1 Computing Inside the Billiard Ball Model

Jérôme Durand-Lose

The chapter studies relations between billiard ball model, reversible cellular automata, conservative and reversible logics and Turing machines. At first we introduce block cellular automata and consider the automata reversibility and simulation dependencies between the block cellular automata and classical cellular automata. We prove that there exists a universal, i.e. simulating a Turing machine, block cellular automaton with eleven states, which is geometrically minimal. Basics of the billiard ball model and presentation of an information in the model are discussed then. We demonstrate how to implement ball movement, reflection of a signal, delays and cycles, collision of signals in configurations of the cellular automaton with Margolus neighborhood. Realizations of Fredkin gate and NOT gate with dual signal encoding are offered. The rest of the chapter deals with a Turing and an intrinsic universality, and uncomputable properties of the billiard ball model. The Turing universality is proved via simulation of a two-counter automaton, which itself is Turing universal. We demonstrate that the billiard ball model is intrinsically universal, or complete, in a class of reversible cellular automata, i.e. the model can simulate any reversible automaton over finite or infinite configurations. A novel notion of space-time simulation, that employs whole space-time diagrams of automaton evolution, is brought up. It is proved that the billiard ball model is also able to space-time simulate any (ir)reversible cellular automaton. Since the billiard ball model possesses the Turing computation power we can project a Turing machine's halting problem to development of cellular automaton simulating the billiard ball model. Namely, we uncover a connection between undecidability of computation and high unpredictability of configurations of the billiard ball model.

The whole field of collision-based computing was initiated by a billiard ball model. It is a well-known model because it clearly shows how to perform a universal computation by a simple reversible cellular automaton machine in an intuitively appealing way.

Reversibility allows to run evolution of a system, an automaton in our case, backward. Both information and energy are preserved during a computation implemented by a reversible machine. A first published result on the reversible universal computing, dealing with reversible Turing machines, is dated forty years ago *universal* [12,2]. A universality is a capability to compute any computable function, in the recursion theory, given its definition and its arguments in some encoding.

Cellular automata (CA for short) are a well known model of synchronous and uniform processes over large arrays. They operate over infinite d-dimensional arrays of finite state automata, or *cells*. Each cell takes a finite number of *states*. All cells of an array update their states in discrete time by the same cell state transition rule, or a local function.

A reversibility of CA has been studied from the 1960s from a mathematical point of view, and from the 1970s for a more practical trend: saving energy. In 1970, Burks [3] conjectured that there did not exist any universal reversible CA. This hypothesis was disproved for two-dimensional CA in 1977 by Toffoli [30]. In 1992, Morita [18] proved that there also exist universal reversible one-dimensional CA. We would like to attract readers' attention to a comprehensive survey on reversible CA by Toffoli and Margolus [29], which is a rich source of references.

CA models of lattice gas and relevant physical considerations lead Margolus [13] to introduce the *billiard ball model* (BBM). The BBM is not exactly a CA. It has the same configurations as a CA— there are only two states — but cell state updating is done differently. The array is partitioned into regularly displayed rectangular 2 by 2 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 done twice, with distinct partitions, in order to let information spread over the array.

The BBM can easily be generalized: any finite set of states, any size of blocks, any number of partitions, any function from blocks to blocks. This variation of CA is called a *block CA* (BCA), partitioned CA or CA with the Margolus neighborhood. In this chapter, we use the term *block CA* .

In [28], it is claimed that since any Boolean function can be implemented within the BBM, the BBM is universal. The construction employs *conservative logic*, reversible gates of which have the same number of ones in the input and in the output. However, conservative logic based implementation has two drawbacks. First, it needs constant inputs and produces garbage signals inside the configuration; universality is not so obvious to achieve. Second and most important, zeroes are encoded by the lack of any signal and it is impossible to distinguish, without employing additional "clock" signal, between no information and a zero result.

In this chapter, after presenting BCA and a small universal one-dimensional BCA, we show how any two-counter automaton, a universal model introduced by Minsky [14], is simulated by BCA. The simulation is implemented by embedding *reversible logic*, gates of which are reversible but the number of ones is not necessarily preserved, into the BBM. The encoding keeps both zero and one signals tangible.

Using partitioned CA, a class of CA introduced by Morita et al. [15]–[21], we also prove that the BBM can simulate any reversible CA over any configuration, finite or not [5]. Such an ability to simulate a reversible CA is

called an *intrinsic universality*. The difference from the usual Turing universality, in the CA context, is that, during a single transition step, an infinite configuration can be completely changed. This clearly falls outside of Turing universality. There are reversible CA that are able to simulate any CA with one less dimension [30]. It is still an open problem whether the BBM or any reversible CA can simulate all CA, including irreversible CA over infinite configurations, of the same dimension.

We present a new notion of universality, *space-time universality*, which does not rely on automata configurations but whole orbits of the automata evolution or space-time diagrams. We show that the BBM is space-time intrinsically universal among the whole class of CA. Definitions and results from [6] are used to achieve this.

This chapter ends with some undecidability results like the passage of a ball at some point or the apparition of a given pattern inside the configuration.

The chapter is structured as follows. Definitions of block cellular automata (BCA) and reversibility, as well as connections with classical cellular automata are gathered in Sect. 1.1. In Sect. 1.2, 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 Sect. 1.3, we recall the definition of the billiard ball model (BBM) and basic constructions with conservative logic as they are originally presented by Margolus. Another version of encoding, *dual encoding*, is made by encoding the value of a bit by the position of a signal; this encoding is the "double-line trick" of von Neumann as mentioned by Minsky [14, p. 69]. Any function of the reversible logic can be embedded in the BBM with this encoding, garbage or constant signals.

In Sect. 1.4, we simulate a two-counters automaton in the BBM and rigorously prove that the BBM is universal.

Partition CA are defined in Sect. 1.5. Then the intrinsic universality of the BBM among reversible CA is proved. In the same section, space-time simulation is presented and the space-time intrinsic universality of the BBM is demonstrated.

In Sect. 1.6, some types of BBM behavior are proved to be undecidable; the undecidability leads to high unpredictability of the way along which a configuration may develop.

1.1 Definitions

Block cellular automata, like "conventional" cellular automata (see Subsect. 1.1.4) and partitioned cellular automata (see Subsect. 1.5.1), operate over bi-infinite arrays of dimension d. The elements of $\mathbb{Z}^d$ are referred as *cells*. Each cell has a value chosen from a finite set of *states* S. A *configu-*

ration is a valuation of the whole array, *i.e.*, an element of $S^{\mathbb{Z}^d}$. The set of configurations is denoted $\mathcal{C}$ ($\mathcal{C} \equiv S^{\mathbb{Z}^d}$).

1.1.1 Block Cellular Automata

A *block cellular automaton* (block CA or BCA for short) performs local, parallel and uniform updates of configurations. Updating is done by partitioning the configuration into rectangular blocks and independently calculating each block's next state. The update may be repeated several times for various partitions.

All blocks have the same shape $V = [0, v_1 - 1] \times [0, v_2 - 1] \times \cdots \times [0, v_d - 1]$, which is a finite sub-array of $\mathbb{Z}^d$; and, v_1, v_2, ... v_d are positive integers. The *block transition function* t, used to update blocks, is a mapping over S^V, $t : S^V \to S^V$.

