About the Universality of the Billiard ball model

Jérôme DURAND-LOSE*

LaBRI, URA CNRS 1 304,
Université Bordeaux I,
351, cours de la Libération,
F-33 405 Talence Cedex,
France.

Abstract

Block cellular automata (BCA) make local, parallel, synchronous and uniform updates of infinite lattices. In the one-dimensional case, there exist BCA with 11 states which are universal for computation. The Billiard ball model of Margolus is a reversible two-dimensional BCA which is able to simulate any two-register automaton and is thus universal. This simulation is achieved by embedding a logical circuitry with balls. The construction uses Fredkin gate and conservative and reversible logics.

1 Introduction

Cellular automata (CA for short) are well known models of synchronous and uniform processes over large arrays. They operate over infinite d-dimensional arrays of *cells*. Each cell has a *state* taken inside a finite set. At each iteration, each cell is updated according to a unique local function and the states of the cells around it. Cellular automata can simulate any Turing machine and are therefore universal for computation.

Reversibility is the ability to run backward an automaton. Reversible Turing machines were the first reversible model to be proven universal s[1]. Reversibility of CA has been studied from the sixties from a mathematical point of view, and from the seventies for a more practical trend: saving energy.

In 1970, Burks [2] conjectured that there did not exist any universal reversible CA. This conjecture was proven false in dimension two (and higher) in 1977 by Toffoli [12]. In 1992, Morita [10] proved that there also exists universal reversible CA in dimension one. In 1990, Toffoli and Margolus wrote a large survey about reversible CA [15].

Physical considerations about lattice gas lead Margolus [7] to introduce a new kind of CA, block CA (BCA), together with a practical example: the Billiard ball model (BBM). They operate over the same configurations as CA. The underlying lattice is partitioned into regularly displayed rectangular blocks. For any given partition, the corresponding updating step is made by replacing each block by its image according to a unique mapping from blocks to blocks. This replacement is repeated for various partitions in order to let information spread over the configuration.

In [14], it is claimed that since any boolean function can be implemented within the BBM, it is universal. But this implementation has two drawbacks. First, it needs constant inputs and

*jdurand@labri.u-bordeaux.fr, http://dept-info.labri.u-bordeaux.fr/~jdurand

produces garbage signals inside the configuration, and universality is not so obvious to achieve. Second, zeroes are encoded by the lack of any signal and it is impossible to distinguish between zero and no information. In this paper, we make a full construction of a simulation of any two-counter automaton with conservative and reversible logic inside the BBM.

The paper is architectured as follows. Section 2 gathers the definition of block CA and reversibility. It is shown that one-dimensional BCA are able to simulate any Turing machine and that there exists an universal one-dimensional BCA with only 11 states.

In section 3, we define the BBM and recall some basic constructions with conservative logic. Another encoding, which we call *dual*, is made by encoding the value of a bit by the position of a signal. Let us remark that this encoding is the "double-line trick" of von Neumann as mentioned by Minsky [8, p. 69]. Both dual signals zero and one are tangible. Any function of the reversible logic can be embedded in the BBM with this encoding without garbage nor constant signals.

In section 4, we built a simulation of any two counters automaton by the BBM and proved rigorously that the BBM is universal.

2 Definitions

The elements of $\mathbb{Z}^d$ are referred as *cells*. They take their value in a finite set of *states* Q. A *configuration* is an element of $Q^{\mathbb{Z}^d}$.

2.1 Block cellular automata

Block cellular automata (block CA or BCA for short) perform parallel and uniform updates of configurations. They use regular partitions of the underlying lattice $\mathbb{Z}^d$ in blocks of size $v_1 \times v_2 \times \cdots \times v_d$. A partition is identified by an *origin* $o^i \in \mathbb{Z}^d$ as illustrated in Fig. 1.

Let V be the following subset of $\mathbb{Z}^d$:

$$V = [0, v_1 - 1] \times [0, v_2 - 1] \times \cdots \times [0, v_d - 1] \ .$$

It represents the underlying lattice of any block. The local block function t is a mapping over Q^V: $t : Q^V \to Q^V$.

The *updating step* corresponding to a partition T_{o^i} is the synchronous replacement of all the blocks by their images by a *local block function* t as depicted in Fig. 1. The update is done by making successive steps corresponding to a sequence of partitions. All the updating steps use the same local block function t.

