To appear in Multiple-Valuedc Logic (journal published by Gordon and Breach Publishing Group), 1999.

Back to the Universality of the Billiard Ball Model

Jérôme O. DURAND-LOSE*†

Laboratoire I3S, CNRS UPREES-A 6070, 930 Route des Colles, BP 145, F-06 903 SOPHIA ANTIPOLIS Cedex, FRANCE.

Abstract

A full construction of the universality of the Billiard ball model, a lattice gas model introduced by Margolus in 84 is provided. The BBM is a reversible two-dimensional block cellular automaton with two states. Fredkin's gate and reversible logic can be emulated inside the Billiard ball model. They are use to embed two-counters automata, a model universal for computation.

In the one-dimensional case, there exists a universal block cellular automaton with 11 states.

1 Introduction

The Billiard ball model is a reversible cellular automaton of some sort.

Reversibility allows to run backward an automaton; information and energy are preserved. Reversible Turing machines were the first reversible model to be proven universal [1].

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* chosen inside a finite set. Each iteration, each cell is updated according to a unique local function and the states of the cells around it.

The 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

*jdurand@unice.fr, http://www.i3s.unice.fr/~jdurand.

†This work was done while the author was in the Departamento de Ingeniería Matemática, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Santiago, Chile.

universal reversible CA. This conjecture was proven false in dimension two in 1977 by Toffoli [13]. In 1992, Morita [9] proved that there also exist universal reversible CA in dimension one. Toffoli and Margolus wrote a large survey about reversible CA [15].

Physical considerations about lattice gas lead Margolus [6] to introduce a new kind of CA, *block CA* (BCA), together with a practical example: the Billiard ball model (BBM). Block CA have the same configurations as CA but the updating is done differently. The array is partitioned into regularly displayed rectangular blocks. A transition step is done by replacing each block of a given partition by its image according to a unique block transition function 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. Their construction uses *conservative logic* (reversible gates with the same number of ones in the input and in the output). But this implementation has two drawbacks. First, it needs constant inputs and produces garbage signals inside the configuration; 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-counters automaton, a universal model introduced by Minsky [7], by embedding *reversible logic* inside the BBM. With our encoding, both zero and one signals are tangible.

The definition of block CA and reversibility are gathered in section 2. It is shown that one-dimensional BCA are able to simulate any Turing machine and that there exists a universal one-dimensional BCA with 11 states.

In section 3, we recall the definition of the BBM and basic constructions with conservative logic as presented by Margolus. 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 [7, p. 69]. Any function of 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 and proved rigorously that the BBM is universal.

2 Definitions

Block cellular automata operate over bi-infinite arrays of dimension d. The elements of $\mathbb{Z}^d$ are referred as *cells*. Each cell has a value chosen inside a finite set of *states* S. A *configuration* is a valuation of the whole array, *i.e.*, an element of $S^{\mathbb{Z}^d}$.

2.1 Block cellular automata

Block cellular automata (block CA or BCA for short) perform parallel and uniform updates of configurations.

Let v_1, v_2, ... v_d be strictly positive integers. Let V be the following finite sub-array of $\mathbb{Z}^d$: $V = [0, v_1 - 1] \times [0, v_2 - 1] \times \cdots \times [0, v_d - 1]$. It represents the shape of any block. The *block transition function* t is a mapping over S^V: $t : S^V \to S^V$.

A V-partition is a regular partition of the array in blocks of size V. It is defined by an *origin* $o_i \in \mathbb{Z}^d$ as illustrated in Fig. 1. The *transition step* corresponding to a partition T_{o_i} is the synchronous replacement of all the blocks by their images by the block transition function t as depicted in Fig. 1. The update is done by making successive transition steps corresponding to a sequence of partitions.

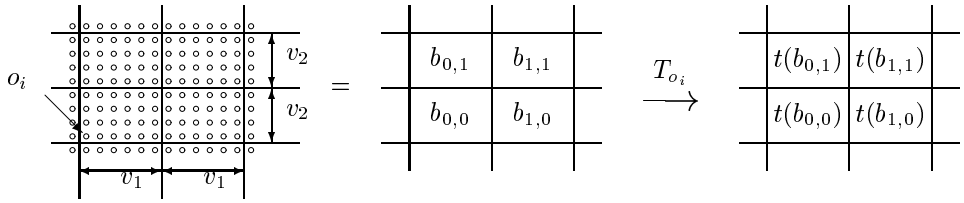


Figure 1: Transition step of origin o_i.

