Collision-based Computing

Andrew Adamatzky, Jérôme Durand-Lose

Collision Based Computing

Andrew Adamatzky and Jérôme Durand-Lose

Abstract Collision-based computing is an implementation of logical circuits, mathematical machines or other computing and information processing devices in homogeneous uniform unstructured media with traveling mobile localizations. A quanta of information is represented by a compact propagating pattern (glider in cellular automata, soliton in optical system, wave-fragment in excitable chemical system). Logical truth corresponds to presence of the localization, logical false to absence of the localization; logical values can be also represented by a particular state of the localization. When two more or more traveling localizations collide they change their velocity vectors and/or states. Post-collision trajectories and/or states of the localizations represent results of a logical operations implemented by the collision. One of the principle advantages of the a collision-based computing medium —hidden in 1D systems but obvious in 2D and 3D media— is that the medium is architecture-less: nothing is hardwired, there are no stationary wires or gates, a trajectory of a propagating information quanta can be see as a momentary wire. We introduce basics of collision-based computing, and overview the collision-based computing schemes in 1D and 2D cellular automata and continuous excitable media. Also we provide an overview of collision-based schemes where particles/collisions are dimensionless.

Key words: Belousov-Zhabotinsky, Cellular automata, Collision computing, Cycle tag systems, Geometrical computation, Gliders, Localizations, Turing machines

Andrew Adamatzky
Department of Computer Science
University of the West of England, Bristol, United Kingdom e-mail: andrew.adamatzky@uwe.ac.uk

Jérôme Durand-Lose
Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, B.P. 6759, F-45067 ORLÉANS Cedex 2, FRANCE. e-mail: Jerome.Durand-Lose@univ-orleans.fr

1 Introduction

Ideas of a collision-based computing[1] are based on studies dealing with collisions of signals, traveling along discrete chains, one-dimensional cellular automata. The interaction of signals traveling along one-dimensional conductors was amongst famous problems in physics, biology and physiology, for centuries, and the problem of interaction was interpreted in terms of finite state machines in 1960s. First ever computer science related results on signal interaction can be attributed to

- Atrubin [8], who designed fist ever multiplier based on a one-dimensional cellular automaton in 1965;
- Fisher [25], who developed cellular automaton generator of prime numbers in 1965; and
- Waksman [50], who initiated so much popular firing squad synchronization problem and provided an eight-state solution, in 1966.

In 1971 Banks [9] showed how to build wires and simple gates in configurations of a two-dimensional binary-state cellular automaton. This was not an architecture-free computing because a wire was represented by a particular stationary configuration of cell states (this was rather a simulation of a "conventional" electrical, or logical, circuit). However, the Banks's design made a huge influence on a theory of computing in cellular automata and beyond.

In 1982 Elwyn Berlekamp, John Conway and Richard Gay proved that Game of Life "can imitate computers" [11].

They mimicked electric wires by lines "along which gliders travel" and demonstrated how to do a logical gate by crashing gliders one to another. Chapter 25 of their "Winning Ways" [11] demonstrates computing designs that do not simply look fresh twenty years later but are still re-discovered again and again by Game of Life enthusiasts all over the Net.

Berlekamp, Conway and Gay employed a vanishing reaction of gliders — two crashing gliders annihilate themselves— to build a NOT gate. They adopted Gosper's eater to collect garbage and to destroy glider streams. They used combinations of glider guns and eaters to implement AND and OR gates, and the shifting of a stationary pattern, or block, by a mobile pattern, or glider, when designing auxiliary storage of information.

"There is even the possibility that space-time itself is granular, composed of discrete units, and that the universe, as Edward Fredkin of M.I.T. and others have suggested, is a cellular automaton run by an enormous computer. If so, what we call motion may be only simulated motion. A moving particle in the ultimate microlevel may be essentially the same as one of our gliders, appearing to move on the macro-level, whereas actually there is only an alteration of states of basic space-

[1] Edward Fredkin, Tommaso Toffoli, Norman Margolus and Elwyn Berlekamp and John Conway are fully recognized founders of the field of the collision-based computing. The term 'collision-based computing' was first used in [3], there are however several equivalent but less used now terms as 'signal computing', 'ballistic computing', 'free space computing', and 'billiard ball computing'.

time cells in obedience to transition rules that have yet to be discovered." —finishes Berlekamp-Conway-Gay's book [11].