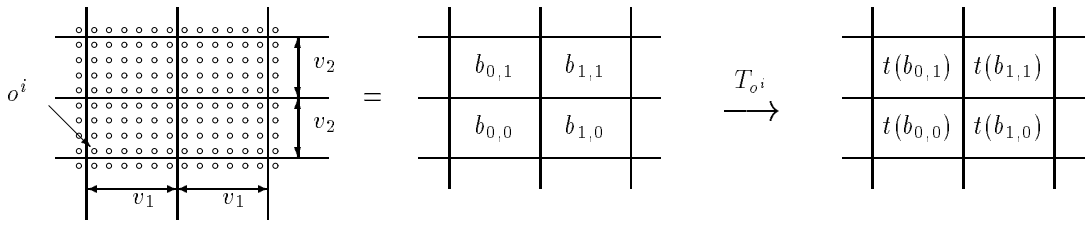


Figure 1: Updating step of origin o^i.

A BCA is totally defined by (d, Q, v, O, t) where v defines the size of the blocks and $O = (o^i)_i$ is the finite sequence of the origins o^i of the used partitions. Replacements are successively made

over various partitions identified by their origins $(o^i)_i$ as in Fig. 1, but all those updating steps uses the same size of blocks v and the same local function t. More than one partition is needed in order to let information spreads over the lattice.

The *global function* of a BCA $\mathcal{G}$ is the composition of all the updating steps:

$$\mathcal{G} = T_{o^n} \circ T_{o^{n-1}} \cdots \circ T_{o^1} \quad .$$

Definition 1 An automaton A is *reversible* if its global function is a bijection and its inverse is itself the global function of some automaton of the same kind, called the *inverse* and denoted A^{-1}.

Concerning BCA:

Lemma 2 *A BCA is reversible if its local function t is reversible, and then, its inverse is:*

$$B^{-1} \;=\; (\; Q,\; v,\; \overline{O},\; t^{-1}\;) \quad .$$

where $\overline{O}$ is the sequence of the origins in reverse order.

Since Q^V is finite, reversibility is decidable for BCA.

2.2 Universality

Definition 3 A Turing *machine* is defined by:

$$(\; \Sigma,\; Q,\; \delta,\; s_0\;)$$

where Σ is a finite set of *symbols* for the tape, Q a finite set of *states* of the machine, δ is the *transition function* and s_0 is the *initial state*.

The function δ yields the symbol to be written on the tape, the new state and the movement of the head according to the state and the read symbol:

$$\delta \;:\; Q \times \Sigma \;\rightarrow\; Q \times \Sigma \times \{-1, 1\} \cup \{\textsc{stop}\} \quad .$$

An automaton is *universal for computation* if it is able to simulate any Turing machine or is able to simulate an universal automaton. There exists universal CA [11] and universal reversible CA [12, 10].

Proposition 4 *There exists universal BCA.*

Let $M = (\Sigma, Q, \delta, s_0)$ be a universal Turing machine with m states and n symbols distinct from the states ($m = |\Sigma|$, $n = |Q|$ and $\Sigma \cap Q = \emptyset$).

Let B be the following one-dimensional BCA:

$$B \;=\; (\; Q \cup \Sigma \cup \{\;\textsc{stop}\;\},\; (2),\; (\;(0), (1)\;),\; t_M\;) \quad .$$

There are two partitions; their origins are (0) and (1). The states of B are either symbols, states of M or STOP. The local transition is defined on Fig. 2. The location of the head is encoded by the presence of a M-state ($\in Q$) in the same block.

The initial configuration and some iterations are depicted in Fig. 3. Each iteration of B makes two iterations of M. The end of the computation corresponds to the apparition of STOP.