Replacements are successively made over various partitions identified by their origins $(o_i)_i$ (as in Fig. 1). All the transition steps use the same size of blocks V and the same block transition function t. More than one partition is needed in order to let information spread over the array.

The *global transition function* $\mathcal{T}$ maps configurations into configurations. It is the composition of all the transition steps:

$$\mathcal{T} = T_{o_n} \circ T_{o_{n-1}} \cdots \circ T_{o_1} \ .$$

A BCA is totally defined by (d, S, V, O, t) where $O = (o_i)_i$ is the finite sequence of the origins o_i of the partitions.

Definition 1 An automaton A is *reversible* if its global transition function is a bijection and its inverse is itself the global transition 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 block transition function t is reversible, and then, its inverse is:*

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

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

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

Remark A BCA is not exactly a cellular automaton since it does not commute with all the shifts. Yet, it commutes with all $(\lambda_0 v_0, \lambda_1 v_1, \ldots \lambda_d v_d)$-shifts ($\lambda_i \in \mathbb{Z}$). At block scale, a BCA is indeed a cellular automaton.

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 transition 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:

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

An automaton is *universal for computation* if it is able to simulate any Turing machine or is able to simulate a universal automaton. There exists universal CA [12] and universal reversible CA [13, 9].

Proposition 4 *There exists universal BCA.*

Let $M = (\Sigma, Q, \delta, s_0)$ be a universal Turing machine with distinct m states and n symbols ($m = |\Sigma|$, $n = |Q|$ 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 an halting symbol STOP. The local

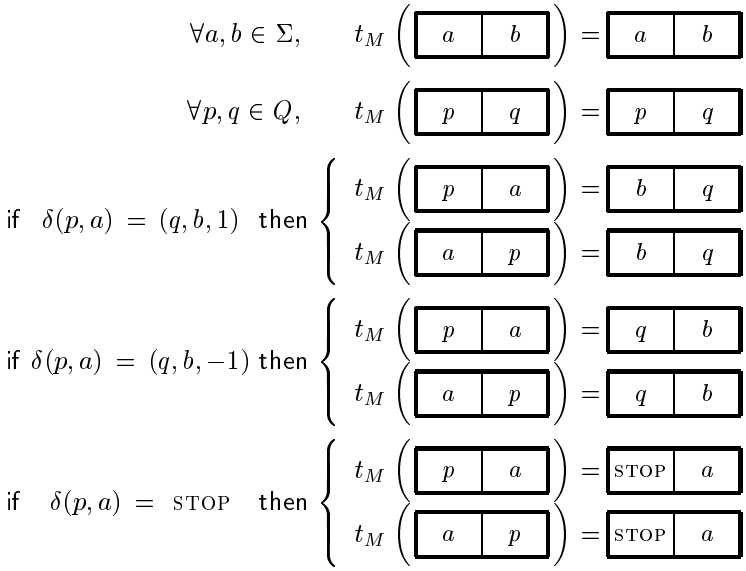


Figure 2: Block transition function of B to simulate M.

transition is defined on Fig. 2. The location of the head is encoded by the presence of a M-state (in Q) together with a M-symbol (in Σ) in one block.

The initial configuration and some iterations are depicted in Fig. 3. Each transition step corresponds to one iteration of M. Each iteration of B makes two iterations of M. The end of the computation corresponds to the apparition of state 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. Rogozhin [10, 11] proved that there exists a universal Turing machine with 5 states and 5 symbols. It comes immediately that:

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

In dimension 2, the Billiard ball model described in the next section is minimal geometrically, has only two states and is reversible. It is minimal for every parameter but for its dimension.