A V-partition is a regular partition of the array into blocks of size V. It is defined by an *origin* $o_i \in \mathbb{Z}^d$ as illustrated in Fig. 1.1. The *transition step* corresponding to a partition T_{o_i}, and the block transition function t, is the synchronous replacement of all the blocks by their images by t as depicted in Fig. 1.1.

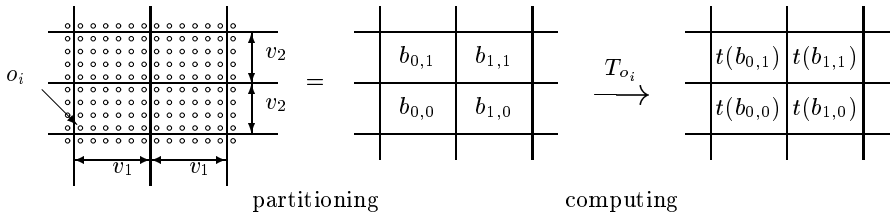


Fig. 1.1. Transition step of the origin o_i.

Replacements are successively made over various partitions identified by the sequence of their origins $O = (o_i)_{1 \leq i \leq n}$. All the transition steps use the same size of blocks V and the same block transition function t. Since the blocks are updated independently, more than one partition is needed in order to let information spread over the array.

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

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

A block cellular automaton BCA is completely defined by (d, S, V, t, n, O).

1.1.2 Reversibility

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 an *inverse* and denoted A^{-1}.

Reversibility can be tested and the inverse can be built very easily as stated by the following lemma:

Lemma 1. *A BCA B is reversible if and only if its block transition function t is reversible, and then, its inverse is:*

$$B^{-1} = \left(d,\ S,\ V,\ t^{-1},\ n,\ \overline{O} \right) ,$$

where $\overline{O}$ is the sequence of the origins in reverse order ($\overline{O} = (o_{n+1-i})_{1 \leq i \leq n}$).

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

1.1.3 Simulation

In this chapter, by simulation, we mean the following:

Definition 2. For any two functions $f : F \to F$ and $g : G \to G$ the function g *simulates* the function f (in real time) if and only if there exists two encoding functions $\alpha : F \to G$ and $\beta : G \to F$, recursive, space and time inexpensive compared to f and g, such that: $f^n = \beta \circ g^n \circ \alpha$ for any natural n. The function g can be used instead of f for iterating the global function.

A cellular automaton simulates another cellular automaton if and only if a global function of the first automaton simulates the global function of the second one.

A simulation corresponds to the commuting diagram of Fig. 1.2. The simulation is therefore a transitive relation.

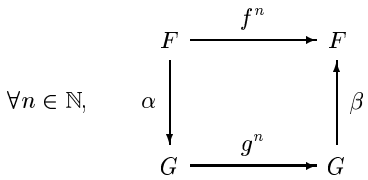


Fig. 1.2. g simulates f.

The encoding and decoding functions α and β are not always given explicitly, but their meaning is usually clear from the context.

1.1.4 Cellular Automata

Cellular automata (CA) use the same set of configurations as BCA. They also perform local, parallel and uniform updates of configurations, but the updating is done differently.

A cellular automaton is defined by $(d, S, \mathcal{N}, f)$. The *neighborhood* $\mathcal{N}$ is a finite subset of $\mathbb{Z}^d$. It represents the coordinates (relatively to the cell) of the cells whose states are to be considered for updating. The *local function* $f : S^{\mathcal{N}} \to S$ maps the states of cell neighbors into the cell's next state. The local function is said to compute the new state of each cell.

The *global function* $\mathcal{G} : \mathcal{C} \to \mathcal{C}$ maps configurations into configurations as follows:

$$\forall c \in \mathcal{C},\ \forall x \in \mathbb{Z}^d,\ \mathcal{G}(c)_x \ = \ f\left((c_{x+\nu})_{\nu\in\mathcal{N}}\right) \ .$$

A new state of a cell depends only on its neighbors' states as depicted by Fig. 1.3 where $\mathcal{N} = \{-1, 0, 1\}$.

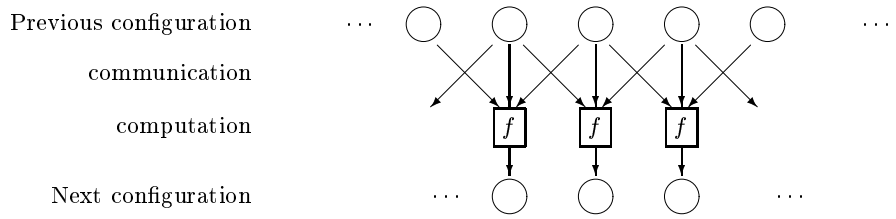


Fig. 1.3. Cell state updating in one-dimensional CA.

Block CA and partitioning CA (defined in Sect. 1.5) are sub-classes of CA. They are often considered just as CA, with a special constraint on their definitions.

1.1.5 Relations with Classical Cellular Automata

BCA are CA In a BCA, the next value of a cell depends on the values of its neighboring blocks, this is a local updating. Yet the value is highly dependent on the position of the cell inside the blocks of the various partitions.

Let us consider the blocks of the first partition of the BCA as a cell, with a larger but still finite set of states. The composition of block transitions only consider blocks which are around the block. Thus, any BCA can be expressed as a CA at the block scale. Since this is an identification, if the BCA is reversible then the corresponding CA is reversible.

In fact, one of the reasons to introduce the billiard ball model and BCA was to design as simple as possible a two-dimensional CA which is reversible and universal. It was a tricky problem since reversible CA are complicated to build and to manipulate, and even worse, as Kari proved in [10], reversibility is not decidable for CA of dimension two and above. In comparison, reversibility of BCA is easily checkable in any dimension as stated in Lemma 1.

Representation of CA as BCA It is possible to embed CA in BCA if more states are allowed during the computation. One can also represent a CA which is known to be reversible as a reversible BCA [4,7].

One open problem is whether, and how, this representation can be made without increasing the number of states. The answer is positive in dimensions one and two *if* a composition with a partial translation is allowed [11].

1.2 Universality of One-Dimensional Block CA

How good are BCA for computing? How complex can a space-time diagram of a BCA be?

In the previous subsection we found that any CA can be simulated by a BCA, so that one-dimensional BCA can carry out any computation and its space-time diagrams can be very complex (see e.g. the four complexity classes of Wolfram [31]).

In this section we present a simulation of Turing machines by BCA and prove the existence of a universal one-dimensional BCA with only 11 states.

Recall, that Turing machines are simply finite automata, or processors, that can read and write on an infinite tape, or a memory. The machines capture a notion of computability and they are able to carry out any computation. More information on computability and Turing machines can be found in many textbook, e.g. [25].

Definition 3. A Turing *machine* is defined by the tuple $(\ \Sigma,\ Q,\ \delta,\ s_0\ )$, where Σ is a finite set of *symbols* for the tape, Q a finite set of *states* of the machine, δ is a *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:

$$\delta\ :\ Q \times \Sigma\ \rightarrow\ Q \times \Sigma \times \{\leftarrow, \rightarrow\} \cup \{\textsc{stop}\}\ .$$

An automaton is *universal (for computation)* if it is able to simulate any Turing machine or is able to simulate a universal automaton. There exist universal CA [26] and universal reversible CA [30,18].