The built BCA has minimal dimension (1), minimal width (2) and minimal number of partitions (2) to be universal. It has $m+n+1$ states. Margenstern and Rogozhin [6], proved that there exists a universal Turing machine with 5 states and 5 symbols. So:

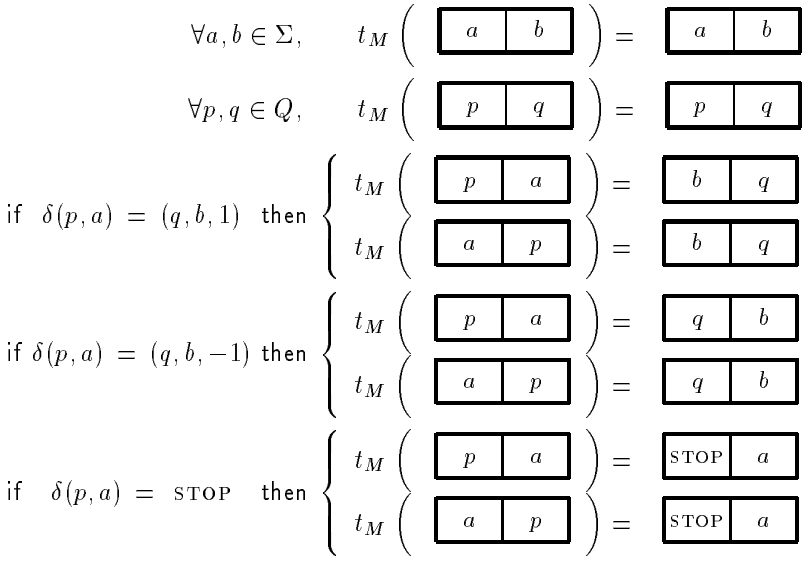


Figure 2: Local block function of B to simulate M.

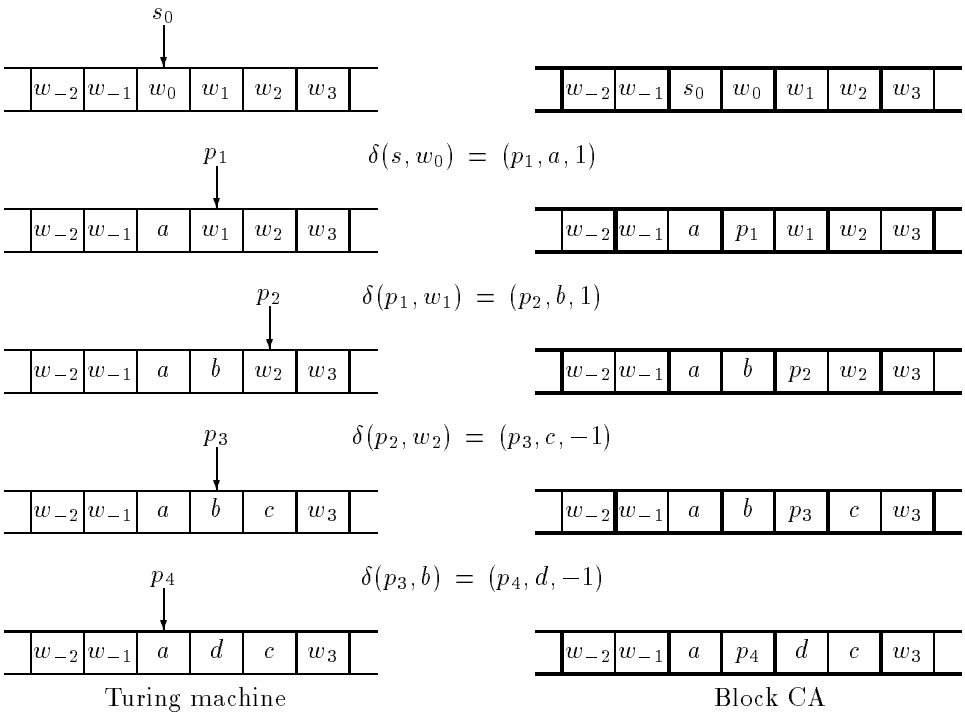


Figure 3: Simulation of a Turing machine by a BCA.

Theorem 5 *There exists a universal BCA with 11 states which is geometrically minimal.*

This number of state should be lowered.
In dimension 2, the Billiard ball model described in the next section is minimal geometrically,

has only two states and is reversible.

3 Billiard ball model

The Billiard ball model (BBM) is a two-dimensional reversible BCA.

3.1 Definition

The BBM is defined by:

$$\text{BBM} = (\ \{\_, \bullet\},\ (2,2),\ (\,(0,0),(1,1)\,),\ t_{\text{BBM}}\ )\quad .$$

There are only two states: void and a particle symbolized by a ball $\bullet$. The local block function t_{BBM} is only partially given in Fig. 4; it should be completed by rotations and symmetries. It works as follows:

- if there is only one ball, the ball moves to the opposite corner (case (iv));
- if there are two balls diagonally opposed, they move to the other diagonal (case (ii));
- in any other case, nothing changes.

