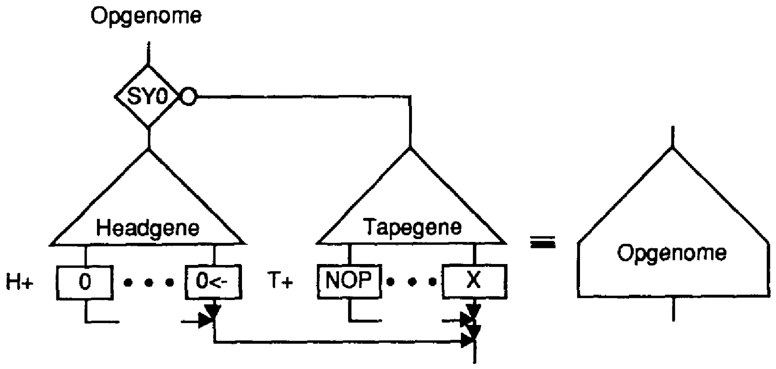


Fig. 12. Flowcharts of the genome operational part (sub-program **Opgenome**).

the description of Fig. 5, the two main genes **Headgene** for the head cells and **Tapegene** for the tape cells can be discriminated by the single $SY0$ variable. The final flowchart for the operational part of the genome, **Opgenome**, which includes the sub-programs **Headgene** and **Tapegene**, is illustrated in Fig. 12.

4.8. Physical configuration: The MICROTREE cell

The original specifications of our specialized Turing machine, a parenthesis checker, allowed us to generate five genes, realized by five sub-programs:

- The **Xcoord** and **Ycoord** genes, dedicated to the computing of the coordinates, and the **Initcond** gene, required for the calculation of the initial conditions of both the tape states and head states. The **Xcoord** and **Initcond** genes depend strongly on the given example, in our case the short "(()" expression.
- The **Tapegene** and **Headgene**, which calculate the original state table of the parenthesis checker as designed by Minsky. These genes are independent of the chosen example and constitute the invariant part of the final microprogram; they are differentiated by the coordinate variable $SY0$ and generate

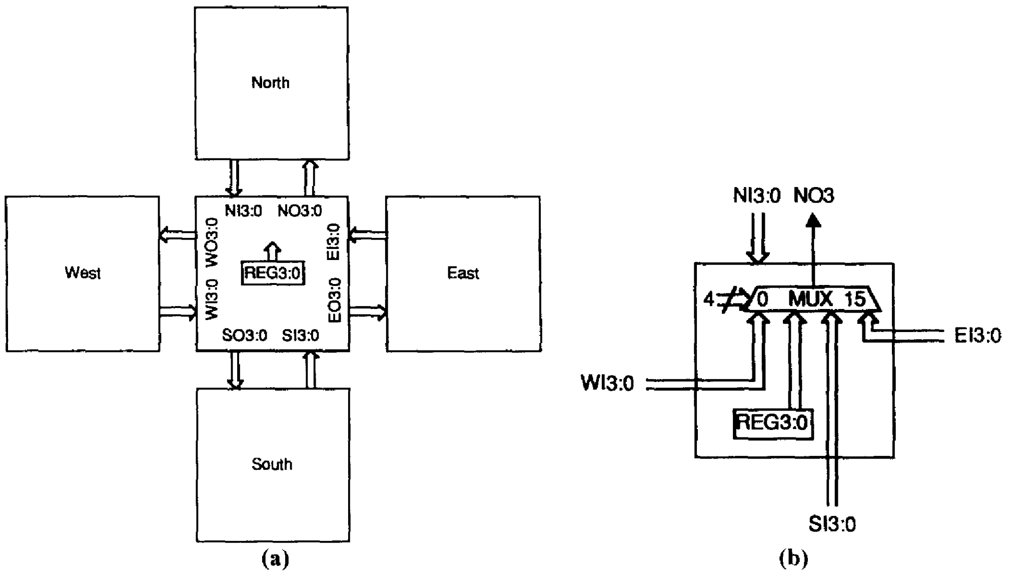


Fig. 13. MICROTREE cell: (a) four neighboring cells connection diagram; REG3 : 0: state register; (b) output variable NO3 example: 16 different sources programming capability.

the sub-program **Opgenome**, the operational part of the genome.

Thus the dynamic part of our microprogram is basically complete. We must now define the static part of the final microprogram, which will fix the connections between cells and realize the final physical configuration of the cellular space.

Each cell is implemented as a new kind of coarse-grained programmable logic network (field-programmable gate array), called MICROTREE (for tree of micro-instructions). Each MICROTREE cell (Fig. 13(a)) has four neighbors (to the South, West, North, and East). Four 4-bit busses enter the cell from its neighbors (SI3 : 0 from the South, WI3 : 0 from the West, NI3 : 0 from the North, and EI3 : 0 from the East) and, correspondingly, four output busses go out in the four cardinal directions (SO3 : 0 to the South, WO3 : 0 to the West, NO3 : 0 to the North, and EO3 : 0 to the East).

Each MICROTREE cell has, therefore, 16 outputs SO3, ..., EO0. Each of these outputs can be programmed to take a value from one of the 16 possible sources (Fig. 13(b)). For example, output NO3 can take one of the following 16 values:

- the four bits REG3 : 0 of the state register REG;
- the four bits SI3 : 0 of the south input bus SI;
- the four bits WI3 : 0 of the west input bus WI;
- the four bits EI3 : 0 of the east input bus EI.