Meantime, in 1978 Edward Fredkin and Tommaso Toffoli submitted a one-year project proposal to DARPA, which got funding, and thus started unfolding a chain of remarkable events.

Originally, Fredkin and Toffoli aimed to "drastically reduce the fraction of" energy "that is dissipated at each computing step" [26]. To design a non-dissipative computer they constructed a new type of a digital logic —conservative logic— that conserves both "the physical quantities in which the digital signals are encoded" and "the information present at any moment in a digital system" [26].

Fredkin and Toffoli further developed these ideas in the seminal paper "Conservative Logic", published in 1982 [27]. There a concept of ballistic computers emerged. The Fredkin-Toffoli model of conservative computation —the billiard ball model— explores "elastic collisions involving balls and fixed reflectors". Generally, they proved that "given a container with balls we can do any kind of computation".

The billiard ball model became a masterpiece of cellular automaton theory thanks to Norman Margolus who invented a cellular-automaton (block cellular automata or partitioned cellular automata) implementation of the model. Norman published this result in 1984 [36]. "Margolus neighbourhood" and "billiard ball model cellular automata" are exploited widely nowadays.

We do not provide a detailed account of collision-based computing. There are no excuses to avoid reading original sources [11, 27, 36]. A comprehensive, self-contained and addressed to a wide auditory report of modern state of collision-based computing is provided in the book [3]. In present chapter we rather discuss our personal experience on designing collision-based computing schemes in one- and two-dimensional cellular automata, and spatially extended non-linear media. We also provide few hands-on examples of recently discovered collision-based computing devices, we hope the examples will help readers to experiment with their own designs.

The chapter is structured as follows. Principles of collision-based computing are outlined in Sect. 2. Section 3 shows how basic logical gates can be implemented by colliding localizations in natural systems —simulated reaction-diffusion medium (Subsect. 3.1) and light-sensitive Belousov-Zhabotinsky medium (Subsect. 3.2). Theoretical foundations of computing with signals in 1D cellular automata are presented in Sect. 4, including collision-based implementation of 1D Turing machine (Subsect. 4.2.1) and cyclic tag systems (Subsect. 4.2.2). We complete our excursion to architectureless computing with abstract geometrical computation, Sect. 5, where time and space are continuous and particles/localizations are dimensionless.

2 Principles of collision based computing

Here we try to summarize all types of collision-based computers. A collision-based computer is an empty space populated with mobile and stationary localizations. The mobile localizations used for computing so far are

- billiard balls [27]
- gliders in cellular automata models [5, 11, 16, 42, 43],
- solitons in non-linear media [7, 30, 31, 40, 41, 47], and
- localized wave-fragments in excitable chemical media [2, 4].

Examples of stationary localizations are

- "still lives", blocks and eaters in Conway's Game of Life [11, 42],
- standing waves in automaton models of reaction-diffusion systems [5], and
- breathers in computing devices based on polymer chains and oscillons in vibrating granular materials [1].

Usually mobile localizations represent signals and stationary localizations are used to route the signals in the space; however, by allowing balls and mirrors to change their states, in addition to velocity vectors, we can in principle build multi-valued logic circuits.