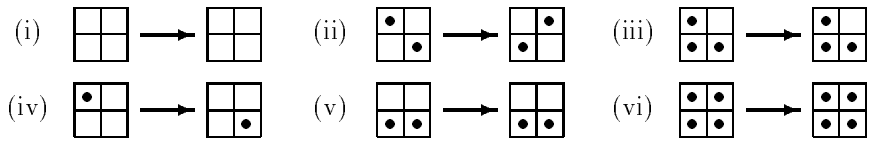


Figure 4: Definition of t_{BBM}.

The number of $\bullet$ is preserved. From lemma 2, BBM is reversible. Up to one shift, BBM is its own inverse.

Two levels of encoding of logical signals are used: the *basic* one of Toffoli and Margolus and the *dual* one. They are defined in the next subsections.

3.2 Basic encoding

This subsection is inspired by the work of Toffoli and Margolus [14].

Figure 5 depicts an example of iterations of BBM. It can be seen that the two rules (i) and (iv) of Fig. 4 are enough to create a signal: a moving ball.

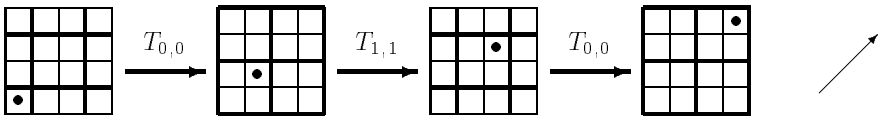


Figure 5: Ball movement.

Single balls could be used as a signals. But it should also be possible to change their directions and to make them interact with each other.

Let balls travel by pairs. If there is a motionless rectangle on their way, they bounce on it as depicted in Fig. 6. The key rule is number (ii) of Fig. 4. When two pairs meet, they are delayed and shifted aside.

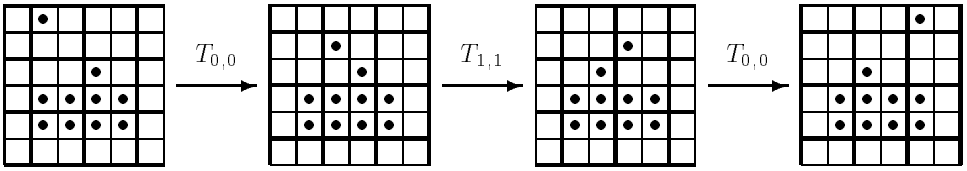


Figure 6: Reflection of a signal.

Signals are now encoded with two consecutive balls. They can move diagonally, in both directions, everywhere.

To build a delay, the path of a signal is enlarged as depicted in Fig. 7.

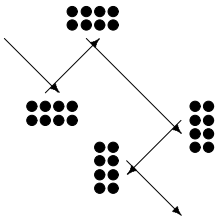


Figure 7: Delay.

Margolus and Toffoli [13] have a logical approach to computation. They encoded signal *1* with one signal and *0* with no signal. They used the so-called *conservative logic* where all gates are reversible and the number of ones (and zeroes) is preserved.

For example, the only conservative gate working with one bit is the identity and with two bits is the permutation (and the identity). To get a gate with a minimal computing ability, one has to consider a gate with three bits: the *Fredkin gate*. This gate works as follows: one bit goes through unaffected and depending on its value, the two others just pass through or are permuted. This is detailed in Fig. 8.

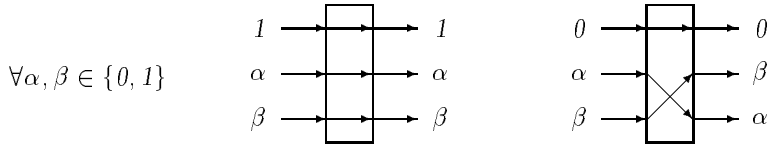


Figure 8: Fredkin gate.

In 1982, Fredkin and Toffoli [5] proved that any logical function could be built out of Fredkin gates. There are drawbacks: the function has to be embedded in another one working with more bits, constant bits have to be fed, and unwanted bits are generated. This can not be avoided with irreversible functions.