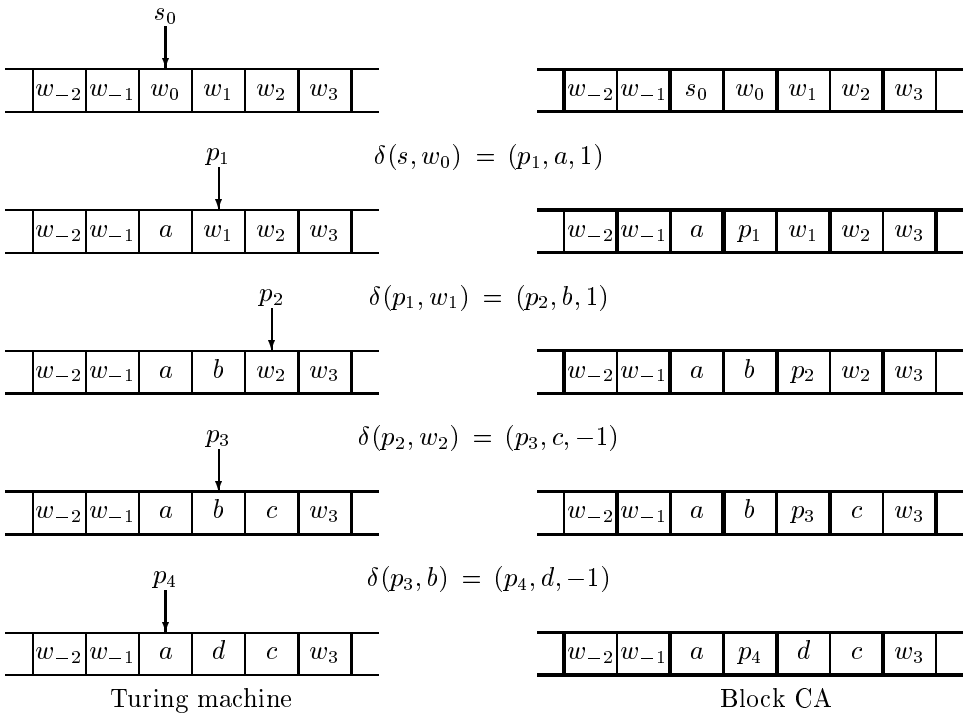


Figure 3: Simulation of a Turing machine by a BCA.

3 Billiard ball model

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

The number of $\bullet$ is preserved. The block transition function t_{BBM} is reversible, from lemma 2, the BBM is reversible. Up to one shift, the BBM is its own inverse.

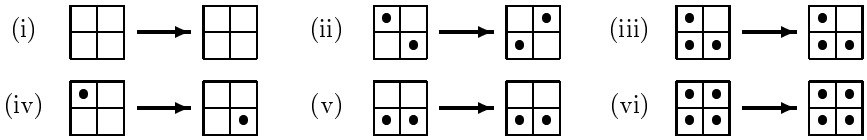


Figure 4: Definition of t_{BBM}.

To prove the ability of the BBM to compute, we implement logical wiring. Two levels of encoding of binary signals are used: the *basic* one of Toffoli and Margolus and the *dual* one. They are defined in the next subsections.

3.1 Basic encoding

This subsection is inspired by the works of Fredkin, Margolus and Toffoli [5, 6, 14].

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

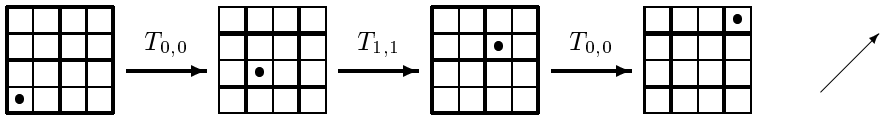


Figure 5: Ball movement.

Single balls could be used as signals. But it should be possible to change their directions and to make them interact with each other. To do this, let balls travel by pairs, one behind the other. If there is a motionless rectangle on their way, they bounce on it as depicted in Fig. 6. The key rule is the rule (ii) of Fig. 4.

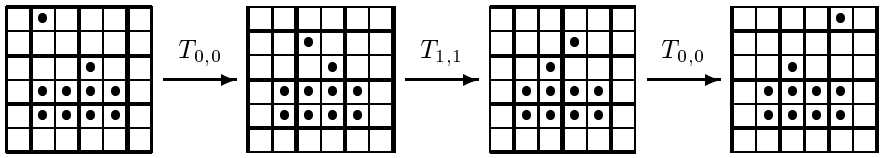


Figure 6: Reflection of a signal.

Signals are now encoded with two consecutive balls. They can move diagonally, in both directions, everywhere. With reflections of signals it is easy to build delays: the path of a signal is enlarged as depicted in Fig. 7.