Note that it is impossible for NO3 to get the value of one of the four bits NI3 : 0 of the input bus corresponding the same cardinal direction.

In our assembly language, a single assignment instruction is sufficient to perform this operation. The mnemonic expression for this instruction is:

do VAROUT = VARIN (8)

4.9. *Global configuration*

A physical configuration is *global* when it is realized in all the MICROTREE cells of the array, independently of the value of the coordinates (X and/or Y).

For computing the initial conditions, we need a boolean variable INIT (see the complete microprogram in Fig. 17). This variable will be introduced in the north side of the cellular space and must reach each cell. We thus have the global configuration of Fig. 14, described by an assignment instruction of type (8):

do SO1 = NI1 (9)

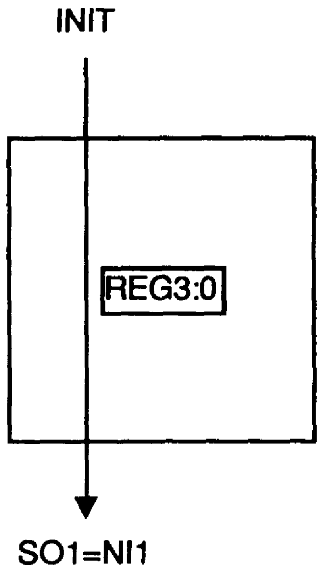


Fig. 14. Parenthesis checker: global configuration.

4.10. Local configuration

A physical configuration is *local* if it is realized by a subset of the MICROTREE cells of the array. Such a configuration depends therefore on the value of the X and/or Y coordinates.

The architecture of our Turing machine (Fig. 5) consists of two cellular rows which are uniform, i.e., characterized by cells with identical structure and behavior (Fig. 15):

- for $SY0 = 0$ ($Y = 1$), the tape cells calculate the tape state REG3 : 0 = T3 : 0. According to Figs. 9(b) and 11, bits of T3 : 0 are needed for the calculation of T+ itself (but do not need any programmable connection) and for the calculation of the next state H+ of the central cell. In this case (Fig. 11), we need connections from the central tape cell to the central head cell (T1 : 0 via the bus NO1 : 0), from the West tape cell to the central head cell (WT0 via the EO0 and NO3 busses, WT1 via the NO1 bus), from the East tape cell to the central head cell (ET0 via the WO0 and NO2 busses, ET1 via the NO1 bus). The corresponding assignment instructions may be written as follows:

$$\begin{array}{ll} \textbf{Tapelocalconfig}: & \\ \textbf{do}\ \mathrm{NO0} = \mathrm{REG0} & (\equiv \mathrm{T0}) \\ \textbf{do}\ \mathrm{NO1} = \mathrm{REG1} & (\equiv \mathrm{T1})(\equiv \mathrm{WT1})(\equiv \mathrm{ET1}) \\ \textbf{do}\ \mathrm{EO0} = \mathrm{REG0} & (\equiv \mathrm{WT0}) \\ \textbf{do}\ \mathrm{NO3} = \mathrm{WI0} & (\equiv \mathrm{WT0}) \\ \textbf{do}\ \mathrm{WO0} = \mathrm{REG0} & (\equiv \mathrm{ET0}) \\ \textbf{do}\ \mathrm{NO2} = \mathrm{EI0} & (\equiv \mathrm{ET0}) \end{array} \tag{10}$$

- For $SY0 = 1$ ($Y = 0$), the head cells calculate the head state REG3 : 0 = H3 : 0. According to Figs. 9(b) and 11, bits of H3 : 0 are needed for the calculation of the next tape state T+ (H3 : 2 and H0

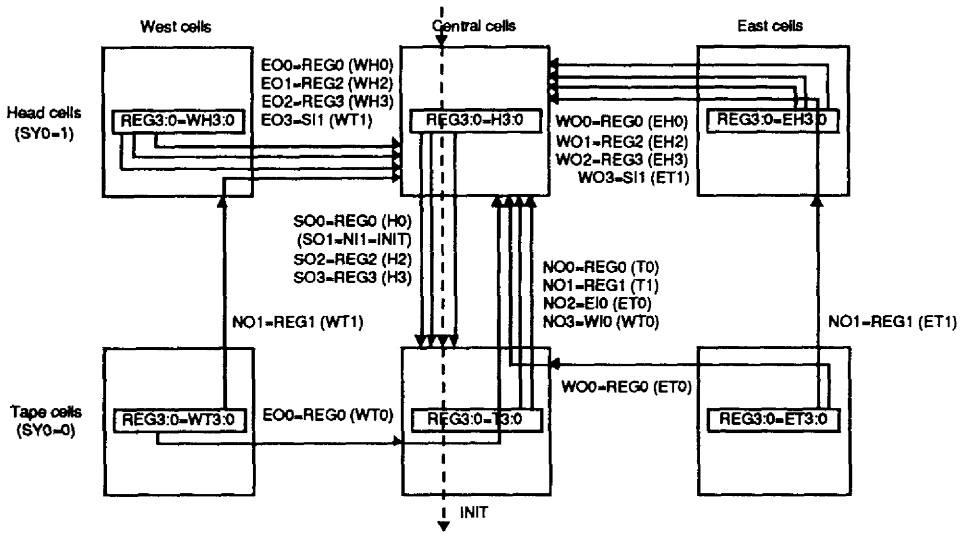


Fig. 15. Parenthesis checker: local configuration.