In 1990, Morita [9] proved that it is possible to built any conservative gate out of Fredkin gates. The only signals needed are zero signals which are regenerated at the end.

Toffoli and Margolus [14] proved that it is possible to simulate a Fredkin gate on the BBM with the basic encoding. Also BBM is simple, their construction is designed in two levels and takes a large amount of space and time.

Since zeroes are implemented by the lack of any signal, from these two results comes:

Lemma 6 *BBM is able to simulate any conservative logical function with basic signals without feeding nor disposal problem.*

3.3 Dual encoding

The preceding construction is interesting as long as one uses automata which works in a finite and known time. But when this time is unknown, it is impossible to distinguish between the answer *0*, *i.e.* no signal, and an unfinished computation. Additional features have to be designed to solve this problem which is particularly annoying with Turing machines which may unpredictablely stop at any time, even never.

To mind this, we use the *dual encoding* also known as the "double-line trick" of von Neumann [8]. The implementation of signals is now done by doubling the signal as depicted in Fig. 9.

s^+	s^-	$\boldsymbol{s}$
0	*0*	No signal
0	*1*	**0**
1	*0*	**1**
1	*1*	Error

Figure 9: From basic encoding to dual encoding.

A signal is now always composed of two balls. Their position indicates the value of the bit. The presence of any dual signal is explicit.

It is possible to build a Fredkin gate with the new encoding as depicted in Fig. 10. Delays are needed, but are not indicated for clarity.

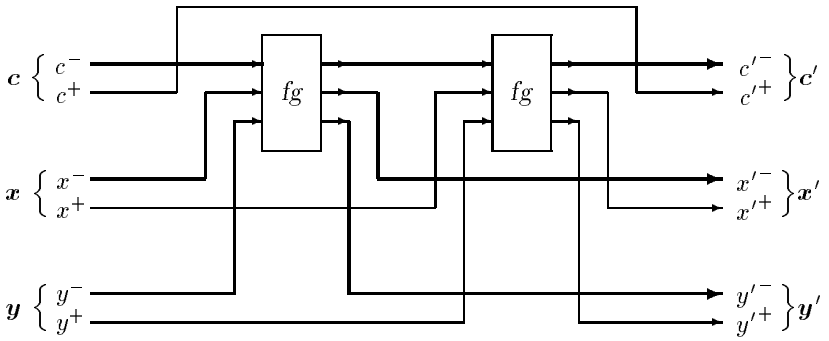


Figure 10: Fredkin gate for dual signals.

It is still possible to compute any conservative function, but it is now possible to make an autonomous `not` gate (Fig. 11). There is no risk of collision because there is only one real signal inside any dual signal.

The *reversible logic* is the restriction of the logical functions to the bijective ones.

Let $f : \{\mathbf{0}, \mathbf{1}\}^n \to \{\mathbf{0}, \mathbf{1}\}^n$ be any reversible logic function encoded with dual signals. It can also be viewed as a function $f_1 : \{(0,1),(1,0)\}^n \to \{(0,1),(1,0)\}^n$ in the basic encoding. This function f_1 is a partial but conservative definition of a function $f_2 : \{0,1\}^{2n} \to \{0,1\}^{2n}$. This function f_2 can be completed into a conservative one. From Lem. 6 comes:

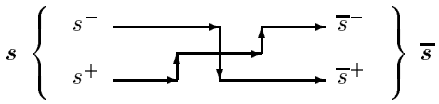


Figure 11: A `not` gate with dual signals.

Lemma 7 *BBM is able to simulate any reversible logical function with dual signals without any feeding nor disposal.*

4 Universality of the BBM

Minsky [8] introduces two-counter automata and proves that they are universal. To prove that BBM is universal, it is enough to prove that it is able to simulate any two counters automaton.

A two counters automaton is a finite automaton linked to two counters which can hold any positive integer value. The automaton can perform the following operations on the counters: add one, subtract one (zero if it is already zero), and test for nullity and branch.

In this section, we prove:

Lemma 8 *BBM is able to simulate any two counters automaton.*

The construction relies on the automaton on the one side, and on the counters on the other side.

The main automaton can be simulated by a large logical unit to which some of its output is injected back in order to store the current state. It yields orders with signals and then waits for a notification of their executions. Its state is only updated when a notification comes in. The automaton is depicted in Fig. 12.