Proposition 1. *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:

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

There are two partitions; their origins are (0) and (1). The states of B are either symbols, states of M or a halting state STOP. The local transition

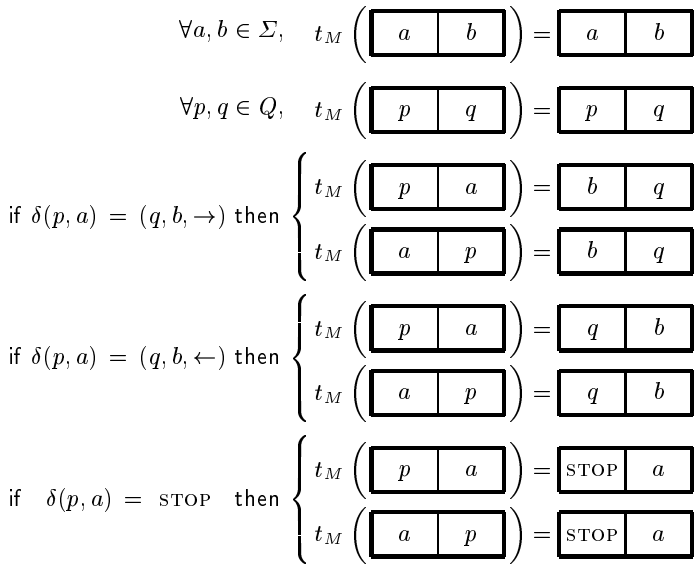


Fig. 1.4. Block transition function of B to simulate M.

is defined on Fig. 1.4. 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.

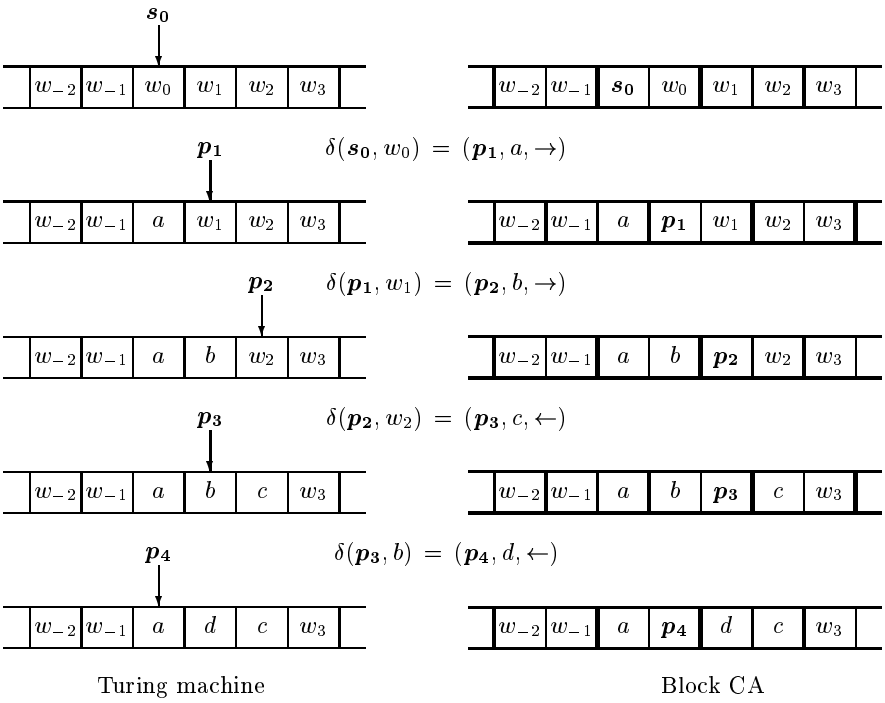
The initial configuration and some iterations are depicted in Fig. 1.5. Each transition step corresponds to one iteration of M. Each iteration of B makes two iterations of M. The end of the computation corresponds to the occurrence of the state STOP.

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. Earlier, Rogozhin [23,24] proved that there exists a universal Turing machine with 5 states and 5 symbols. It comes immediately that:

Theorem 1. *There exists a universal BCA with 11 states which is geometrically minimal.*

The extensive definition of this BCA is skipped because it is lengthy and not so relevant. Anyway, it can be constructed quite easily.

In two-dimensional space, the billiard ball model described in the next section, is also minimal geometrically, has only two states and is reversible. The model is minimal for every parameter but the dimension.



The thick lines indicate the iterated partitions.

Fig. 1.5. Simulation of a Turing machine by a BCA.

1.3 Billiard Ball Model

The billiard ball model (BBM) is a two-dimensional reversible BCA. It is defined by the following tuple:

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

There are only two states: void and a particle symbolized by a *ball*, •. The size of the blocks is 2×2. There are two partitions, one is deduced from the other by a $(1,1)$-shift. The four cells of a block are split between four blocks in the next partition.

The block transition function t_{BBM} is only partially given in Fig. 1.6; 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.

In all the figures, the iterated partitions are indicated by thick lines.

The block transition function t_{BBM} preserves the number of •. Applying twice t_{BBM} to any block, one gets the same block, i.e., $t_{\text{BBM}}^{-1} = t_{\text{BBM}}$. Since t_{BBM}

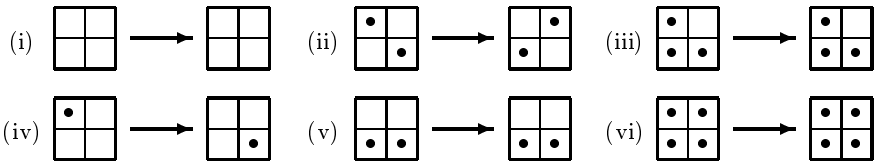


Fig. 1.6. Definition of t_{BBM}.

is reversible, from Lemma 1, the BBM is reversible. Up to a $(1,1)$-shift, the BBM is its own inverse.

Logical wiring is used to prove the computational ability of the BBM. Two levels of encoding of signals are used: the *basic* one of Toffoli and Margolus and the *dual* one. They are defined in the next two subsections.

1.3.1 Basic Encoding

This subsection is inspired by the works of Fredkin, Margolus and Toffoli [8,13,28].

Figure 1.7 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. 1.6 are enough to create a signal: a moving ball.

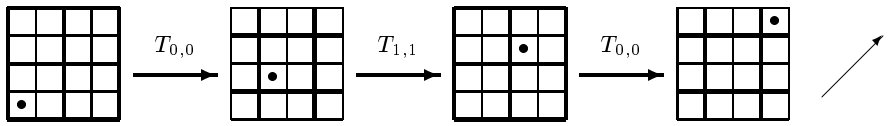


Fig. 1.7. Ball movement.

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 rectangle (which is a stable pattern) on their way, they bounce on it as depicted in Fig. 1.8. The key rule is the rule (ii) of Fig. 1.6.

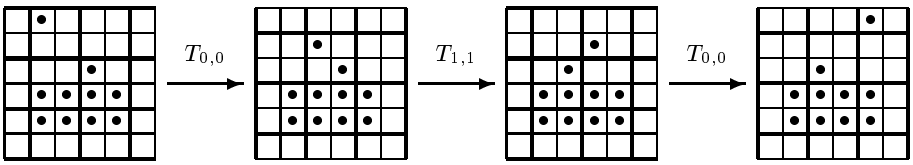


Fig. 1.8. Reflection of a signal.

