Conservative Computing in a One-dimensional Cellular Automaton with Memory

GENARO J. MARTÍNEZ1,2,∗ AND KENICHI MORITA3

1 Unconventional Computing Center, University of the West of England, Bristol, United Kingdom
2 Escuela Superior de Cómputo, Instituto Politécnico Nacional, México
3 Hiroshima University, Higashi Hiroshima, Japan
E-mail: km@hiroshima-u.ac.jp

Received: June 2, 2017. Accepted: June 27, 2017.

We propose a scheme to simulate Fredkin gates in a one-dimensional cellular automaton with memory by collision of particles, which is a moving pattern in this cellular space. Operations by collisions are confined in a black box with ballistic interaction, solitons and other collisions. We made a systematic analysis of binary collisions, i.e., collisions of two particles with different phases. They are used for handling these particles and obtaining the final outputs.

Keywords: Fredkin gates, elementary cellular automata, memory, particles, collisions, unconventional computing.

1 PRELIMINARIES

In the study of universal cellular automata (CA) — similar to the problem of small universal Turing machines — a number of parameters are reduced constantly: the number of states, the neighbourhood radius, the size of a configuration, and the dimension. The present elementary CA with memory (ECAM) gives a framework for implementing Fredkin gates based on a one-dimensional two-state CA having the neighborhood of radius 1. There, only one kind of glider is used to implement logical operations.

∗ Contact author: E-mail: genaro.martinez@uwe.ac.uk
On one-dimensional universal CA, there exist a number of relevant and interesting reductions [23].* Let \((s, r)\) denote the order of a CA, where \(s\) is the number of states, and \(r\) is the neighborhood radius. Smith III proved in 1971 [25] that a one-dimensional CA of order \((18, 1)\) is able to simulate a universal Turing machine. Albert and Culik II in 1987 [1] showed another universal CA of \((14, 1)\) with a totalistic local function. Later, Lindgren and Nordahl in 1990 [10] have reduced this order to \((7, 1)\). All of these CAs simulate a universal Turing machine, and their characteristic is that the spatial constructions were done by the interaction of signals not by the collisions of gliders. In 2004 Cook proved that ECA rule 110 (order \((2,1)\)) is universal, where collisions of hundred of gliders are used to control a cyclic tag system [6, 30]. As for reversible CAs, a one-dimensional universal reversible CA of order \((24, 1/2)\) controlled by signals was reported by Morita in 2011 [22].

In this paper, we research a complex ECAM rule \(\phi_{R22\text{maj:}4}\) (discovered in [11]). This automaton is able to simulate computable devices, such as: elastic collisions, solitons, and logic gates. Also, we present a proposal to simulate Fredkin gates in one dimension using this automaton. This ECAM is a derivative of the ECA rule 22 that is the projection of the Game of Life in one dimension [19]. The ECAM rule \(\phi_{R22\text{maj:}4}\) uses just one kind of particle (and its reflection) to develop computable devices.

This paper is organized as follows. Section two presents basic concepts. Section three offers a brief discussion of ECA rule 22. Section four discusses ECAM rule \(\phi_{R22\text{maj:}4}\) including its dynamics, basic gates, and the proposal of Fredkin gates in one dimension. Section five displays conclusions and future works.

2 ONE-DIMENSIONAL CELLULAR AUTOMATA

2.1 Basic notation

One-dimensional CA can be represented as a tuple as follows: \(< \Sigma, \varphi, \mu, c_0 >\). The system evolve on an array of cells \(x_i\), where \(i \in \mathbb{Z}\) (integer set) and each cell takes a value from the finite alphabet \(\Sigma\), thus \(x_i \in \Sigma\). A finite sequence of cells \(\{x_i\}\) represents a global configuration \(c\), such that \(c \in \Sigma^*\). This way, the set of finite configurations of length \(n\) is represented as \(\Sigma^n\).