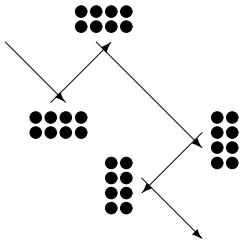


Figure 7: Delay.

When signals meet on the side, they go in the same two directions but the are shifted one diagonal backward as depicted in Fig. 8. The dotted line is the way they would have follow if only one signal would have been present.

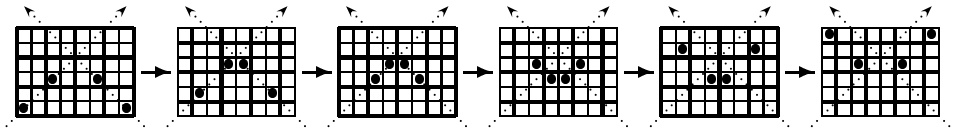


Figure 8: Signals collision.

Signal *1* is encoded with one signal and *0* with no signal.

In *conservative logic*, all gates are reversible and the number of ones (and zeroes) is preserved. It is not possible to duplicate a signal nor to discard it. 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 three bits gate: 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 as represented in Fig. 9.

$\forall \alpha, \beta \in \{0, 1\}$

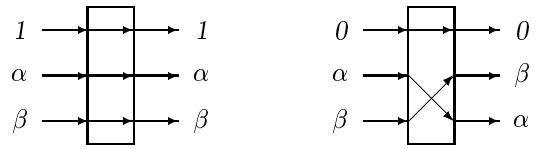


Figure 9: Fredkin gate.

Fredkin gates can be simulated on the BBM with the basic encoding [14]. Also the BBM is simple, their construction is designed in two levels and takes a large amount of space and time. Morita [8] 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. Since zeroes are implemented by the lack of any signal:

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

Any binary function f can be implemented with conservative logical functions. It is done using a larger conservative function φ in the following way. Constant bits c are added to the f-entry x to form a φ-entry $x.c$. The output of φ is the output of f together with bits which are only there to guaranty that φ is conservative.

This technique has drawbacks: constant bits have to be provided, and unwanted bits are generated (and have to be disposed off in some way). This can not be avoided with irreversible functions. The BBM does not allow to create nor to remove balls.

3.2 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 provided to solve this problem which is particularly annoying with Turing machines which may unpredictably stop at any time.

To mind this, we use the *dual encoding* also known as the "double-line trick" of von Neumann. This is done by doubling the signal as depicted in Fig. 10. A signal is now always composed of two consecutive balls. Their position indicates the value of the bit. The presence and value of any dual signal are explicit.

It is possible to build a Fredkin gate with the new encoding as depicted in Fig. 11. Some delays are needed, but they are not indicated for clarity.

It is still possible to compute any conservative function, but it is now possible to make an autonomous `not` gate (Fig. 12). There is no risk of collision because there is only one real signal for any dual signal.

We call *reversible logic* 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 logical function encoded with dual signals. It can also be viewed as a function f_1 :

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

Figure 10: From basic encoding to dual encoding.

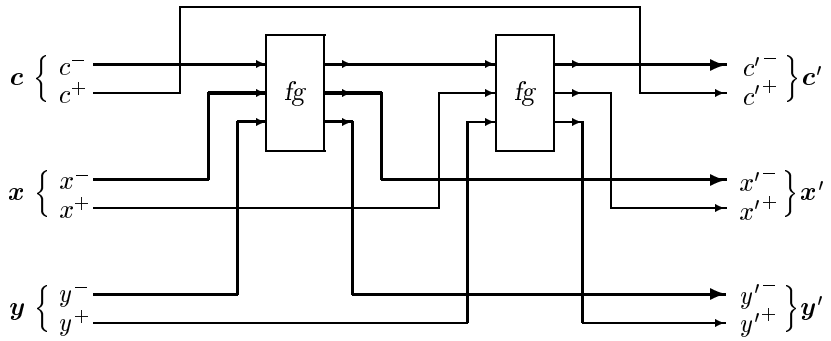


Figure 11: Fredkin gate for dual signals.