Signals are now encoded with two consecutive balls. They 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 the left part of Fig. 1.9. Any even delay can be built this way.

It is also possible to built a cycle of even length by trapping a ball between four reflexive walls as indicated on the right part of Fig. 1.9.

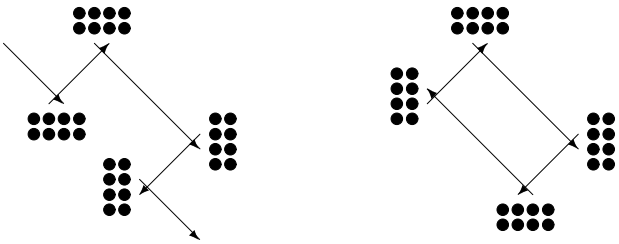


Fig. 1.9. Delay and cycle.

When signals meet on the side, they go in the same two directions but they are shifted one diagonal backward as depicted in Fig. 1.10. The dotted line is the path they would have followed if only one signal would have been present.

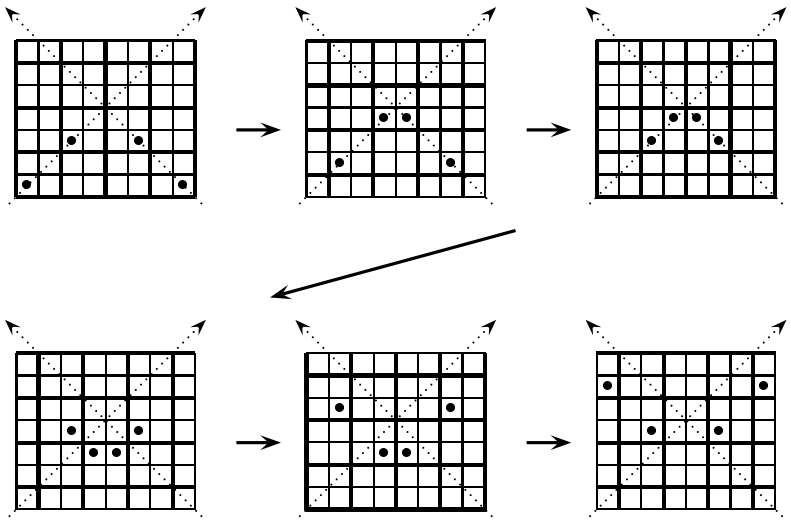


Fig. 1.10. Collision of two signals.

The computation is carried out in a binary form. Now we know how to make a signal. How can we encode different values in signal forms? This can be done following the encoding proposed by Margolus in [13]. Signal *1* is encoded as the signal one and *0* as no signal. To distinguish between *0* and no signal present, one has to know somehow whether a signal is present.

Signals *0* and *1* should interact in order to compute, so some logical gate has to be provided. With the collision presented in Fig. 1.10, without adding a ball, we can make the micro gate of Fig. 1.11. This is a two-signal gate with four possible outputs: two balls, one ball on the left, another on the right and no ball at all. If the two signals are *1*, then there are two balls and the two balls come out one row lower (the two $a \wedge b$ exits). If a is *1* (resp. *0*) and b is *0* (resp. *1*) then there is only one ball and thus no collision, the ball leaves on exit $a \wedge \overline{b}$ (resp. $\overline{a} \wedge b$). If no ball is present, then no ball leaves.

In any case, the output of balls (or lack of any ball) exactly corresponds to the encoding. This micro gate is devised by Margolus in [13] in order to build a more elaborated gate (i.e. Fredkin gate defined and used in the next subsections).

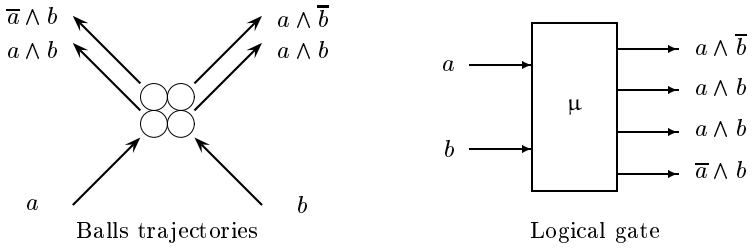


Fig. 1.11. Micro gate.

1.3.2 Conservative Logic

In the conservative logic, invented in [8], 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. To get a gate with a minimal computing ability, one has to consider a three-bit gate – the *Fredkin gate*. This gate works as follows. One bit goes through the gate unaffected and depending on value of this bit, two other bits just pass through or are permuted, as represented in Fig. 1.12.

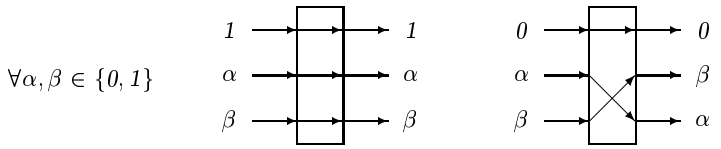


Fig. 1.12. Fredkin gate.

Fredkin gates can be built in the BBM for the basic encoding out of micro gates of Fig. 1.11 [28]. Also the BBM is simple, the construction of Fredkin gates is designed in two levels and uses a large amount of space and time. Morita [16] proved that it is possible to built any conservative gate out of Fredkin gates; the only other signals needed are zero signals which are regenerated at the end. Since zeroes are implemented by the lack of any signal we can say that

Lemma 2. *The BBM is able to simulate any conservative logical function with basic signals without feeding nor disposal of a signal.*

Any binary function f can be implemented with conservative logical functions. It is done using a larger conservative function φ. 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 guarantee that φ is conservative. There are drawbacks of the technique: constant bits have to be provided and unwanted bits are generated (and have to be disposed off in some way or another). This cannot be avoided with irreversible functions. Since the BBM does not allow the creation or removal of balls, one has to be very careful when implementing an irreversible gate.

1.3.3 Dual Encoding

The preceding construction is interesting as long as one uses Boolean circuits which work 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 cope with 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. 1.13. 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.

s^+	s^-	s
0	*0*	No signal
0	*1*	**0**
1	*0*	**1**
1	*1*	Error

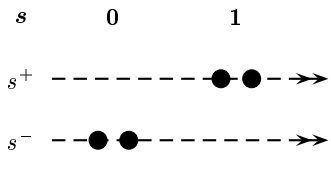


Fig. 1.13. From basic encoding to dual encoding.

It is possible to build a Fredkin gate with this encoding as depicted in Fig. 1.14. Some delay-elements may be required, but they are not indicated here for brevity.

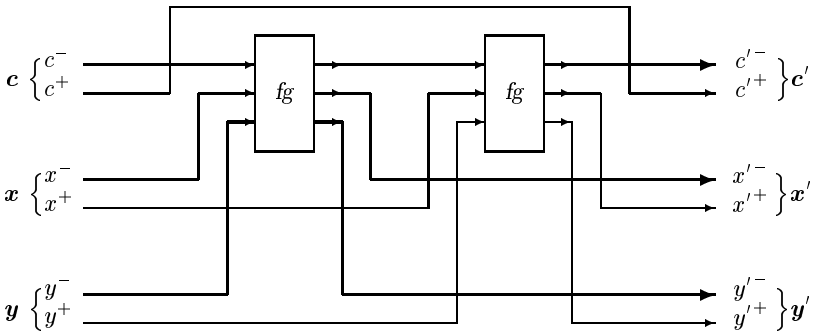


Fig. 1.14. Fredkin gate for dual signals.