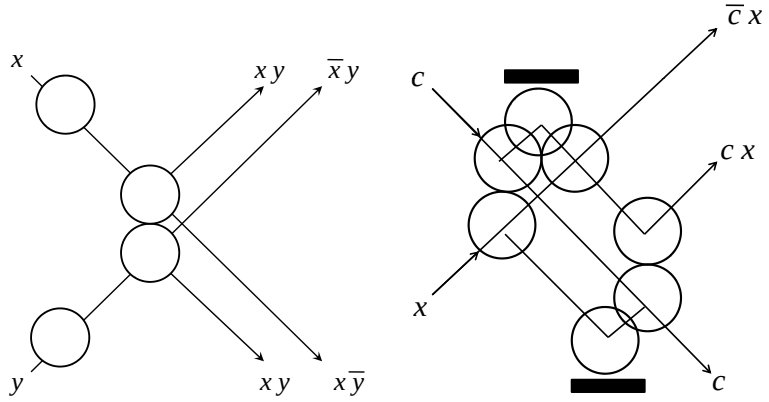


Fig. 1 Basics of billiard ball model: Fredkin-Toffoli interaction gate (a) and switch gate (b). From [27].

Classical examples of collision-based gates are shown in Fig. 1. The interaction gate involves two balls, to represent values of variables x and y (Fig. 1a). If two balls are present at the input trajectories, this corresponds to both variables having TRUTH values they collide and deflect in the result of collision. The deflected trajectories

of the balls represent conjunction of the input variables, xy. If only one ball, say the ball corresponding to the variable x, is present initially, then this ball travels along its original trajectories and does not change its velocity vector. The undisturbed trajectories of the balls represent logical functions $\bar{x}y$ and $x\bar{y}$ (Fig. 1a).

Switch gate (Fig. 1b) is another famous example, which also demonstrate a role of mirrors (stationary localizations). In the switch gate the signal x is conditionally routed by a control signal c. If signal is not present the ball x continues along its original trajectory traveling South-East. The ball x is delayed and its trajectory shifter eastward if the signal c is present in the system. After collision balls x and c are reflected but their further propagation is restricted by mirrors: the ball c collides with Northern mirror and the ball x with the Southern mirror (Fig. 1b).

3 Collision-based computing in natural systems

Those focused on implementation issues may wonder how a scheme of collision-based computing can be implemented in natural, chemical, physical or biological, materials. In this section we provide two examples of computing with localizations in spatially extended quasi-chemical (reaction-diffusion cellular automata) and excitable chemical (Belousov-Zhabotinsky reaction) systems.

3.1 Computing schemes in reaction-diffusion cellular automata: Spiral rule

The reaction-diffusion cellular automaton Spiral Rule [5] exhibits wide range of mobile and stationary localizations thus offering a unique opportunities to employ both travelling and still patterns in a computation process.

We designed automaton that emulates non-linearity of activator (A) and inhbitor (I) interaction for sub-threshold concentrations of activator. The following quasi-chemical reaction was used to construct cell-state transition rules [5]:

$$
\begin{array}{lll}
A + 6S \to A & A + I \to I & A + 3I \to I \\
 & A + 2I \to S & 2A \to I \\
 & 3A \to A & \beta A \to I \\
 & & I \to S.
\end{array}
$$

For sub-threshold concentration of the inhibitor and threshold concentrations of activator, the activator is suppressed by the inhibitor. For critical concentrations of the inhibitor both inhibitor and activator dissociate producing the substrate.

The quasi-chemical reactions are mapped to cellular automaton rules as follows. Take a totalistic hexagonal cellular automaton (CA), where a cell takes three states —substrate S, activator A and inhibitor I— and the cell updates its state depending

on just the numbers of different cell-states in its neighborhoods. The update rule can be written as follows:

$$x^{t+1} = f(\sigma_I(x)^t, \sigma_A(x)^t, \sigma_S(x)^t) \ ,$$

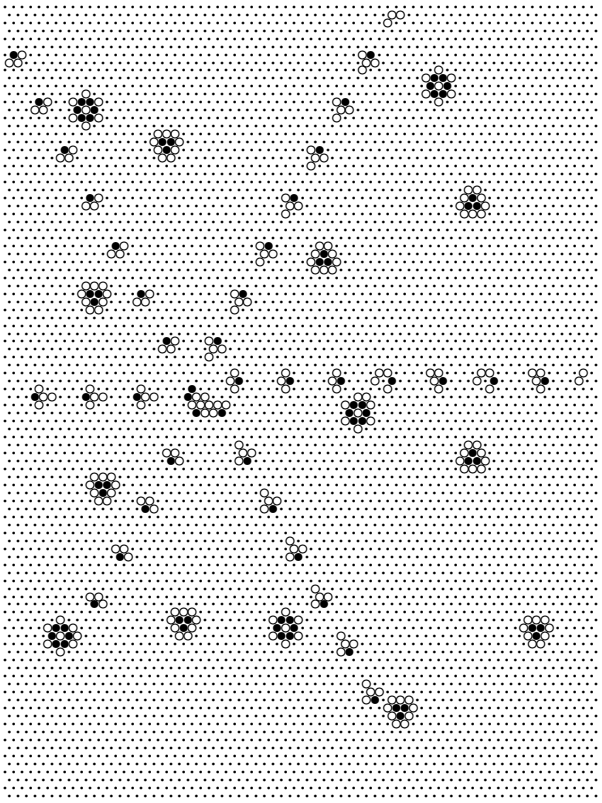
where $\sigma_p(x)^t$ is the number of cell x's neighbors (in seven cells neighborhood) with cell-state $p \in \{I, A, S\}$ at time step t. The rule is compactly represented a matrix $\mathbf{M} = (M_{ij})$, where $0 \leq i \leq j \leq 7$, $0 \leq i + j \leq 7$, and $M_{ij} \in \{I, A, S\}$ [6]. The output state of each neighborhood is given by the row-index i (the number of neighbors in cell-state I) and column-index j (the number of neighbors in cell-state A). We do not have to count the number of neighbors in cell-state S, because it is given by $7-(i+j)$. A cell with a neighborhood represented by indices i and j will update to cell-state M_{ij} which can be read off the matrix. In terms of the cell-state transition function this can be presented as follows: $x^{t+1} = M_{\sigma_2(x)^t \sigma_1(x)^t}$. The exact structure of the transition matrix is as follows [5]:

$$M = \begin{Bmatrix} S\ A\ I\ A\ I\ I\ I\ I \\ S\ I\ I\ A\ I\ I\ I \\ S\ S\ I\ A\ I\ I \\ S\ I\ I\ A\ I \\ S\ S\ I\ A \\ S\ S\ I \\ S\ S \\ S \end{Bmatrix}$$

The entry $M_{01} = A$ symbolizes the diffusion of activator A, $M_{11} = I$ represents the suppression of activator A by the inhibitor I, $M_{z2} = I$ ($z = 0, \cdots, 5$) is the self-inhibition of the activator in particular concentrations. $M_{z0} = S$ ($z = 1, \cdots, 7$) means that the inhibitor is dissociated in absence of the activator, and that the activator does not diffuse in sub-threshold concentrations; $M_{zp} = I$, $p \geq 4$ is an upper-threshold self-inhibition.

Starting in a random initial configuration the automaton will evolve towards a quasi-stationary configuration, with typically two types of stationary localizations, and a spiral generator of mobile localizations, or gliders (Fig. 2). The core of a glider-gun is a discrete analog of a 'classical' spiral wave. However, at some distance from the spiral wave tip the wave front becomes unstable and splits into localized wave-fragments. The wave-fragments continue traveling along their originally determined trajectories and keep their shape and velocity vector unchanged unless disturbed by other localizations. So, the wave-fragments behave as in sub-excitable Belousov-Zhabotinsky systems.