via the SO3 : 2 and SO0 busses) and for the calculation of the next state H+ of the central cells. In this last case, we need connections from the central head cell itself (H3 : 0 : no programmable connection is needed), from the West head cell to the central head cell (WH3 : 2, 0 via the EO2 : 0 bus, WT1 via the EO3 bus), from the East head cell to the central head cell (EH3 : 2, 0 via the WO2 : 0 bus, ET1 via the WO3 bus). The corresponding assignment instructions may be written as follows:

```
Headlocalconfig :
do SO0 = REG0   (≡ H0)
do SO2 = REG2   (≡ H2)
do SO3 = REG3   (≡ H3)

do EO0 = REG0   (≡ WH0)
do EO1 = REG2   (≡ WH2)
do EO2 = REG3   (≡ WH3)              (11)
do EO3 = SI1    (≡ WT1)

do WO0 = REG0   (≡ EH0)
do WO1 = REG2   (≡ EH2)
do WO2 = REG3   (≡ EH3)
do WO3 = SI1    (≡ ET1)
```

Given the characteristics of a MICROTREE cell (Fig. 13(a)), we observe that routing the different busses between head cells and tape cells is not a trivial task; in fact, we remark that, except the north side, we use all the input/output capabilities of the MICROTREE head cells. A minor simplification, such as that mentioned in Section 4.6 (WH1 test), is of great interest, as the central cell may be independent of the WH1 variable, thus saving one crucial bus connection.

The vertical coordinate $SY0$ performs the differentiation between the local configuration of tape cells and head cells. It is therefore possible to add to the previous **Ycoord** sub-program (Fig. 7(b)) the new **Tapelocalconfig** and **Headlocalconfig** sub-programs determined above. We obtain then the final sub-program (or gene) **Ycoordlocalconfig** (Fig. 16), which combines the calculation of Y coordinate and of the local configuration depending on this variable. The realization of this flowchart does not require additional instructions beyond those defined above.

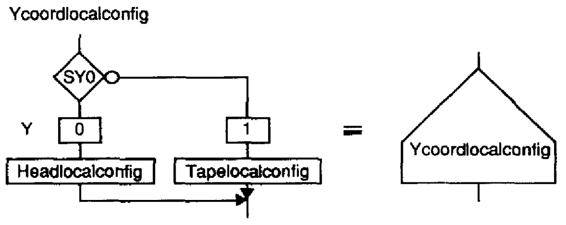


Fig. 16. Flowcharts for Y coordinate and local configuration calculation (**Ycoordlocalconfig** sub-program).

4.11. Microprogram of the complete genome

The complete genome is represented by the final flowchart **Turinggenome** of Fig. 17. It starts with clear conditions assuring that:

- all the state registers of the array are set to 0 (REG = 0);
- the coordinates X and Y are set to 0 ($X = 0$, $Y = 0$).

The microprogram then executes the single global configuration instruction (SO1 = NI1) and, for INIT= 1, the left loop (initialization loop). If K is the largest value for either the X or the Y coordinate (according to Fig. 5, $K = 6$ in our example), the left-hand loop must be executed at least $K = 6$ times: at the start of the microprogram, or when a repair involving a change of coordinates occurs, the coordinates are recomputed starting from the mother cell (with $WX, SY = 00$). At least K executions of the left-hand loop are necessary to ensure that the right-most (or upper-most) cell computes the correct coordinates. This computation occurs in the sub-programs **Xcoord** and **Ycoordlocalconfig**, and is immediately followed by the calculation of the initial conditions (**Initcond**).

When the initial conditions have been computed and/or recomputed after self-repair, it is possible to set INIT= 0; the microprogram executes either the middle or the right loop, depending on the value of the G variable, the global clock signal charged with synchronizing the Turing machine. Each rising edge of G allows the transition from a present tape state T or head state H to a next tape state T+ or head state H+.

The right-hand loop, the operational part of the genome (**Opgenome**), is executed once every period of the global clock signal G. To assure the synchronization of all the cells, tests are performed throughout