It is still conceivable to compute any conservative function, but it is now also possible to make an autonomous `not` gate (Fig. 1.15). There is no risk of collision between signals because there is only one real signal for any dual signal.

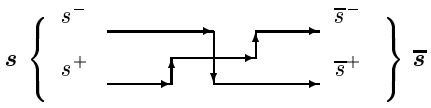


Fig. 1.15. A `not` gate with dual signals.

1.3.4 Reversible Logic

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

Lemma 3. *The BBM can simulate any reversible logical function with dual signals without any feeding nor disposal of the signals.*

1.4 Turing Universality of the BBM

A *two-counter automaton* is a finite automaton linked to two counters which can hold any non-negative integer value. The automaton can perform the following operations on the counters: add one, subtract one (zero minus one is zero) and test for nullity and branch.

Minsky [14] proved that there exist universal two-counters automata. To prove that the BBM is universal, it is enough to show that the BBM is able to simulate any two-counter automaton. The construction relies on the automaton on the one side, and on the counters on the other side.

1.4.1 Automaton

An 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 sends 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.

Since the transition function of the automaton can be irreversible, constant inputs have to be provided and garbage outputs are generated. The flow of constants is infinite since no one can presume the duration of the computation.

The automaton communicates with the counters by sending (dual) signals $\mathbf{o}$ to indicate modifications; receiving (dual) signal $\mathbf{e}$ indicates that an order was performed. It receives and sends the first bits of the counters in order to be able to test itself if a counter is zero. These signals are endlessly sent and received, most of the time, signals $\mathbf{o}$ and $\mathbf{e}$ are meaningless. The *order* $\mathbf{o}$ is encoded with two dual signals: $\mathbf{o} = (\mathbf{o}^p, \mathbf{o}^d)$. The signal $\mathbf{o}^p$ indicates whether or not the order is to be carried out ($\mathbf{o}^p = 0$ means no action). The signal $\mathbf{o}^d$ defines it: $\mathbf{o}^d = 0$ for subtraction and $\mathbf{o}^d = 1$ for addition. The *end of execution notification* signal $\mathbf{e}$ equals $\mathbf{1}$ to notify that an order was properly executed by the register, otherwise it is $\mathbf{0}$.

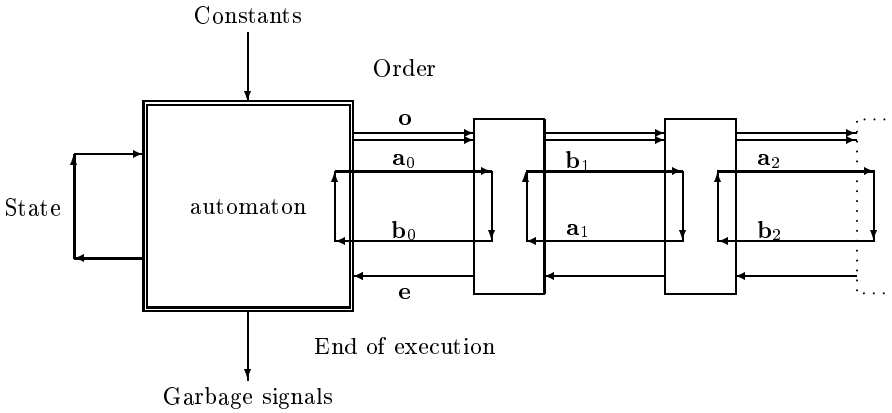


Fig. 1.16. The automaton and two register units.

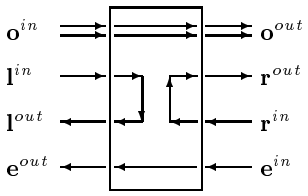
1.4.2 Counters

Counters are stored and handled inside a unique infinite line of logical *register units*. The value of counters are encoded in unary form with dual signals ($n \equiv \mathbf{1}^n\mathbf{0}^\omega$). The two counters are denoted as $\mathbf{A} = \mathbf{a}_0\mathbf{a}_1\ldots$ and $\mathbf{B} = \mathbf{b}_0\mathbf{b}_1\ldots$ The corresponding signals are locked between consecutive register units as depicted in figures 1.16 and 1.18. 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 nested not 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. 1.17. It should be noted that even if it is not conservative, it is reversible as it can be seen from the table, so that no extra signal is involved. The last two lines of the table look like they can be merged into a rule like "if $\mathbf{o}^p$ is $\mathbf{0}$ then nothing changes" but the function would not be "one to one" anymore. Each register unit implements a reversible function. Thanks to Lemma 3, register units can be totally autonomous. They do not bring any perturbation in the configuration.

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 ($\mathbf{o}^p = \mathbf{1}$), 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. 1.17). The modification is indicated by $\mathbf{o}^d$: $\mathbf{0}$ 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 $\mathbf{1}$.



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

$-$: any $\quad *$: unchanged

Fig. 1.17. Register unit and the corresponding logical function.

Signal $\mathbf{e}$ is $\mathbf{0}$ except when it brings the notification that the order was executed (and then it is $\mathbf{1}$). It is set to $\mathbf{1}$ by a register unit which carries out the 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. 1.18. Each unit receives only $\mathbf{a}$'s, then only $\mathbf{b}$'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 $\mathbf{a}$'s or only $\mathbf{b}$'s, but it meets all of them as it can be seen on Fig. 1.18.

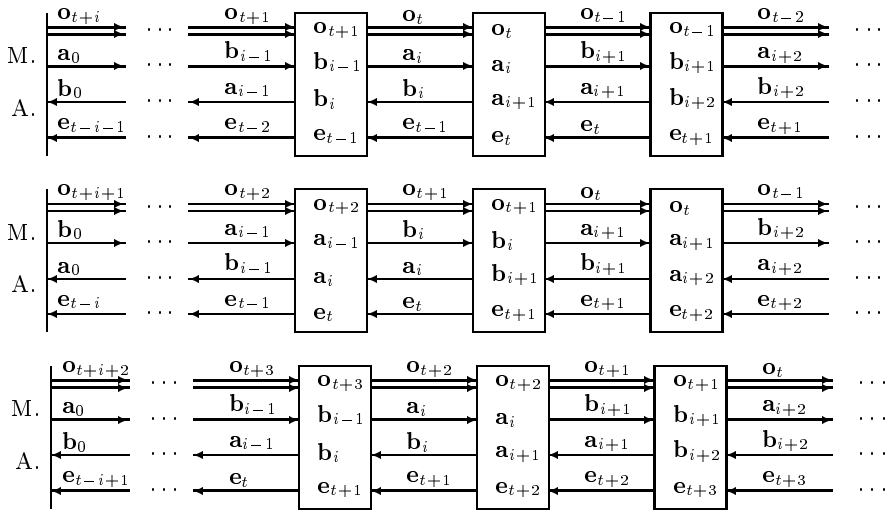


Fig. 1.18. Automaton, units and wiring for three successive iterations.