Basic gliders, those with one (activator) head, are found in five types (Fig. 3), which vary by the number of trailing inhibitors. Three types (G_{34}, G_{24}, G_{43}) alternate between two forms. Two types (G_4, G_5) have just one form. The spiral glider-gun in Figs. 2 emits G_{34} gliders. An alternative, low frequency, spiral glider-gun [53] (not shown) releases G_4 gliders. These basic gliders, and also a variety of more



(a)

Fig. 2 A typical quasi-stable configuration of the CA which started its development in a random initial configuration (with $1/3$ probability of each cell-state). Cell-state I (inhibitor) is shown by a black disk, cell-state A (activator) by a circle, and cell-state S (substrate) by a dot. We can see there are two types of stationary localizations (glider eaters) and a spiral glider-gun, which emits six streams of gliders, with frequency a glider per six time steps in each glider stream. From [5].

complicated gliders including mobile glider-guns, are also generated by many other interactions. Stationary localizations, or eaters, (Fig. 3ij) is yet one more important feature of the CA.

Principle components of any computing device are input interface, memory, routers and logical gates. We refer readers to the paper [5] to study possible input functions.

How to implement a memory in the Spiral Rule CA? The eater E_6 can play the role of a 6-bit flip-flop memory device. The substrate-sites (bit-down) between inhibitor-sites (Fig. 3ij) can be switched to an inhibitor-state (bit-up) by a colliding glider.

An example of writing one bit of information in E_6 is shown in Fig. 4. Initially E_6 stores no information. We aim to write one bit in the substrate-site between

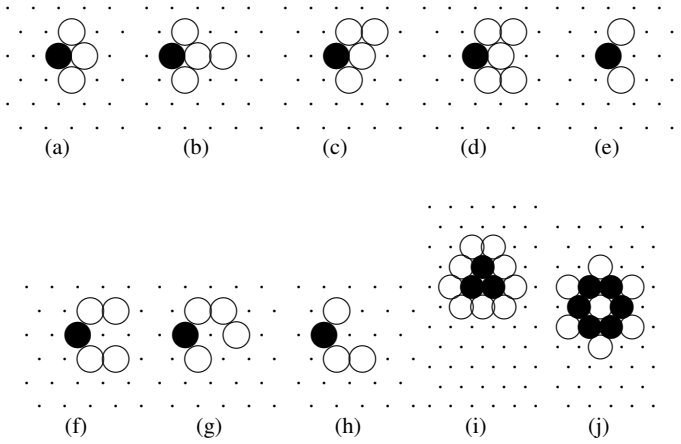


Fig. 3 The basic localizations in Spiral rule automaton: gliders (a)–(g) and eaters (h)–(i). (a)–(g) Five types of gliders, shown here traveling West, in the direction of their activator head (cell-state *A*), with a tail of trailing inhibitors made up of several cell-states *I*. The glider designator G_{ab} refers to the numbers of trailing inhibitors. (a) and (b) two forms of glider G_{34}. (c) glider G_4. (d) glider G_5. (e) and (f) two forms of glider G_{24}, (g) and (h) two forms of glider G_{43}. (i) eater E_3, (j) eater E_6. From [5].

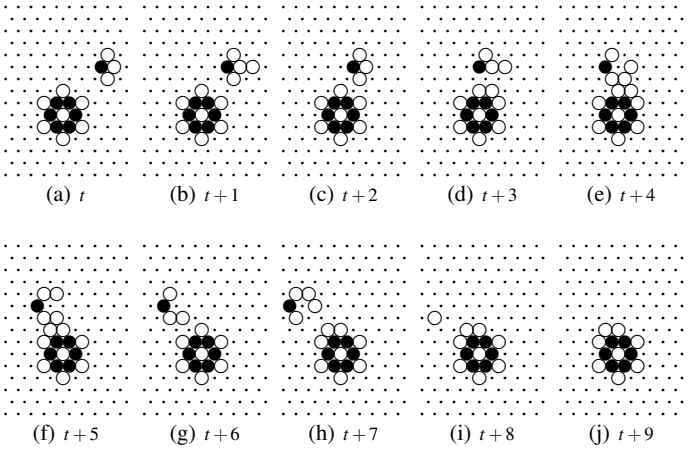


Fig. 4 Write bit. From [5].

the northern and north-western inhibitor-sites (Fig. 4a). We generate a glider G_{34} (Fig. 4bc) traveling West. G_{34} collides with (or brushes past) the North edge of E_6 resulting in G_{34} being transformed to a different type of glider, G_4 (Fig. 4gh). There is now a record of the collision —evidence that writing was successful. The

structure of E_6 now has one site (between the northern and north-western inhibitor-sites) changed to an inhibitor-state (Fig. 4j) —a bit was saved.