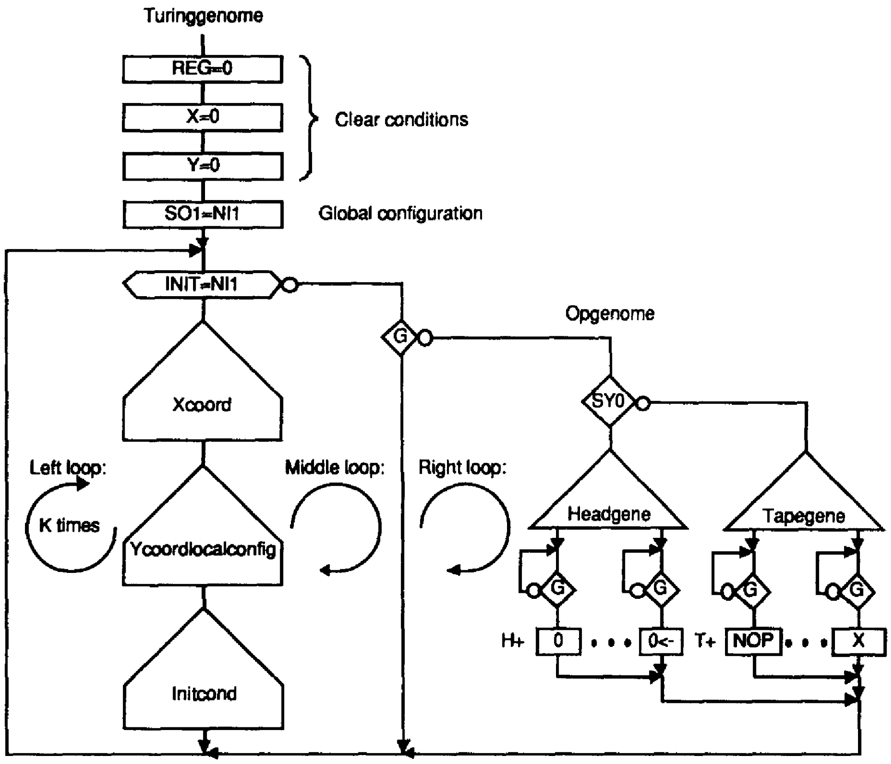


Fig. 17. Flowchart of the complete genome microprogram (**Turinggenome**).

the half-period when $G = 0$, but no assignment is made until the rising edge of G ($G = 0 \rightarrow 1$), when all the registers REG (i.e., the H and T states of the cells) are updated simultaneously.

5. A new field-programmable gate array based on a binary decision machine

5.1. General description

While our long-term objective is the conception of very large scale integrated circuits, we started by realizing a demonstration system in which each MICROTREE cell is embedded into a plastic container called *BIODULE* (type 601) (Fig. 18) [10,15].

The MICROTREE cell consists essentially of a *binary decision machine* [16], executing the microprograms wirtten using the following set of instructions, defined above in the conception of the parenthesis checker, plus the null instruction **nop** (no operation):

- **if** VAR **else** LABEL (2)
- **goto** LABEL (3)
- **do REG** = DATA [**on** MASK] (7)
- **do X** = DATA (4)
- **do Y** = DATA (6)
- **do** VAROUT = VARIN (8)
- **nop** (12)

The state register REG and both coordinate registers are 4-bit wide (REG3 : 0, X3 : 0, and Y3 : 0). The 4 bits of MASK in expression (7) allow us to select which of the bits of REG3 : 0 will be affected by the assignment. By default, MASK = 1111 (all bits are affected).

The variables VAROUT correspond to the four cardinal output busses, for a total of 16 bits (Fig. 13(a)):

$$\text{VAROUT} \in \{\text{SO3} : 0,\ \text{WO3} : 0,\ \text{NO3} : 0,\ \text{EO3} : 0\} \quad (13)$$

while the variables VARIN in expression (8) correspond to the four cardinal input busses and the register REG, for a total of 20 bits:

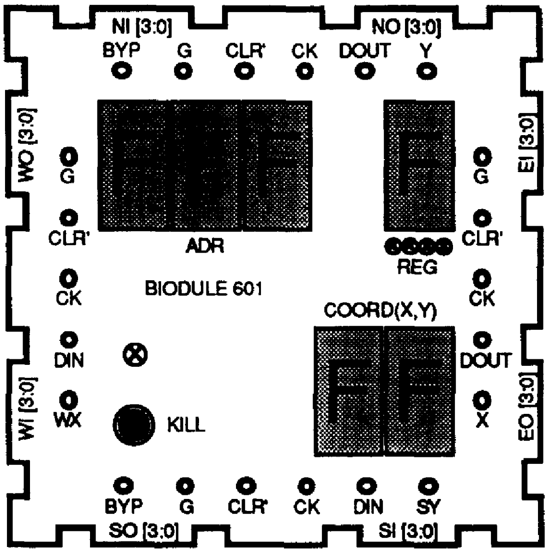


Fig. 18. BIODULE 601: demonstration module including a MICROTREE cell.

$$\text{VARIN} \in \{\text{SI3}:0,\ \text{WI3}:0,\ \text{NI3}:0,\ \text{EI3}:0,\ \text{REG3}:0\} \quad (14)$$

remembering that VAROUT and VARIN can never refer to the same cardinal direction (Fig. 13(b)).

The test variables VAR include the set VARIN and the following additional variables:

$$\text{VAR} \in \{\text{VARIN},\ \text{WX3}:0,\ \text{SY3}:0, G\} \quad (15)$$

where G is a global variable, usually reserved for the synchronization clock.

The coordinates are transmitted from cell to cell serially, but are computed in parallel. Therefore, each cell performs a series-to-parallel conversion on the incoming coordinates WX and SY of the western and southern neighbors respectively, and a parallel-to-series conversion of the coordinates X and Y it computes and propagates. By default (that is, with the external connections WX and SY not connected), the mother cell recognizes the values $WX = SY = 0$.

The genome microprogram is also coded serially. It enters through the DIN pin of the mother cell and is then propagated through its DOUT pin, according to the cellular division path determined by the user (as in Fig. 3(b)).

The pins CK and CLR' are used for the propagation of the clock signal and of the clear of the binary decision machine, while the signal BYP (bypass), connecting all the cells of a column, is used for self-repair.

The size of the artificial organism embedded in an array of MICROTREE cells is limited in the first place by the coordinate space ($X = 0, \ldots, 15$, $Y = 0, \ldots, 15$, that is, a maximum of 256 cells in our current implementation), and then by the size of the memory of the binary decision machine storing the genome microprogram (1024 instructions).

An editor, a compiler for the assembly language, and a loader [10,15] simplify the task of writing and debugging the microprograms and generating the genome's binary code, charged serially through the DIN input of the mother cell.