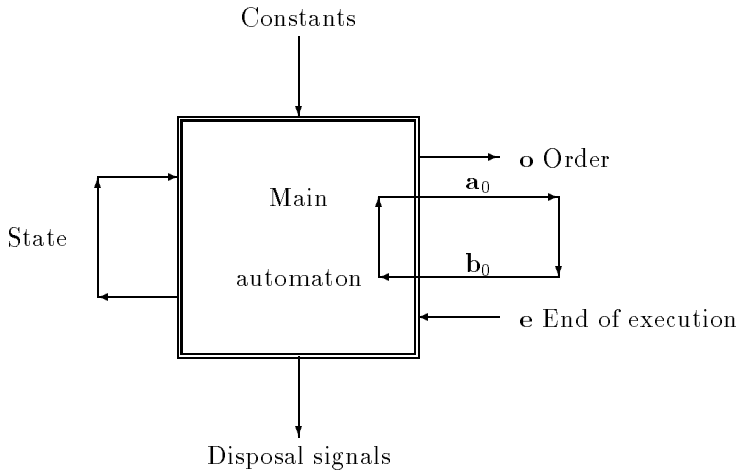


Figure 12: Main automaton.

Constant inputs and disposal signals are needed because the function of the automaton can be not invertible. The constant flow is infinite since no one can presume of the duration of the computation.

Counters are handled by an infinite line of logical *register units*. The value of any given counter is encoded in unary: $n \equiv 1^n$. Boolean values are held in signals locked between register units. These signals are updated by the units according to the orders received from the main automaton.

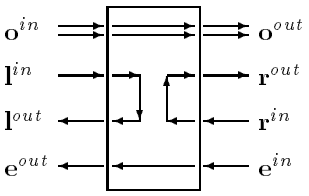
The units implement a reversible function. Thanks to Lem. 7 they can be totally autonomous. They do not bring any perturbation in the configuration.

An order $\mathbf{o}$ is encoded with two dual signals: $\mathbf{o} = (\mathbf{o}_0, \mathbf{o}_1)$. Signal $\mathbf{o}_0$ is used to state that there is an order, and $\mathbf{o}_1$ to define it. The *end of execution notification* signal $\mathbf{e}$ is equal to $\mathbf{1}$ to notify that an order was well carried out by the register, otherwise it is $\mathbf{0}$.

Counters are encoded in unary with dual signals: $n \equiv \mathbf{1}^n\mathbf{0}^\omega$. The two counters are denoted $\mathbf{A} = \mathbf{a}_0\mathbf{a}_1\ldots$ and $\mathbf{B} = \mathbf{b}_0\mathbf{b}_1\ldots$ Theses signals are stocked in an infinite line. They are nested between identical register units as depicted in figures 13 and 14.

Signals $\mathbf{a}_0$ and $\mathbf{b}_0$ are only bounced back by the automaton. Since The dual signal $\mathbf{a}_0$ ($\mathbf{b}_0$) is $\mathbf{0}$ only if $\mathbf{a}$ ($\mathbf{b}$) is 0, the main automaton can test easily whether $\mathbf{a}$ ($\mathbf{b}$) is 0. This allows the automaton to test directly the nullity of any counter.

The difficult part is the administration of the registers. The register units are all identical and communicate with the signals $\mathbf{o}$, $\mathbf{l}$, $\mathbf{r}$ and $\mathbf{e}$. Their function is depicted in Fig. 13. It should be noted that even if it is not conservative, it is are reversible as it can be proved from the table.



$\mathbf{o}^{in}$	$\mathbf{l}^{in}$	$\mathbf{r}^{in}$	$\mathbf{e}^{in}$	$\mathbf{o}^{out}$	$\mathbf{l}^{out}$	$\mathbf{r}^{out}$	$\mathbf{e}^{out}$
0,0	*	*	*	0,0	*	*	*
1,*	1	1	0	1,*	1	1	0
1,0	1	0	0	0,1	0	0	1
1,1	1	0	0	0,1	1	1	1
0,1	*	*	0	0,1	*	*	0

$*$: unchanged

Figure 13: Register unit and its logical function.