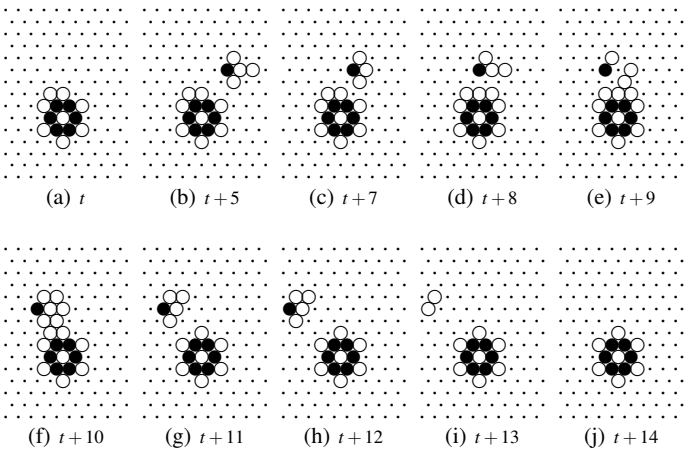


Fig. 5 Read and erase bit. From [5].

To read a bit from the E_6 memory device with one bit-up (Fig. 5a), we collide (or brush past) with glider G_{34} (Fig. 5b). Following the collision, the glider G_{34} is transformed into a different type of basic glider, G_34 (Fig. 5g), and the bit is erased (Fig. 5j).

To route signals we can potentially employ other gliders to act as mobile reflectors. Figure 6 shows how a glider traveling North-West collides with a glider traveling West, and is reflected South-West as a result of the collision. However both gliders are transformed to different types of gliders. This is acceptable on condition that both types of glider represent the same signal, or signal modality.

There are two more gates which are useful in designing practical collision-based computational schemes. They are the FANOUT gate and the ERASE gate. The FANOUT gate is based on glider multiplication. There are a few scenarios where one glider can be multiplied by another glider (for details see the original beehive rule [52]), e.g. one can make a FANOUT gate by colliding glider G_{34} with glider G_{24}. The gliders almost annihilate as a result of the collision, however recover into a complicated, which splits into three G_5 gliders. To annihilate a glider we can collide it with the central body of an eater, or with another glide, e.g. head-on collisions usually lead to annihilation.

The *asynchronous* XOR gate can be constructed from the memory device in Figs. 4 and 5, employing the eater E_6 and the glider G_{34}. The incoming trajectory of gliders is an input $x = \langle x, y \rangle$ of the gate, and the state of the cell which is switched to the inhibitor state by gliders, is an output z of the gate (this cell is shown by $\otimes$ in Fig. 7a). As seen in Fig. 4, when glider G_{34} brushes by the eater E_6 it 'adds' one inhibitor state to the eater configuration (Fig. 4, $t+7$), and transforms itself in