5.2. Self-repair

In the BIODULES 601 (Fig. 18), the existence of a fault is decided by the human user by pressing the KILL button of a cell. Therefore, fault detection and fault location, two features which will be indispensable in the final system, where they will be implemented using BIST (Built-In-Self-Test) techniques [1,7,21], are not present in the BIODULES 601.

To implement self-repair, we have chosen, favoring simplicity, the following process (Figs. 18 and 19):

- pressing the KILL button identifies the faulty cell (the KILL light, normally green, becomes red);
- the entire column to which the faulty cell belongs is considered faulty, and is deactivated (column $X = 3$ in Fig. 19; all the KILL lights become red);
- all the functions of the MICROTREE cell 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 (two spare columns in the example of Fig. 19). It also implies some modifications to the MICROTREE cell, so as to add the capability of bypassing the faulty cell and shifting to the right all or part of the original cellular array.

With the present and rather simple realization of our parenthesis checker, self-repair may only occur during the initialization process (left loop in Fig. 17). As soon as the calculation of the genome has begun, it is no longer possible to save intermediate results in case of repair, and the computation process, i.e. the check of parentheses, must be restarted.

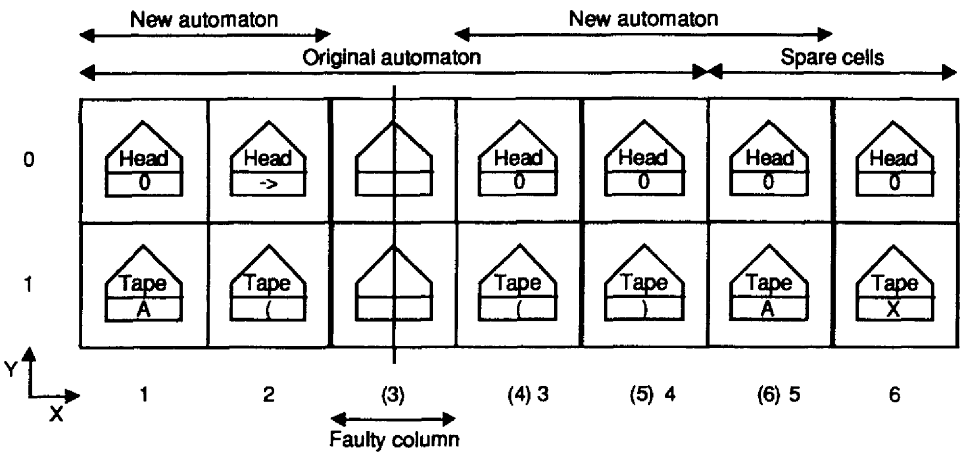


Fig. 19. Self-repair of a 10-cell parenthesis checker in a 14-BIODULE array.

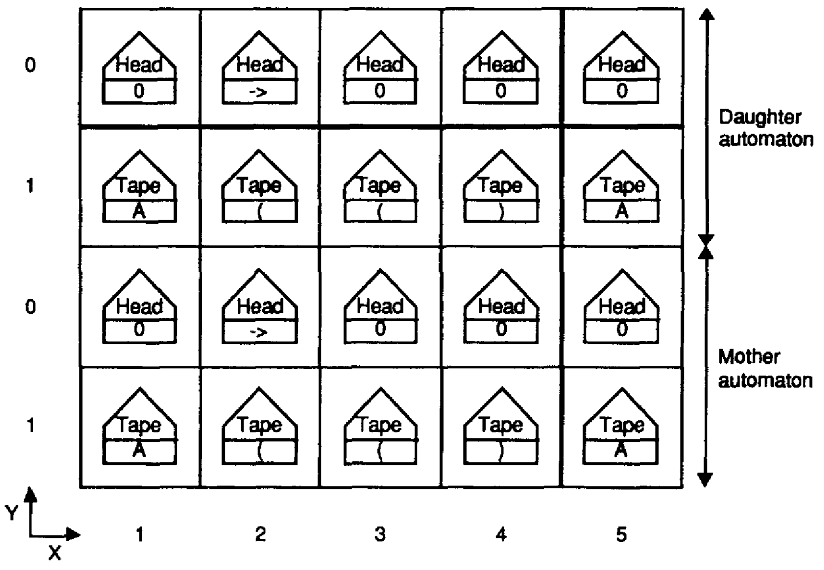


Fig. 20. Self-reproduction of a 10-cell parenthesis checker in a 20-BIODULE array.

5.3. Self-reproduction

The self-reproduction of an artificial organism, for example the specialized Turing machine of Fig. 3(a), rests on two hypotheses: (1) there exists a sufficient number of spare cells (unused cells at the upper side of the array, at least 10 for our example) and (2) the calculation of the coordinates produces a cycle ($Y = 1 \rightarrow 0 \rightarrow 1$ in Fig. 20).

As the same pattern of coordinates produces the same pattern of genes (with the initial conditions), self-reproduction can be easily accomplished if the microprogram of the genome, associated to the homogeneous network of cells, produces several occurrences of the basic pattern of coordinates ($Y = 1 \rightarrow 2$ in Fig. 3(a)). In our example, the repetition of the vertical coordinate pattern, i.e., the production of the pattern $Y = 1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ (Fig. 20), produces one copy, the *daughter automaton*, of the original or *mother automaton*. Given a sufficiently large space, the self-reproduction process can be repeated for any number of specimens in the Y axis (remember that X axis is reserved for self-repair and/or for a possible growth of the Turing machine).