Let us decompose the execution of an order, first the test and branch and then changing the value of a counter.

If the counter $\mathbf{A}$ ($\mathbf{B}$) is null, the automaton knows it since $\mathbf{a}_0$ ($\mathbf{b}_0$) is in its inputs. It can test and branch directly.

If the automaton wants to subtract one from a null counter, the automaton just goes to the next instruction. If it wants to add one to a null counter, it sets $\mathbf{a}_0$ (or $\mathbf{b}_0$) to $\mathbf{1}$ and goes to the next instruction.

Let us consider that counter $\mathbf{A}$ is not null (it works exactly the same for counter $\mathbf{B}$). To make an operation $\mathbf{op}$ ($\mathbf{op}$ is $\mathbf{1}$ for addition, $\mathbf{0}$ for subtraction) over $\mathbf{A}$ the automaton sends a signal $\mathbf{o} = (\mathbf{1}, \mathbf{op})$ synchronized with $\mathbf{a}_0$. Then it waits till it receives a the signal $\mathbf{e}$ equal to $\mathbf{1}$ indicating that the operation was performed. After that the automaton goes on to the next operation.

The order $\mathbf{o}$ is received and treated by the register units as follows. Remember that counters are encoded in a unary form. The signal $\mathbf{o}$ travels and meets successively all the pairs $(\mathbf{a}_i, \mathbf{a}_{i+1})$, which are equal to $(\mathbf{1}, \mathbf{1})$, until it reaches the end of the meaning part of the counter ($(\mathbf{a}_i, \mathbf{a}_{i+1})$ equal $(\mathbf{1}, \mathbf{0})$). If $\mathbf{op}$ is $\mathbf{1}$ (addition) then the output value $(\mathbf{a}_i, \mathbf{a}_{i+1})$ is $(\mathbf{1}, \mathbf{1})$, otherwise (subtraction) it is $(\mathbf{0}, \mathbf{0})$. 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 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 the counter.

Going backward in time, the register unit, which performs the operation, is defined by the meeting of the $\mathbf{e}$ equal to $\mathbf{1}$ and the first $\mathbf{o}$ 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 has to enlarge the distance between register units and add new trapped signals.

Lemma 4. *The BBM is able to simulate any two-counters automaton.*

Since there exist universal two-counters automata we have the following:

Theorem 2. *The BBM is universal.*

The universality of the BBM was proved using reversible techniques. For the register units, reversibility was designed abstractly before it was implemented. This is a general strategy in a reversible context and can be translated to other reversible models.

1.5 Intrinsic Universality of the BBM

As we already know the BBM can compute any computable function. Can it do something more sophisticated? If yes, to what extent? The BBM can be programmed very easily on any computer. The real computer is less powerful than a Turing machine, because computer memory is finite. Does it contradict to the universality of the BBM?

No, it does not since the configuration of the BBM are infinite and only finite configurations can be simulated in a computer. The BBM handles infinite configurations and these configurations may well not be periodic or just

recursive. Turing universality deals with finite inputs and finite modification. The BBM, as well as CA and BCA, handles infinite data. Particularly, CA correspond exactly to all the functions over $S^{\mathbb{Z}^d}$ that are continuous for the product topology [9,22], and an automaton can change the state of every cell in a single transition step. There lies a computing ability which is beyond a Turing computability.

The question is whether the BBM can capture all the computing ability of CA over $\mathcal{C}$. Another way to understand this is to remember that a Turing machine is universal if it is able to simulate any Turing machine. We say that a CA (resp. R-CA) is *intrinsically universal* if it is able to simulate any CA (resp. R-CA) over any configuration (finite or not).

Definition 4. A machine is *intrinsically universal* inside a class of machine if it is able to simulate any machine of this class for any configuration. For a given machine, the way the simulation is carried out must be the same for every configuration.

In this section, we shall prove that the BBM is intrinsically universal for R-CA with our usual understanding of simulation. Then if simulation is taken in a wider way, namely *space-time simulation* [6], the BBM is intrinsically space-time universal for CA.

We introduce another sub-class of CA which is used throughout this section: partitioned CA (PCA). This model was proposed by Morita et al. [15]. The model, like BCA, was introduced to study and manage reversible "classical" CA.

1.5.1 Partitioned Cellular Automata

Partitioned cellular automata (PCA) imply mapping over the same set of configurations as BCA and CA. They also perform local, parallel and uniform updates of configurations, but the configuration updating is done differently.

Instead of partitioning the array, we partition the cells themselves. A cell state is partitioned into as many sub-states as there are neighbors. A transition is done by exchanging these parts with the neighbors and then updating the state. An intermediate state is constituted with the part which is left and the parts which are received. Updating is done by using a mapping ϕ from states to states, just as BCA use the mapping from blocks into blocks.

A PCA is totally defined by the tuple $(d, \mathcal{N}, (S_i)_{i\in\mathcal{N}}, \phi)$. The state is partitioned in such a way that there is a piece corresponding to every neighbor. The first part of the transition is to send all these pieces to the corresponding cells. Each cell receives one and only one sub-state of each part. Once a full state is reconstructed, the state is changed according to the local function $\phi : (S_i)_{i\in\mathcal{N}} \to (S_i)_{i\in\mathcal{N}}$.

This is illustrated for $\mathcal{N}_{vN} = \{(0,0), (0,1), (0,-1), (-1,0), (1,0)\}$, the von Neumann neighborhood, (the sub-states are denoted c, n, s, e and w)

on Fig. 1.19. On the right part, it is indicated which parts of the neighboring cells are considered for ϕ for updating.

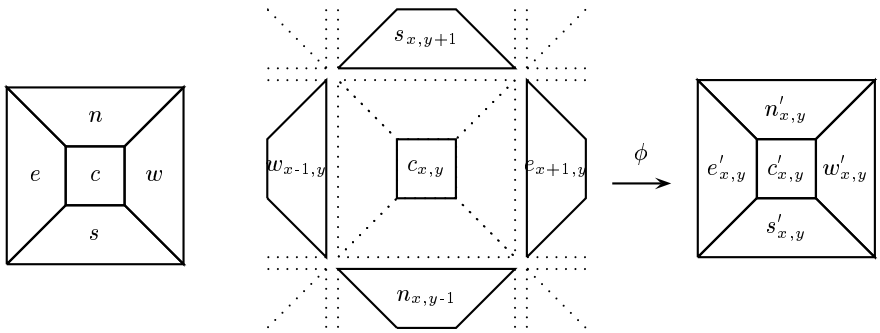


Fig. 1.19. Decomposition of a PCA cell and parts considered for ϕ.

Partitioned CA have one thing in common with the conservative logic, discussed in the Subsect. 1.3.1: there is no duplication – each sub-state is sent to one and only one cell and it is not kept anymore by the cell.

The following lemma is the PCA counterpart of Lemma 1.

Lemma 5. *A PCA P is reversible if and only if its transition function ϕ is a bijection.*

Proof. The exchange of sub-states is reversible. Each sub-state goes to one and only one cell, there is no duplication. Then, the global function of the PCA is reversible if and only if ϕ is one-to-one and onto.

Let us denote that the inverse of PCA is not exactly a PCA since the inverse of ϕ is applied before the sub-states are sent. Anyway, it is still a CA.