$\{(0,1),(1,0)\}^n \rightarrow \{(0,1),(1,0)\}^n$ in the basic encoding. Function f_1 is a partial definition of a conservative function $f_2 : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$. From Lem. 6 comes:

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

4 Universality of the BBM

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

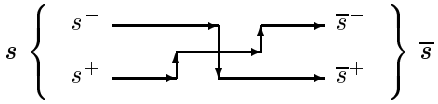


Figure 12: A `not` gate with dual signals.

zero) and test for nullity and branch.

Minsky proved that there exist universal two-counters automata [7]. To prove that the BBM is universal, it is enough to show that it is able to simulate any two-counters automaton.

Lemma 8 *The 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 automaton can be simulated by a large logical unit. The state of the automaton is encoded in a part of the output which is fed back to the input. To perform an action on the counters, the automaton unit send the corresponding *order signal* to the counters. The state remains the same until a notification of the execution of the order is received. Then the state is changed and a new cycle starts. The automaton is depicted in Fig. 13. Constant input has to be provided and garbage output is produced because the function of the automaton can be irreversible. The flow of constants is infinite since no one can presume of the duration of the computation.

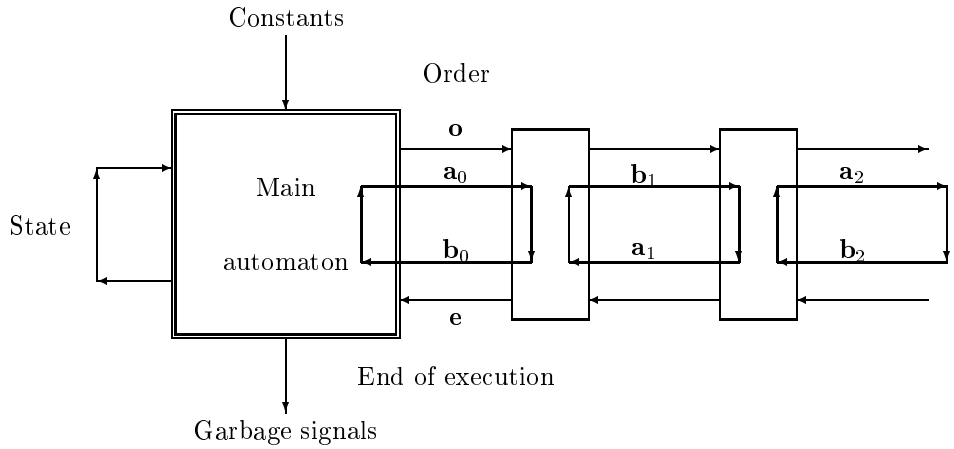


Figure 13: Main automaton and two register units.

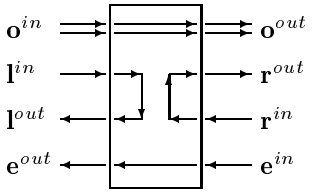
An *order* $\mathbf{o}$ is encoded with two dual signals: $\mathbf{o} = (\mathbf{o}_0, \mathbf{o}_1)$. Signal $\mathbf{o}_0$ is used to state that there is an order, and $\mathbf{o}_1$ to define it. The *end of execution notification* signal $\mathbf{e}$ equals $\mathbf{1}$ to notify that an order was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

Counters are stored and handled inside a unique infinite line of logical *register units*. The value of 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 $\mathbf{b} = \mathbf{b}_0\mathbf{b}_1\ldots$ The corresponding signals are locked between consecutive register units as depicted in figures 13 and 15. The register units update the values of the signals according to the orders received from the automaton.

Signals $\mathbf{a}_0$ and $\mathbf{b}_0$ are not nested between two register units but between the automaton and the first register unit. Since signal $\mathbf{a}_0$ ($\mathbf{b}_0$) equals $\mathbf{0}$ only if $\mathbf{a}$ ($\mathbf{b}$) equals 0, the automaton can test easily whether $\mathbf{a}$ ($\mathbf{b}$) is 0. This allows the automaton to test directly the nullity of any counter.

The crucial part is the administration of the counters. The register units are all identical and communicate with the signals $\mathbf{o}$, $\mathbf{l}$, $\mathbf{r}$ and $\mathbf{e}$. The function of a register unit is defined by the table of Fig. 14. It should be noted that even if it is not conservative, it is reversible as it can be proved from the table. The last two lines of the table look like they can be merge into a rule like "if $\mathbf{o}_0$ is $\mathbf{0}$ then nothing changes" but the function wouldn't be one to one anymore. Each register unit implements a reversible function. Thanks to Lem. 7, register units can be totally autonomous. They do not bring any perturbation in the configuration.