The units work by modifying $\mathbf{l}$ and $\mathbf{r}$ according to their values and the order $\mathbf{o}$. If there is an order to execute ($\mathbf{o}_0 = \mathbf{1}$), the values are modified only at the end of meaning part of the counter ($\mathbf{l} = \mathbf{1}$ and $\mathbf{r} = \mathbf{0}$). The modification is indicated by $\mathbf{o}_1$: $\mathbf{0}$ for subtraction and $\mathbf{1}$ for an addition. To subtract one, the appropriate register unit sets $\mathbf{l}$ to $\mathbf{0}$ at the appropriate time; to add one, it sets $\mathbf{r}$ to $\mathbf{1}$.

Signal $\mathbf{e}$ is $\mathbf{0}$ except when it brings a notification message (and then it is $\mathbf{1}$). It is set to $\mathbf{1}$ when a unit carries out an order. There is never more than one active order ($\mathbf{o}_t = (\mathbf{1}, .)$) or notification signal ($\mathbf{e}_t = \mathbf{1}$) in a whole configuration.

The automaton and the units are connected as depicted in Fig. 14 where it appears that the units only have $\mathbf{a}$'s, then $\mathbf{b}$'s, successively. Each time, $(\mathbf{l}^{in}, \mathbf{r}^{in})$ and $(\mathbf{l}^{out}, \mathbf{r}^{out})$ are both either $(\mathbf{a}_k, \mathbf{a}_{k+1})$ or $(\mathbf{b}_k, \mathbf{b}_{k+1})$ depending on the parity of the clock.

For the same reasons, depending on the parity of t, $\mathbf{o}_t$ only meets $\mathbf{a}$'s or $\mathbf{b}$'s, but it meets all of them.

Let us decompose the execution of an order.

If the counter $\mathbf{a}$ ($\mathbf{b}$) is null, the main automata knows it since $\mathbf{a}_0$ ($\mathbf{b}_0$) is part of its inputs. In this case, it can test and branch directly. If it wants to subtract one it just goes to the next instruction. If it wants to add one, it sets $\mathbf{a}_0$ ($\mathbf{b}_0$) to $\mathbf{1}$ and goes to the next instruction.

To make an operation $\mathbf{op}$ over $\mathbf{a}$ ($\mathbf{b}$) the automaton sends a signal $\mathbf{o} = (\mathbf{1}, \mathbf{op})$ synchronized with $\mathbf{a}_0$ ($\mathbf{b}_0$). Then it waits till it receives a $\mathbf{e}$ equal to $\mathbf{1}$ indicating that the operation was executed; then it goes on to the next operation.

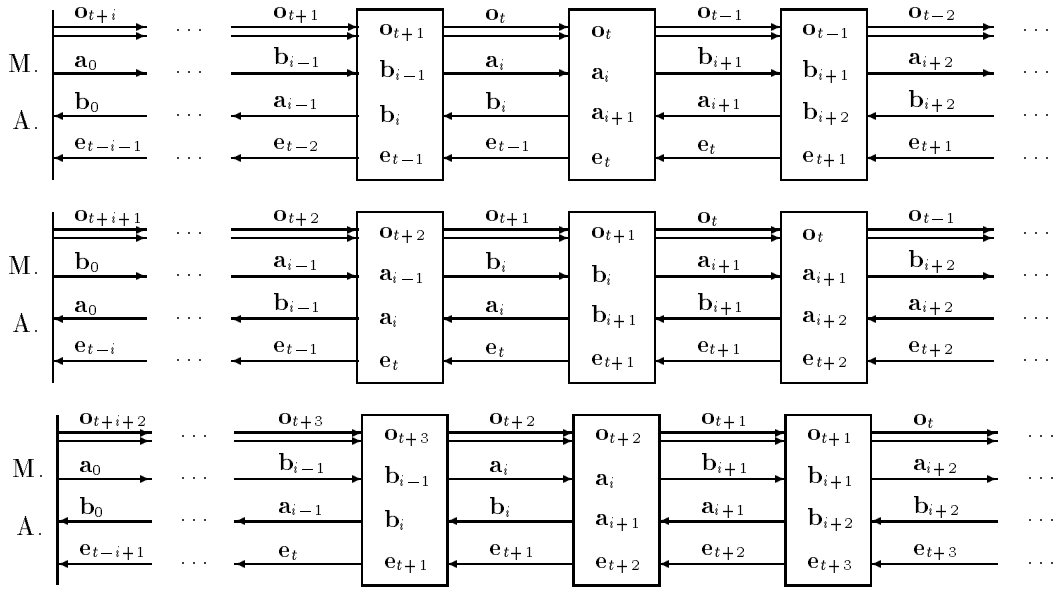


Figure 14: Automaton, units and wiring for three successive iterations.