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

6. Conclusions

6.1. Results

The main result of our research is the development of a new family of coarse-grained FPGAs called MICROTREE and based on a binary decision machine capable of executing a microprogram of up to 1024 instructions. The original features of this FPGA are essentially:

- a completely homogeneous organization of the cellular array;
- an integration of the routing into each cell, both for the short- and the long-distance (bus) connections;
- a sequential execution of microprograms methodically derived from a chosen representation, the binary decision tree or diagram.

Our FPGA satisfies all the general hypotheses described in von Neumann's automaton (Section 2.1), as well as the three features of the Embryonics project (Section 3): multicellular organization, cellular differentiation, and cellular division. The MICROTREE cell, itself realized with a commercial FPGA and a RAM, was finally embedded into a demonstration module called BIODULE 601 and we showed that an array of BIODULES 601 is capable of self-repair and self-reproduction.

The trivial applications of the MICROTREE family are those in which all the cells in the array contain the same gene: the genome and the gene then become indistinguishable and the calculation of the coordinates is superfluous. In this case, the cellular array is not limited in space. One-dimensional (Wolfram's) and two-dimensional (life, Langton's loop, etc.) uniform cellular automata are natural candidates for this kind of realization. The non-trivial applications are those in which the cells of an array have different genes: the genome is then a collection of genes, and the coordinates become necessary. The cellular array is then limited by the coordinate space ($16 \times 16 = 256$ cells in the proposed realization). One-dimensional (like the example of a random number generator described in [18]) and two-dimensional non-uniform cellular automata (like the present Turing machine) fall within this category. Let us also mention that the realization of uniform cellular automata with a pre-determined initial state is an important special case which also requires separate genes and a coordinate system: the cellular realization of our parenthesis checker (head row or tape row), represents an application of this kind.

In the first phase of the Embryonics project [19,20, 29], we have proposed a first kind of BIODULE (type 600). The main drawback of this realization was the lack of balance between the application layer (a cell based on a multiplexer with a single control variable, realizing the universal function of a single variable) and the configuration layers (a processor storing and interpreting the genome program). In the new MICROTREE cell we have introduced (BIODULE type 601), the application and configuration layers are indistinguishable. A single microprogram, describing the entire genome, realizes at the same time the operations described by the specifications (the check of parentheses, in our example) and the control of these actions (the calculation of the coordinates, the differentiation of the genes and of the physical configuration). By accepting a sacrifice in execution speed (the binary decision trees are no longer arrays of multiplexers working in parallel, but rather microprograms executed sequentially), we obtain a considerable gain in computational power (1024 executable instructions per cell instead of a multiplexer, equivalent to a single test instruction).

6.2. Historical and theoretical perspectives

Coming back to biology, it should be recalled that the cell is the smallest part of a living being containing the complete blueprint of the creature, the genome. On the basis of this definition, it has been shown that von Neumann's automaton is a *unicellular organism*, since it contains a single copy of the genome, i.e., the description of the universal constructor and computer D(Uconst + Ucomp) (Figs. 1(c) and 21(a)). Each element of the automaton is thus a part of the cell, or, in biological terms, a *molecule*. Von Neumann's automaton, therefore, is a *molecular automaton*; universal construction and self-reproduction are complex processes, as they are caused by the interaction of thousands of elements, the molecules, each

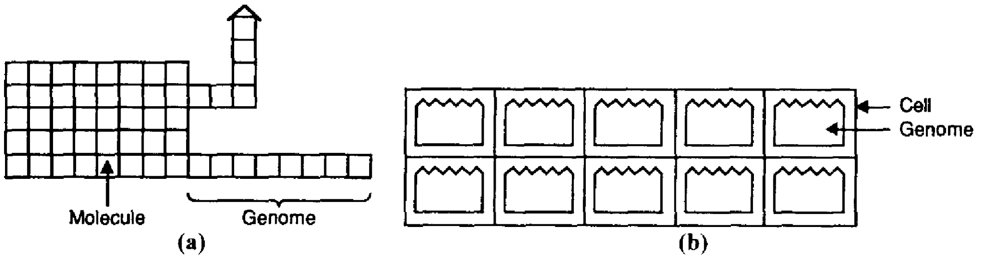


Fig. 21. Comparison between von Neumann's molecular automaton and the new multicellular automaton. (a) Von Neumann's automaton is a unicellular artificial organism, containing a single copy of the genome. Each element of this automaton, a finite state machine with 29 states, is thus a part of the unicellular organism, i.e. a molecule: this automaton is molecular. Each square in the figure symbolically represents an element, and thus a molecule. (b) Our automaton contains as many copies of the genome as there are elements (10 in the case of the parenthesis checker). Each element of this automaton, a binary decision machine with the genome memory, is a cell in the biological sense: this automaton is multicellular.

one realized by a finite-state machine with 29 states. In contrast, the automaton we propose is a *multicellular organism*, as each of its elements contains a copy of the genome (10 copies in the case of the Turing machine of Figs. 3(b) and 21(b)). Each element of an automaton is thus a *cell* in the biological sense, and our automaton is truly a *multicellular automaton*. Self-reproduction and self-repair are simple, even trivial, processes, as the MICROTREE cell has been conceived especially to carry out globally the operations of cellular differentiation and division.