Cell states in a configuration \(c(t)\) are updated at the next configuration \(c(t + 1)\) simultaneously by a local function as:

\[
\varphi(x_{i-r}, \ldots, x_i, \ldots, x_{i+r}) = x_{i+1}^{t+1}.
\]
Thus the local function determines the relation \( \varphi : \Sigma^\mu \to \Sigma \), where \( \mu \) represents the *neighbourhood* of \( \varphi \) with \( 2r + 1 \) neighbourhoods, and \( r \in \mathbb{Z}^+ \). So, the set of states of neighbour cells is \( \Sigma^\mu \), and the space of evolution rules is \( \Sigma^{2^\mu} \).

ECAs are defined for the parameters \(|\Sigma| = 2\), and \( r = 1 \) [29].

An *evolution* is represented by a sequence of finite configurations \( \{c^t\} \) given by the global mapping, \( \Phi : \Sigma^n \to \Sigma^n \). Thus the *global relation* is given as the function between configurations \( \Phi(c^t) = c^{t+1} \).

### 2.2 Cellular automata with memory

Conventional CA are ahistoric (memoryless): i.e., the new state of a cell depends on the neighbourhood state solely at the preceding time step of \( \varphi \). A CA with *memory* (CAM) can be considered as an extension of the standard framework of CA where every cell \( x_i \) is allowed to remember some period of its previous evolution. CAMs were introduced by Sanz and condensed basically in a book exploring an ample kind of memories on different kinds of CAs [24].

When using the memory we need to specify a kind of memory function \( \phi \) as follows:

\[
\phi(x_i^{t-\tau}, \ldots, x_i^{t-1}, x_i^t) = s_i. \tag{2}
\]

The parameter \( \tau < t \) specifies the degree of memory backwards, and the state \( s_i \in \Sigma \) is determined by the function \( \phi \) of the series of states \( x_i \) in the memory up to the time step \( t \). To make an evolution step we apply the original rule as:

\[
\varphi(s_i^{t-r}, \ldots, s_i^t, \ldots, s_i^{t+r}) = x_i^{t+1}. \tag{3}
\]

The main feature in CAM is that the mapping \( \varphi \) remains unaltered, while historic memory of all the past iterations is retained by featuring each cell as a summary of its past states by \( \phi \). This way, cells *canalise* memory to the map \( \varphi \).

As an example, we can consider memory function \( \phi \) as a *majority memory* \( \phi_{maj} \), where in case of a tie we will take the last value \( x_i \). In the case of three variables, the function \( \phi_{maj} \) represents the classic majority function [20] as follows:

\[
\phi_{maj}(a, b, c) = (a \land b) \lor (b \land c) \lor (c \land a). \tag{4}
\]

Note that memory is a simple function added to a CA. Particularly, an ample and systematic analysis on ECAM with majority, minority, and parity memories was done in [11].
3 ELEMENTARY CELLULAR AUTOMATON RULE 22

ECA rule 22 is a projection in one dimension of the famous two-dimensional CA Conway’s Game of Life (for details see [19]). Although its global behaviour does not display much information to deduce some complex dynamics, rule 22 is classified as a chaotic rule (class III) in the Wolfram’s classification [29,30].

Rule 22 can be stated as a relation similar to the Game of Life [4] given in the next conditions [19]:

I. **Born**: a dead cell $x_i$ at the time $t$ will be born in $t + 1$ if there is just one live neighbour.

II. **Survival**: an alive cell $x_i$ at the time $t$ will survive in $t + 1$ if there is not live neighbours.

III. **Dead by overcrowding**: an alive cell $x_i$ at the time $t$ will be dead in $t + 1$ if there is just two or one live neighbours.

Such a relation covers each condition of the Game of Life. Nevertheless, from a quick exploration in the one-dimensional dynamics, it does not exhibit complex behaviour. Figure 1 shows classic dynamics in ECA rule 22 beginning with a cell in state one and a random initial condition.