The order $\mathbf{o}$ is treated by the registers as follows. The signal $\mathbf{o}$ travels and meets successively all the pairs $(\mathbf{a}_i, \mathbf{a}_{i+1})$ which are equal to $(\mathbf{1}, \mathbf{1})$ till it reaches the end of the meaning part of the counter, where $(\mathbf{a}_i, \mathbf{a}_{i+1})$ is equal to $(\mathbf{1}, \mathbf{0})$. If $\mathbf{op}$ is $\mathbf{1}$ (addition) then the output value $(\mathbf{a}_i, \mathbf{a}_{i+1})$ is set to $(\mathbf{1}, \mathbf{1})$, otherwise, subtraction, it is set to $(\mathbf{0}, \mathbf{0})$. The signal $\mathbf{o}$ is set to $(\mathbf{0}, \mathbf{1})$ and moves endlessly to the right. The signal $\mathbf{e}$ is set to $\mathbf{1}$ and moves backward to the main automaton and indicates that the operation was carried out. The next operation can start.

The notification time is proportional to the value of $\mathbf{a}$ ($\mathbf{b}$).

Going backward, the unit which performs the operation is defined by the meeting of the $\mathbf{e}$ which is equal to $\mathbf{1}$ and the $\mathbf{o}$ which is equal to $(\mathbf{0}, \mathbf{1})$. The performed operation is defined by the value of $(\mathbf{a}_i, \mathbf{a}_{i+1})$.

It should be noted that to build a n counters automaton, one just have to enlarge the distance between the unit and add new trapped signals.

Theorem 9 *BBM is universal.*

The universality of BBM was totally proved using reversible techniques. For the units, reversibility was designed abstractly before it was implemented.

Acknowledgment

I want to thank the referee for his remarks, especially the right reference for the dual encoding.

References

[1] C. H. Bennett. Logical reversibility of computation. IBM *Journal of Research and Development*, 6:525–532, 1973.

[2] A. Burks. *Essays on Cellular Automata*. Univ. of Illinois Press, 1970.

[3] J. O. Durand-Lose. Reversible cellular automaton able to simulate any other reversible one using partitioning automata. In LATIN *'95*, number 911 in Lecture Notes in Computer Science, pages 230–244. Springer-Verlag, 1995.

[4] J. O. Durand-Lose. *Automates Cellulaires, Automates à Partitions et Tas de Sable*. PhD thesis, LaBRI, 1996. In French.

[5] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21, 3/4:219–253, 1982.

[6] M Margenstern. Non-erasing Turing machines: A new frontier between a decidable halting problem and universality. In LATIN *'95*, number 911 in Lecture Notes in Computer Science, pages 386–397. Springer-Verlag, 1995.

[7] N. Margolus. Physics-like models of computation. *Physica D*, 10:81–95, 1984.

[8] M. Minsky. *Finite and Infinite Machines*. Prentice Hall, 1967.

[9] K. Morita. A simple construction method of a reversible finite automaton out of Fredkin gates, and its related problem. *Transactions of the* IEICE, E 73(6):978–984, June 1990.

[10] K. Morita. Computation-universality of one-dimensional one-way reversible cellular automata. *Information Processing Letters*, 42:325–329, 1992.

[11] A. R. Smith III. Simple computation-universal cellular spaces. *Journal of the Association for Computing Machinery*, 18(3):339–353, 1971.

[12] T. Toffoli. Computation and construction universality of reversible cellular automata. *Journal of Computer and System Sciences*, 15:213–231, 1977.

[13] T. Toffoli. Reversible computing. Technical Report MIT/LCS/TM-151, MIT Laboratory for Computer Science, 1980.

[14] T. Toffoli and N. Margolus. *Cellular Automata Machine - A New Environment for Modeling*. MIT press, Cambridge, MA, 1987.

[15] T. Toffoli and N. Margolus. Invertible cellular automata: A review. *Physica D*, 45:229–253, 1990.