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

– : don't care * : unchanged

Figure 14: Register unit and the corresponding logical function.

A register unit works 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 $\mathbf{r} = \mathbf{0}$, second and third lines of Fig. 14). 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 $\mathbf{0}$; to add one, it sets $\mathbf{r}$ to $\mathbf{1}$.

Signal $\mathbf{e}$ is $\mathbf{0}$ except when it brings the notification that an order was executed (and then it is $\mathbf{1}$). It is set to $\mathbf{1}$ by a register unit which 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 register units are connected as depicted in Fig. 15. Each unit only has $\mathbf{a}$'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. All the same, depending on the parity of t, $\mathbf{o}_t$ meets only **a**'s or only **b**'s, but it meets all of them as it can be seen on Fig. 15.

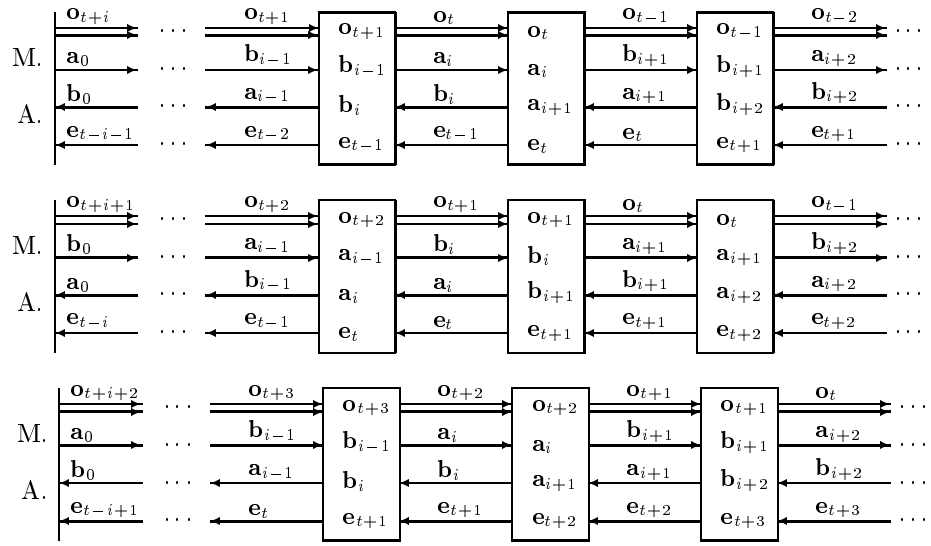


Figure 15: Automaton, units and wiring for three successive iterations.

Let us decompose the execution of an order.

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

To make an operation **op** (**op** is **1** for addition, **0** for subtraction) over **a** (**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 **e** equal to **1** indicating that the operation was performed; then it goes on to the next operation.

The order **o** is treated by the register units as follows. The signal **o** travels and meets successively all the pairs $(\mathbf{a}_i, \mathbf{a}_{i+1})$ which are equal to $(\mathbf{1}, \mathbf{1})$ until it reaches the end of the meaning part of the counter ($(\mathbf{a}_i, \mathbf{a}_{i+1})$ equals $(\mathbf{1}, \mathbf{0})$). If **op** is **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 **o** is set to $(\mathbf{0}, \mathbf{1})$ and moves endlessly to the right. The signal **e** is set to **1** and moves back to the automaton and indicates that the operation was carried out. The next operation can start.

The execution time is proportional to the value of **a** (**b**).

Going backward in time, the register unit which performs the operation is defined by the meeting of the **e** equal to **1** and the first **o** 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 register units and add new trapped signals.

Theorem 9 *The BBM is universal.*

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

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. Durand-Lose. Reversible cellular automaton able to simulate any other reversible one using partitioning automata. In *LATIN '95*, number 911 in LNCS, pages 230–244. Springer-Verlag, 1995.

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

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

[8] 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.

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

[10] Yu. V. Rogozhin. Seven universal Turing machines. In *Systems and theoretical programming*, number 63 in Mat. Issled., pages 76–90. Academia Nauk Moldavskoi SSR, 1982. in Russian.

[11] Yu. V. Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996.

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

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

[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.