Rule 22 is an ECA evolving in one dimension of order $|\Sigma| = 2$ and neighbourhood radius $r = 1$. Thus the local rule $\varphi$ is defined as follows:

$$\varphi_{R22} = \begin{cases} 1 & \text{if } 100, 010, 001 \\ 0 & \text{if } 111, 110, 101, 011, 000 \end{cases}. \quad (5)$$

FIGURE 1
Classic dynamics in ECA rule 22 from (a) the configuration with only one cell in state 1, and (b) a random configuration with the ratio of 50% of state 1, on a ring of 323 cells to 300 generations.
The local function \( \varphi_{R22} \) has a probability of 37.5\% to get states 1 in the next generation and consequently a higher probability to get state 0 in the next generation. Of course, it is the same fixed point value for the Game of Life [18].

4 ELEMENTARY CELLULAR AUTOMATON WITH MEMORY
RULE \( \phi_{R22, maj:4} \)

In [11, 16] ECA rule 22 with memory is identified as a ‘strong’ class into ECAM classification. ECAM rule \( \phi_{R22, maj:4} \) is defined by the ECA rule 22 (R22), the majority memory (\( maj \)) and degree of memory backward \( \tau = 4 \). This way, to represent functions with memory and a ECA is as follows: \( \phi_{ECAm, \tau} \). Such that ECA represents the decimal notation of a specific rule, \( m \) is the kind of memory, and \( \tau \) the degree of memory backward.

The main characteristic is that rule \( \phi_{R22, maj:4} \) has only one kind of particle (and its reflection) traveling on its evolution space, as is illustrated in Figure 2. The set of particles \( G_{\phi_{R22, maj:4}} = \{g_L, g_R\} \) are defined in a perfect square volume of 11 × 11 cells, its properties are described in the Table 1.

Figure 3 displays a typical evolution from a random initial condition in ECAM \( \phi_{R22, maj:4} \). In this evolution, we can see how non-trivial patterns travel

<table>
<thead>
<tr>
<th>particle</th>
<th>period</th>
<th>shift</th>
<th>velocity</th>
<th>mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_R )</td>
<td>11</td>
<td>2</td>
<td>2/11</td>
<td>38</td>
</tr>
<tr>
<td>( g_L )</td>
<td>11</td>
<td>-2</td>
<td>-2/11</td>
<td>38</td>
</tr>
</tbody>
</table>

TABLE 1
Properties of particles of ECAM \( \phi_{R22, maj:4} \).
and interact by collisions. Also, collisions continue for a large number of generations. In this case the evolution lasts than 1000 steps, and their reactions do not become stationary.

We will start to design constructions based on collisions of particles. Initially, we have done a systematic analysis of binary collisions that is given
at the appendix. In the appendix we have 44 different binary reactions that offer an attractive number of possibilities for design devices, as computable functions, nano-devices, or collision theory in $\phi_{R22maj:4}$.

4.1 Ballistic collisions
Fredkin thought that two Boolean states were independently conserved by the logic gates with the intention that this could be represented as balls or atoms shall preserve their identity through of collisions. Thus Fredkin came up the Billiard-Ball Model of Computation consisting of elastic balls and flat mirrors [28]. Later Margolus developed a CA to implement the billiard-ball model, such CA is a block CA (BCA) which shows universal computing by collisions of soft spheres simulating a Fredkin gate in [13] and as crystalline computation in [14].

Ballistic collisions of particles permit to represent functions of two arguments (general signal-interaction scheme is conceptualized in Figure 4a), as follows:

1. $f(u, v)$ is a product of one collision (Figure 4b);
2. $f_i(u, v) \mapsto (u, v)$ identity (Figure 4c);
3. $f_r(u, v) \mapsto (v, u)$ reflection (Figure 4d);

where Figure 4a represents a result without preserving some identity of $u$ and $v$, while $f$ (in Figure 4b) displays the general scheme where two arguments are preserved.

ECAM $\phi_{R22maj:4}$ can reproduce ballistic collisions and preserve them given a good synchronisation. Figure 5 simulates identity collisions. All collisions preserve a reaction like soliton fixing distances between these particles or changing the number and position of them and they are preserved forever. Computing with solitons was explored amply by Steiglitz in [9, 27], a computation to implement a carry-ripple adder [26] can be developed in $\phi_{R22maj:4}$ by solitonic reactions.
Figure 6 simulates elastic collisions $f_i(u, v) \mapsto (v, u)$ with a pair of particles. This way, a pair of gliders reflect identically during all reactions. These elastic collisions are robust on any change of phase, position or number of them; 13 pair of particles in different positions preserve the elastic collisions forever.