1.5.2 Intrinsic Universality of the BBM among R-CA

There exist R-CA and R-PCA that are able to simulate any R-CA (and R-PCA and R-BCA) over any configuration. We refer to the paper [5] for a full construction. The design, offered in [5], is based on simulation of R-CA by R-PCA and then the construction of an intrinsically universal R-PCA. This construction relies on binary encoding of states, encoding the transition function inside the configuration and using a reversible signal approach to test and update.

Theorem 3. *[5] Any R-CA can be simulated by a R-PCA and there exist intrinsically universal R-CA (and R-PCA).*

Moreover, the simulating R-CA use the von Neumann neighborhood. To prove that the BBM is intrinsically universal, we shall prove that it is able to simulate a R-PCA with this neighborhood. Two things must be taken care of. They are exchanging the sub-states and computing the transition function. Exchanging information is not a problem here because all the necessary wiring is present.

Since the PCA is reversible, the transition function is reversible (Lemma 5) as well. Thanks to Lemma 2, it is possible to implement a corresponding gate which does not need feeding nor disposal. Figure 1.20 presents the unit and shows how units are interconnected. This pattern is repeated over all the configurations so that infinite configurations and unbounded orbits are handled.

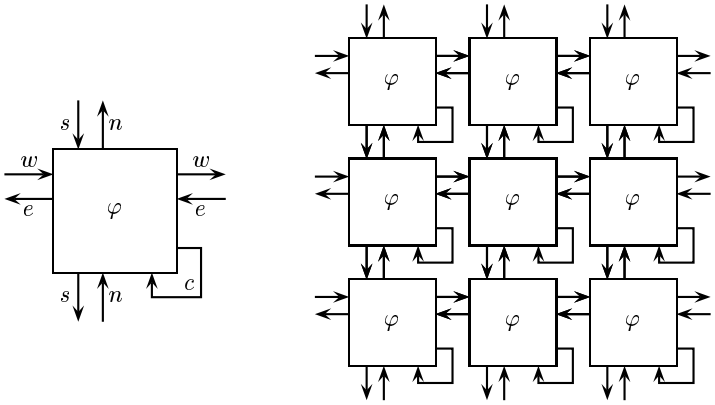


Fig. 1.20. Processing unit and interconnection for PCA.

Theorem 4. *The BBM is able to simulate any R-PCA.*

The following corollary follows from the Theorems 3 and 4 and the transitivity of simulation.

Corollary 1. *The BBM is intrinsically universal for reversible R-CA (R-BCA and R-PCA).*

This only works inside the subset of reversible cellular automata. It is known [1] that there exist CA that are intrinsically universal (for CA) but not reversible. This is an open problem to investigate whether any irreversible CA can be simulated by a reversible one (in terms of the Definition 2). The problem is tackled in the next section. There the simulation relation considers orbits and not just isolated configurations. In this particular context, we can give a positive answer.

1.5.3 Space-time Simulation

Following the Definition 2 of a simulation, a simulated configuration is totally encoded in a simulating configuration. This is a natural idea when one thinks about a dynamical system which goes from one configuration to the next one. Another point of view is to consider that single configurations are not important and that only the orbit, as the whole, is relevant. The next definition tries to capture this idea: the whole simulated space-time diagram is encoded in the simulating space-time diagram.

This definition is quite new. As far as we know, it was first introduced in [6].

Definition 5. Let $\mathcal{G}$ be the global transition function of some CA A and a be A's configuration. The *space-time diagram* $\mathbb{A} : \mathbb{Z}^d \times \mathbb{N} \to S$ associated to A and a is defined by $\mathbb{A}^t_x = \left(\mathcal{G}^t(a)\right)_x$. It is denoted by $(\mathcal{G}, a)$ or (A, a).

In other words, a space-time diagram is the sequence of the iterated images of configurations of a CA. A space-time diagram $\mathbb{B}$ is *embedded* into another space-time diagram $\mathbb{A}$ when it is possible to "reconstruct" $\mathbb{B}$ from $\mathbb{A}$ and the way that $\mathbb{B}$ is embedded into $\mathbb{A}$.

The recovering of an embedded B-configuration is done in the following way. An A-configuration is constructed by taking each cell at a given iteration. This A-configuration is decoded to get an iterated configuration for B. Hence we can use the following definition.

Definition 6. A space-time diagram $\mathbb{B} = (B, b)$ is *embedded* into another space-time diagram $\mathbb{A} = (A, a)$ when there exist three functions $\chi : \mathbb{Z}^d \times \mathbb{N} \to \mathbb{N}$, $\alpha : \mathcal{C}_B \to \mathcal{C}_A$ and $\beta : \mathcal{C}_A \to \mathcal{C}_B$ such that:
- $a = \alpha(b)$;
- $\forall (x, t) \in \mathbb{Z}^d \times \mathbb{N}$, let c^t be the configuration of A such that $c^t_x = \mathbb{A}^{\chi(x,t)}_x$;
- $\forall t \in \mathbb{N}$, $\mathcal{G}^t_B(b) = \beta(c^t)$.

The configuration b is encoded into a with respect to α. To recover an iterated image of b, the function χ indicates which iteration is to be considered for each cell and β decodes the generated configuration. The generation of c^t and then $\mathcal{G}^t_B(b)$ is illustrated in Fig. 1.21.

Definition 7. A CA A *space-time simulates* another CA B if and only if any space-time diagram of B can be embedded inside a space-time diagram of A.

The functions χ, α and β must be as minimal as possible because otherwise the functions could implement all the computation and A could be almost any CA.

The space-time simulation is still under investigation and more natural notion of embedding might appear.

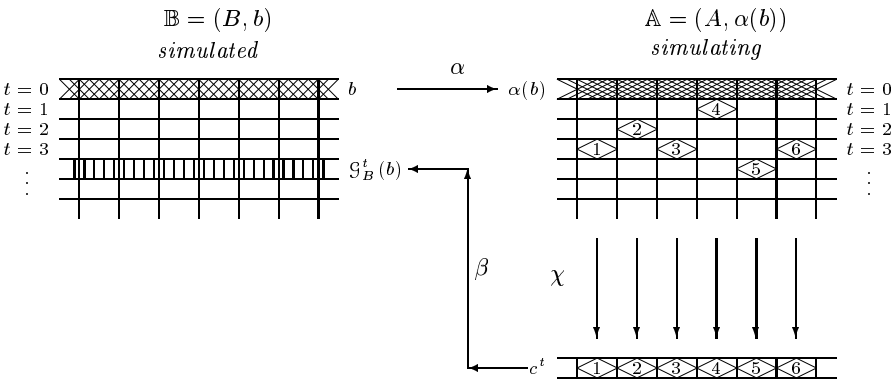


Fig. 1.21. Space-time diagram $\mathbb{B}$ is embedded into $\mathbb{A}$.

1.5.4 Intrinsic Space-Time Universality of the BBM among CA

We use the following result:

Theorem 5 ([6]). *There exist R-PCA which are space-time intrinsically universal for CA.*

In the construction of [6], the simulated configurations are "bent" in parabola inside the simulating space-time diagram. The space between parabola is used to store the information generated for implementation of the reversibility.