The property of *universal computation*, that is, the possibility of realizing, repairing, and reproducing a universal Turing machine, can be verified with the MICROTREE cell subject to the following limitations:

- The setting of the initial conditions is limited by the finite dimensions of the X coordinate register ($X = 0, \ldots, 15$); however, von Neumann and his successors assumed that the initial configuration was given a priori, and thus this limitation has no theoretical significance.
- A universal Turing machine, as described by Minsky [22], obviously requires a redesign of our operative genome **Opgenome** (Section 4.7). The new design involves a larger number of variables for calculating the head and tape states (H,T) and thus a new and more complex architecture of the cellular space (Fig. 15) and/or of the basic MICROTREE cell as designed for our simple example of a parenthesis checker.

The property of *universal construction* poses problems of a different nature, since it requires (always according to von Neumann) that MICROTREE cells be able to implement artificial organisms of any dimension. The finite dimensions of our cells (memories, registers, etc.) are, for the moment, preventing us from meeting this requirement, a challenge which remains one of our main concerns. We will therefore be led to the design of a new cell with a flexible architecture, whose specifications will be part of the genome. The biological inspiration then becomes straightforward: as the interpreter of the genome, the *ribosome*, is itself built by the instructions of a first part of the DNA (the *ribosomic DNA*), we will be studying a two-stage process (Fig. 22):

- In a first stage, a ribosomic genome will program a fine-grained FPGA, for example a multiplexer-based or MUXTREE FPGA [19,20,29]. We thus obtain the RAM and the binary decision machine which make up a MICROTREE cell with an

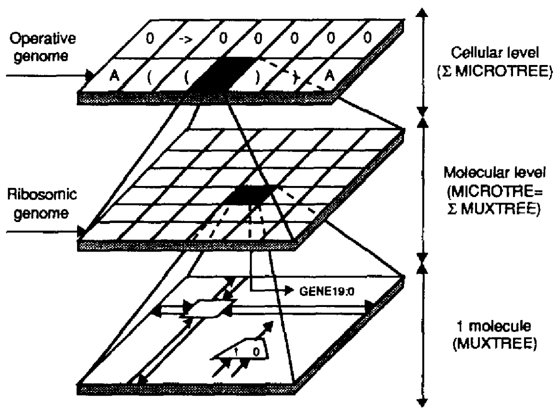

Operative genome
Ribosomic genome
Cellular level (Σ MICROTREE)
Molecular level (MICROTREE= Σ MUXTREE)
1 molecule (MUXTREE)
GENE19:0

Fig. 22. Two-level hierarchical organization for universal construction.

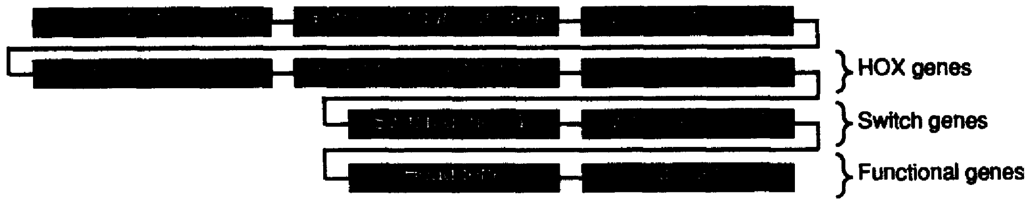


Fig. 23. The artificial genome of the parenthesis checker.

architecture appropriate to our example: this is the *molecular level*, where each FPGA cell is equivalent to a molecule, without any genome memory.

- In a second stage, the operative genome will program the MICROTREE cell in order to execute the given specifications: this is the *cellular level*, where each cell is equivalent to a living cell, with genome memory and interpretation.

We hope that von Neumann's dream can become a hardware reality in a near future.

6.3. A bridge between molecular biology and computer science

As engineers, we have derived our inspiration from the rough organization of multicellular living beings to propose a realistic artificial object. We now want to offer to interested biologists the following considerations:

- the contents of the random access memory of the MICROTREE cell, i.e. the genome, are written in the classical form of a sequential program, comprising a series of instructions;
- in order to limit the number of connections, this program is copied from one cell to another serially (bit by bit); the genome can thus be seen in the form of a linear succession of bits.

We thus obtain a sort of *artificial genome* in the form of a string of binary bases or bits (Fig. 23). In this genome, we can identify:

- *coordinate genes* (**Xcoord**, **Ycoordlocalconfig**, **Initcond**), which handle the computation of the coordinates and the calculation of the initial conditions; these genes are similar to the *homeoboxes* or *HOX genes* recently found to define the general architecture of living beings [32];
- *switch genes* (G and $SY0$ tests), used to express the functional genes according to the cell's position in the organism (that is, according to the value of the cell's coordinates [9]);
- *functional genes*, producing the operative functions of our artificial organism (i.e. calculating head and tape states), analogous to the genes which constitute the coding part of the natural genome.

The existence of these different categories of genes is a consequence of purely logical needs deriving from the conception of our multicellular automaton. We hope that this bridge between computer science and molecular biology could perhaps bring something to biology itself.

Acknowledgements

This work was supported by the Swiss National Science Foundation, grant Nos. 21-36200.92 and 20-39391.93.

References

[1] M. Abramovici and C. Stroud, No-overhead BIST for FPGAs, in: *Proc. 1st IEEE Int. On-line Testing Workshop* (1995) 90–92.