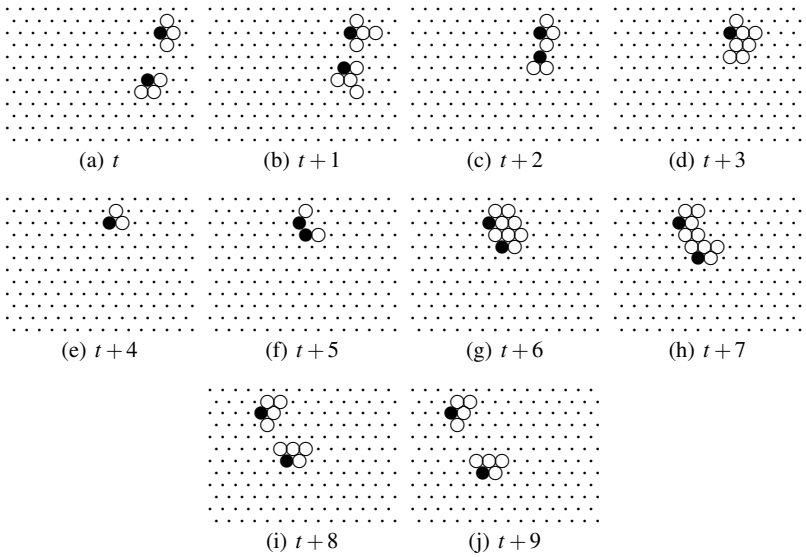
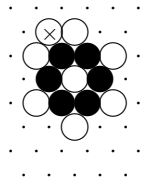


Fig. 6 Glider reflection. From [5].



(a)

x_1	x_2	y
0	0	S
G_{34}	0	I
0	G_{34}	I
G_{34}	G_{34}	S

(b)

Fig. 7 Asynchronous XOR gate. (a) position of output cell is shown by $\otimes$. (b) operation implemented by the gate, input state G_{34} is logical TRUE, output state S is FALSE, output state I is TRUE. From [5].

glider G_{43}. If glider G_{34} brushes by E_6 with an additional inhibitor state (Fig. 5, t) it 'removes' this additional state and transforms itself into glider G_4 (Fig. 5, $t+11$).

Assume that the presence of glider G_{34} symbolizes input logical TRUE and its absence —input FALSE, inhibitor state I in cell $\otimes$— output TRUE and substrate state S —output FALSE. The result of this logical operation can be read directly from the configuration of E_6 or by sending a control glider to brush by E_6 to detect how the glider is transformed. Then the structure implements exclusive disjunction (Fig. 7b). The gate constructed is asynchronous because the output of the operation does not depend on the time interval between the signals but only on the value of signals: when inhibitor state is added or removed from E_6 the configuration of E_6 remains stable and does not change till another glider collides into it.

The eater E_6 can take four different configurations resulting from the interactions of gliders brushing past, and there are seven types of glider produced in collisions with the eater (including some basic types flipped). We therefore envisaged [5] that a finite state machine can be implemented in the eater-glider system. The internal state of such a machine is represented by the configuration of the eater, the type of incoming glider symbolizes the input symbol of the machine, and the type of outgoing glider represents the output state of the machine.

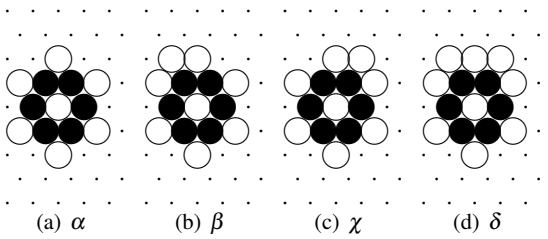


Fig. 8 Encoding the internal states of the glider-eater machine in the configuration of eater E_6.

To construct the full state transition table of the eater-glider machine we collided seven types of glider into four configurations of the eater and recorded the results of the collisions. For the sake of compact representation we encoded the configurations of the eater as shown in Fig. 8. We denote the gliders as follows: G_{34} as a, G_{43} as b, G_5 as c, G_4 as d, G_{24} as e, G^4 (glider G_4 flipped horizontally) is f, and G^{43} (glider G_{43} flipped horizontally) is g. The state transition table is shown in Fig. 9.

	a	b	c	d	e	f	g
α	βb	δc	αb	αe	δd	αe	δc
β	αd	δe	βc	βc	χg	αa	χe
χ	χd	βe	δf	χa	βb	χa	βe
δ	δb	βc	χg	χe	αf	δe	αa

Fig. 9 The state transition table of the eater-glider machine. Tuple xy, a pair made up of an eater state x and glider state y, at the intersection of the ith row and jth column, signifies that being in state i while receiving input j the machine takes state x and generates output y.