Among these R-PCA some have the von Neumann neighborhood. Again using Theorem 4 we draw the following corollary:

Corollary 2. *The BBM is space-time intrinsically universal for reversible CA (BCA and PCA) with the space-time simulation.*

Altogether, this shows that the BBM has a strong computing capability, it is *complete* for reversible CA and, considering the space-time simulation, complete for CA.

1.6 Uncomputable Properties

We already mentioned earlier that Turing machines, two-counters automata, cellular automata, and most computing languages capture the notion of computability. They compute all and only computable functions. What can be solved/computed in one architecture/language is exactly the same that can be solved in other.

All results of this section come directly from the classical *halting problem.*

Theorem 6 (Halting problem). *There is no Turing machine (two-counters automaton, recursive function, C++ program, etc. that, given the code of a Turing machine etc. as entry, stops and indicates whether or not the code would stop on an empty entry.*

For the following results, it is only sketched how to transform any Turing machine T (or two-counters automaton etc.) to a BBM configuration so that it has the desired property if and only if T stops when it is started with an empty tape.

For other results on complexity and calculability of decision problems in the evolution of CA see [27].

1.6.1 Reaching a Stable or Periodic Configuration

Since the BBM employs the "onto" mapping, any configuration has infinite chain of pre-images. It is possible to get any configuration after any number of iterations, provided the BBM is started in the proper predecessor-configuration. Moreover, since the inverse of the BBM is known and is very simple — BBM with the inverted order of the partitions — one can compute a configuration looking random but corresponding to the nth pre-image of a very regular image, e.g., a bottle or a face.

Since the BBM is reversible, it is not possible to reach a stable or periodic configuration from outside the corresponding orbit.

There are periodic orbits. Some can be built very easily as shown in Subsect. 1.3.1. Aperiodic configurations can be built also without problems: just let a single ball move on forever. If the underlying lattice is a finite (e.g. a torus), then all configurations belong to cycles for the same reason.

1.6.2 Reaching a (Sub-)Configuration

One can build a simulation of a two-counter automaton such that its stop signal corresponds to a dual signal. The value of the state is given by the passing of a signal to a given path between activation of the automaton.

Since it is not possible to predict (decide) if a two-counter automaton will stop, it is not possible to predict the passing of a signal in a given location.

Lemma 6. *It is undecidable if a ball will ever cross some given part of the configuration.*

More generally, we have the following:

Corollary 3. *It is undecidable if a given part of the configuration will ever be in a given sub-configuration.*

The whole configuration can be stretched (and padded with "void" state) so that parallel signals are always at least four diagonals away from each other and then make it so that when the simulated machine stops then two closed parallel signals are emitted. So it is decidable if there will ever be two close parallel basic signals. More general result looks as follows:

Theorem 7. *It is undecidable if a given sub-configuration will ever appear somewhere in the space diagram generated from a BBM configuration.*

References

1. Albert J. and Čulik II K. A simple universal cellular automaton and its one-way and totalistic version *Complex Systems* **1** (1987) 1–16.
2. Bennett C.H. Logical reversibility of computation *IBM Journal of Research and Development* **6** (1973) 525–532.
3. Burks A. *Essays on Cellular Automata.* (Univ. of Illinois Press, 1970).
4. Durand-Lose J. Reversible cellular automaton able to simulate any other reversible one using partitioning automata *Lecture Notes in Computer Sciences* **911** (1995) 230–244.
5. Durand-Lose J. Intrinsic universality of a 1-dimensional reversible cellular automaton *Lecture Notes in Computer Sciences* **1200** (1997) 439–450.
6. Durand-Lose J. Reversible space-time simulation of cellular automata *Theoretical Computer Science* **246** (2000) 117–129.
7. Durand-Lose J. Representing reversible cellular automata with reversible block cellular automata In: R. Cori, J. Mazoyer, M. Morvan, and R. Mosery (Editors) *Discrete models, combinatorics, computation and geometry (DM-CCG'01)*, volume AA. Discrete Mathematics and Theoretical Computer Science, 2001.
8. Fredkin E. and Toffoli T. Conservative logic *International Journal of Theoretical Physics* **21** (1982) 219–253.
9. Hedlund G.A. Endomorphism and automorphism of the shift dynamical system *Mathematical System Theory* **3** (1969) 320–375.
10. Kari J. Reversibility of 2D cellular automata is undecidable *Physica D* **45** (1990) 379–385.
11. Kari J. Representation of reversible cellular automata with block permutations *Mathematical System Theory* **29** (1996) 47–61.
12. Lecerf Y. Machines de turing réversibles. Récursive insolubilité en n 2 n de l'equation u = ‘ n où ‘ est un isomorphisme de codes *Comptes rendus de l'Académie francaise des sciences* **257** (1973) 2597–2600.
13. Margolus N. Physics-like models of computation *Physica D* **10** (1984) 81–95.
14. Minsky M. *Finite and Infinite Machines* (Prentice Hall, 1967).
15. Morita K. and Harao M. Computation universality of one-dimensional reversible (injective) cellular automata *Transactions of the IEICE* **E 72** (1989) 758–762.
16. Morita K. A simple construction method of a reversible finite automaton out of Fredkin gates, and its related problem *Transactions of the IEICE* **E 73** (1990) 978–984.
17. Morita K. Any irreversible cellular automaton can be simulated by a reversible one having the same dimension (on finite configurations) *Technical Report of the IEICE, Comp.* **92-45** (1992) 55–64.

18. Morita K. Computation-universality of one-dimensional one-way reversible cellular automata *Information Processing Letters* **42** (1992) 325–329.
19. Morita K. Reversible simulation of one-dimensional irreversible cellular automata *Theoretical Computer Science* **148** (1995) 157–163.
20. Morita K. and Ueno S. Computation-universal models of two-dimensional 16-state reversible automata *IEICE Transactions on Informations and Systems* **E75-D** (1992) 141–147.
21. Morita K. and Ueno S. Parallel generation and parsing of array languages using reversible cellular automata *Lecture Notes in Computer Science* **654** (1992) 213–230.
22. Richardson D. Tessellations with local transformations *Journal of Computer and System Sciences* **6** (1972) 373–388.
23. Rogozhin Yu. V. Seven universal Turing machines In *Systems and Theoretical Programming*, number 63, Matematicheskije Issledovanija (Academia Nauk Moldavskoi SSR) (1992) 76–90; in Russian.
24. Rogozhin Yu. V. Small universal Turing machines *Theoretical Computer Science* **168** (1996) 215–240.
25. Sipser M. *Introduction to the Theory of Computation* (PWS Publishing Co., Boston, Massachusetts, 1997).
26. Smith III A.R. Simple computation-universal cellular spaces *Journal of the Association for Computing Machinery* **18** (1971) 339–353.
27. Sutner K. On the complexity of finite cellular automata *Journal of Computer and System Sciences* **50** (1995) 87–97.
28. Toffoli T. and Margolus N. *Cellular Automata Machine — A New Environment for Modeling* (MIT Press, Cambridge, MA, 1987).
29. Toffoli T. and Margolus N. Invertible cellular automata: A review *Physica D* **45** (1990) 229–253.
30. Toffoli T. Computation and construction universality of reversible cellular automata *Journal of Computer and System Sciences* **15** (1977) 213–231.
31. Wolfram S. Universality and complexity in cellular automata *Physica D* **10** (1984) 1–35.