[2] E.R. Banks, Universality in cellular automata, in: *IEEE Conf. Record of 11th Ann. Symp. on Switching and Automata Theory* (1970) 194–215.

[3] A.W. Burks, ed., *Essays on Cellular Automata* (Univ. of Illinois Press, Urbana, 1970).

[4] J.Byl, Self-reproduction in small cellular automata, *Physica D* 34 (1989) 295–299.

[5] E. Codd, *Cellular Automata* (Academic Press, New York, 1968).

[6] H. de Garis, Evolvable hardware, in: *Proc. Artificial Neural Nets and Genetic Algorithms* (1993) 441–449.

[7] S. Durand and C. Piguet, FPGA with self-repair capabilities, in *Proc. FPGA'94, 2nd Int. ACM/SIGDA Workshop on Field-Programmable Gate Arrays* (1994) 1–6.

[8] R.A. Freitas and W.P. Gilbreath, eds, Advanced Automation for space Missions, (NASA *Conference Publication 2255*, 1982).

[9] S.F. Gilbert, *Developmental Biology*, 3rd ed. (Sinauer Associates, Sunderland, MA, 1991.

[10] M. Goeke, BIODULE 2: documentation technique, Tech. Rep., Logic Systems Laboratory, Swiss Federal Institute of Technology, Lausanne, 1995.

[11] D. Hofstadter, *Godel, Escher, Bach* (Basic Books, New York, 1979).

[12] J.G. Kemeny, Man viewed as a machine, *Scientific American* 192 (1995) 58–67.

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

[14] C. Lee, Synthesis of a cellular computer, in: J.J. Tou, ed., *Applied Automata Theory* (Academic Press, London, 1968) 217–234.

[15] D. Madon, BIODULE 2: description et utilisation, Tech. Rep., Logic Systems Laboratory, Swiss Federal Institute of Technology, Lausanne, 1995.

[16] D. Mange, *Microprogrammed Systems: An Introduction to Firmware Theory* (Champman & Hall, London, 1992).

[17] D. Mange, Teaching firmware as a bridge between hardware and software, *IEEE Trans. Education* 36 (1) (1993) 152–157.

[18] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti and S. Durand, Embryonics: A new family of coarse-grained field-programmable gate array with self-repair and self-reproducing properties, in: E. Sanchez, M. Tomassini, eds., *Towards Evolvable Hardware, Lecture Notes in Computer Science* (Springer, Berlin, 1996) 197–220.

[19] D. Mange, E. Sanchez, A Stauffer, G. Tempesti, S. Durand, P. Marchal and C. Piguet, Embryonics: A new methodology for designing field-programmable gate arrays with self-repair and self-reproducing properties, Tech. Rep. 95/152, Computer Science Department, Swiss Federal Institute of Technology, Lausanne, October 1995, submitted.

[20] P. Marchal and A. Stauffer, Binary decision diagram oriented FPGAs, in: *Proc. FPGA'94, 2nd Int. ACM/SIGDA Workshop on Field-Programmable Gate Arrays* (1994) 1–10.

[21] E.J. McCluskey, *Logic Design principles with Emphasis on Testable Semicustom Circuits* (Prentice-Hall, Englewood Cliffs, NJ, 1986).

[22] M.L. Minsky, *Computation: Finite and Infinite Machines* (Prentice-Hall, Englewood Cliffs, NJ, 1967).

[23] H.C. Morris, Typogenetics: A logic for artificial life, in: C.G. Langton, ed., *Artificial Life* (Addison-Wesley, Redwood City, 1988) 369–395.

[24] F. Nourai and R.S. Kashef, A universal four-state cellular computer, *IEEE Trans. Computers* C-24 (1975) 766–776.

[25] J.-Y. Perrier, M. Sipper and J. Zahnd, Toward a viable, self-reproducing universal computer, Physica D 97 (1966) 335–352.

[26] R. Ransom, *Computers and Embryos* (Wiley, Chichester, 1981).

[27] J.A. Reggia, S.L. Armentrout, H.-H. Chou and Y. Peng, Simple systems that exhibit self-directed replication, *Science* 259 (1993) 1282–1287.

[28] J. Signorini, Complex computing with cellular automata, *Springer Proceedings in Physics* 46 (1990) 57–72.

[29] A. Stauffer, D. Mange, E. Sanchez, G. Tempesti, S. Durand, P. Marchal and C. Piguet, Embryonics: towards new design methodologies for circuits with biological-like properties, in: *Proc. Int. Workshop on Logic and Architecture Synthesis*, Grenoble (1995) 299–306.

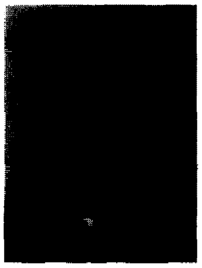
[30] G. Tempesti, A new self-reproducing cellular automaton capable of construction and computation, in: *Lecture Notes in Artificial Intelligence*, Vol. 929 (Springer, Berlin, 1995) 555–563.

[31] J. von Neumann, *The Theory of Self-Reproducing Automata* (Univ. of Illinois Press, Urbana, 1966).

[32] J.D. Watson, N.H. Hopkins, J.W. Roberts, J. Argetsinger Steitz and A.M. Weiner, *Molecular Biology of the Gene*, (Benjamin/Cummings, Menlo Park, 1987).

Daniel Mange.

André Stauffer.


Gianluca Tempesti.

Dominik Madon.