Ballistic ball reactions are interaction gates. The interaction gate is a function with two inputs and four possible outputs (Figure 7a), this one has its inverse (Figure 7b) to return at the original inputs, it is reversible computation [8].

4.2 Fredkin gate
An schematic diagram and implementation to get a Fredkin gate in one dimension is proposed in this section.

Typically, two-dimensional CAs were studied to simulate Fredkin gates [8], rotatory elements [21], and reversible computing in general. In this case,
FIGURE 6
Ballistic collisions in ECAM $\phi_{R22maj4}$. 13 pairs of particles are synchronised to simulate the function reflection $f_r(u, v) \mapsto (v, u)$ from different intervals. Reflections are preserved forever.

FIGURE 7
With ballistic collisions we can handle an (a) interaction gate, (b) its inverse, and (c) its billiard ball model realization.
we use the ECAM $\phi_{R22maj,4}$ to simulate Fredkin gates in one dimension. For this implementation we need to adjust the timing with additional collisions of localizations that can be confined to a dark box. The physical computing device illustrated in Figure 8 (presented originally in [7]) represents a conservative computation with multiple inputs and the same number of outputs. So, precisely we will exploit this feature to adjust a scheme where reversible computing could be manipulated in an ECAM as an unconventional computing paradigm.

A conservative logic gate is a Boolean function that is invertible and preserves signals [8]. Fredkin gate is a classical conservative logic gate. The gate realises the transformation $(c, p, q) \mapsto (c, cp + \bar{c}q, cq + \bar{c}p)$, where $(c, p, q) \in \{0, 1\}^3$. Schematic functioning of Fredkin gate is shown in Figure 9 and the truth table is in Table 2.

Fredkin gate is universal because one can implement a functionally complete set of logical functions with this gate (Figure 10). Other gates implemented with Fredkin gate are shown below:

- $c = u, p = v, q = 1$ (or $c = u, p = 1, q = v$) yields the IMPLIES gate $y = u \rightarrow v$ ($z = u \rightarrow v$).
- $c = u, p = 0, q = v$ (or $c = u, p = v, q = 0$) yields the NOT IMPLIES gate $y = (v \rightarrow u)$ ($z = (v \rightarrow u)$) [8]
4.3 Basic collisions
To simulate a Fredkin gate in a non-invertible ECAM $\phi_{R22maj:4}$, we will take a set of basic collisions to preserve the reactions and the persistence of data, these basic collisions have a specific task and actions which are specified as follows.

- A *mirror* might reflect a mobile localisation, but the mirror should be deleted.
- A *doubler* might copy a value into two.
- A *soliton* crosses two mobile localisations preserving its identity with a small change of phase.
- A *splitter* separate two mobile localisations traveling together in opposite directions.
- A *flag* is a mobile localisation that will be activated from the initial condition depending on an input value given.
- A *displacer* might move forward a mobile localisation.

Fredkin gates implemented in non-invertible systems were proposed and simulated by Adamatzky in a non-linear medium as an oregonator model of Belousov-Zhabotinsky [3].

4.4 Fredkin gates in one dimension
Figure 11 displays the schematic diagram proposed to simulate Fredkin gates in ECAM rule $\phi_{R22maj:4}$. There are three inputs ($c, p, q$) and three outputs ($x, y, z$). During the computation auxiliary gliders are generated. They are deleted before reaching outputs. Gliders travel in two directions in a 1D chain of cells, they can be reused only when crossing periodic boundaries or via combined collisions. We use two gliders as flags, travelling from the left ‘$L_f$’

<table>
<thead>
<tr>
<th>$c$</th>
<th>$p$</th>
<th>$q$</th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>→</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**TABLE 2**
Truth table of a Fredkin gate.
and from the right ‘$R_f$’. Flags are activated depending on initial values for $c$ or $q$ as follows:

- If $c = 0$ then $R_f = 1$,
- If $q = 0$ then $L_f = 1$,
- In any other case $L$ and $R = 0$. 

FIGURE 10
Realisation of Boolean functions using Fredkin gate. (a) Fredkin gate, (b) NOT gate, (c) FANOUT gate, (d-e) OR gate, and (f-g) AND gate [8].
Mirrors $M$ are defined as follows:

- If $c = p = q = 1$ then $M = 2$,
- If $c = p = 1$ then $M = 2$,
- If $c = q = 1$ then $M = 1$,
- If $p = 1$ then $M = 1$,
- In any other case $M = 0$.

Distances between gliders are fixed as positive integers determined for a number of cells in the state ‘0’, as $\nu^0 \forall n \geq 0$. During the computation we split gliders when two gliders travel together, the split gliders travel in opposite directions. We use two gliders as mirrors to change the direction of
TABLE 3
Following the scheme in Figure 11 we specify a sequence of collisions that are controlled with flag gliders ($L_f$, $R_f$) and mirrors (M) glider.

<table>
<thead>
<tr>
<th>$cpq$</th>
<th>$xyz$</th>
<th>$L_f$</th>
<th>$R_f$</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>011</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>010</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

FIGURE 12
Fredkin gate in ECAM rule $\phi_{22maj;4}$. (a) INPUT $c = 1$, $p = 1$, $q = 1$, OUTPUT $x = 1$, $y = 1$, $z = 1$, (b) INPUT $c = 1$, $p = 0$, $q = 1$, OUTPUT $x = 1$, $y = 0$, $z = 1$. 
an argument movement. We use displacer to move an output glider and adjust its distance with respect to other.

Table 3 shows values of inputs, outputs, flags, and mirrors. First column represents INPUTS, the second column OUTPUTS, third column are values of a flag activated in the left side, fourth column are values of the flag activated in the right side, and the last column shows the number of necessary mirrors.

Space-time configurations of ECAM rule $\phi_{R^{22}maj:4}$ implementing Fredkin gate for all non-zero combinations of inputs are shown in Figures 12–15.

FIGURE 13
Fredkin gate in ECAM rule $\phi_{R^{22}maj:4}$. (a) INPUT $c = 1, p = 1, q = 0$, OUTPUT $x = 1, y = 1, z = 0$, (b) INPUT $c = 1, p = 0, q = 0$, OUTPUT $x = 1, y = 0, z = 0$. 
FIGURE 14
Fredkin gate in ECAM rule $\phi_{R22\text{maj}:4}$. (a) INPUT $c = 0$, $p = 1$, $q = 1$, OUTPUT $x = 0$, $y = 1$, $z = 1$, (b) INPUT $c = 0$, $p = 0$, $q = 1$, OUTPUT $x = 0$, $y = 1$, $z = 0$.

A way to connect more Fredkin gates in two dimensions to implement more complex computations is shown in Figure 16 that uses an array of these physical computing devices.

5 FINAL REMARKS

We have proposed a scheme to simulate Fredkin gates in one dimension in ECAM rule $\phi_{R22\text{maj}:4}$. But, although this can be confined in a physical
FIGURE 15
Fredkin gate in ECAM rule $\phi_{R22maj4}$. INPUT $c = 0$, $p = 1$, $q = 0$, OUTPUT $x = 0$, $y = 0$, $z = 1$.

FIGURE 16
Cascaded Fredkin gates (It is a modification of Fredkin array proposed in [15]).
computing device several limitations should be improved: avoid the use of mirrors and the use of flags. An option is that they could be manipulated into in a ring (or virtual CA collider [12, 17]).

REFERENCES


A APPENDIX – BINARY COLLISIONS IN $\phi_{R22 MAJ:4}$

FIGURE 17
Binary collisions in Rule 22 with memory.
FIGURE 18
Binary collisions in Rule 22 with memory.
FIGURE 19
Binary collisions in Rule 22 with memory.