Consider the internal states of the eater-glider machine as unary operators on the set $\{a,b,c,d,e,f,g\}$, i.e. the machine's state is reset to its initial state after the collision with the glider. For example, the unary operator α implements the following transformation: $a \to b$, $b \to c$, $c \to a$, $d \to a$, $e \to d$, $f \to e$, $g \to e$. The operators have the following limit sets: operator α has the limit set $\{a,b,c\}$, β —set $\{c\}$, χ has two limit sets $\{a,d\}$ and $\{b,c\}$, and operator δ— two limit sets $\{a,b,c,d\}$ and $\{e,f\}$. Considering unary operators $a,\cdots,g$ operating on set $\{\alpha,\beta,\chi,\delta\}$ we obtain the limit sets shown in Fig. 10. Many of the operators have more then two limits sets, which may indicate significant computational potential of the eater-glider machine.

operator	limit set
a	$\{\alpha,\beta\},\{\delta\}$
b	$\{\beta,\delta\}$
c	$\{\alpha\},\{\beta\},\{\delta,\chi\}$
d	$\{\alpha\},\{\beta\},\{\chi\}$
e	$\{\alpha,\delta\},\{\beta,\chi\}$
f	$\{\alpha\},\{\chi\},\{\delta\}$
g	$\{\alpha,\delta\},\{\beta,\chi\}$

Fig. 10 Limit sets of unary operators $a, \cdots, g$.

To characterize the eater-glider machine in more detail we studied what output strings are generated by the machine when the machine receives the uniform infinite string s^*, $s \in \{a, \cdots g\}$ on its input. These input string to output string transformations are shown in Fig. 11.

	a^*	b^*	c^*	d^*	e^*	f^*	g^*
α	$(bd)^*$	$c(ce)^*$	b^*	e^*	$(de)^*$	e^*	$(ca)^*$
β	$(db)^*$	$(ec)^*$	c^*	c^*	$(gb)^*$	ae^*	e^*
χ	d^*	$e(ec)^*$	$(fg)^*$	a^*	$b(gb)^*$	a^*	e^*
δ	b^*	$(ce)^*$	$(gf)^*$	ea^*	$(ed)^*$	e^*	$(ac)^*$

Fig. 11 Input string to output string transformations implemented by the eater-glider machine. String s, at the intersection of the ith row and jth column, tells us that being initially in state i and receiving a uniform string j, the machine generates string s.

Input string $abcdefg$ evokes the following output strings when fed into the machine. The machine starting in state α generates string $begabac$, in state β —string $dcgabac$, in state χ— string $deccgae$, and in state δ —string $bcccgae$.

3.2 Collision-based computing in excitable chemical media

In this section we give a brief introduction to implementation of collision-based circuits in excitable chemical media, Belousov-Zhabotinsky (BZ) system. Examples discussed here are based on numerical experiments, see Chapter "Reaction Diffusion Computers" of this book on implementation of collision-based circuits in BZ medium in chemical laboratory conditions. Now we discuss computing with localized wave-fragments in the Oregonator [24, 49] model adapted to a light-sensitive BZ reaction with applied illumination [10, 33]:

$$\frac{\partial u}{\partial t} = \frac{1}{\varepsilon}(u - u^2 - (fv + \phi)\frac{u-q}{u+q}) + D_u\nabla^2 u \ , \text{ and}$$
$$\frac{\partial v}{\partial t} = u - v \ .$$

where variables u and v represent local concentrations of bromous acid $HBrO_2$ and the oxidized form of the catalyst ruthenium Ru(III), ε sets up a ratio of time scale of variables u and v, q is a scaling parameter depending on reaction rates, f is a stoichiometric coefficient, ϕ is a light-induced bromide production rate proportional to intensity of illumination (an excitability parameter —moderate intensity of light will facilitate excitation process, higher intensity will produce excessive quantities of bromide which suppresses the reaction). We assumed that the catalyst is immobilized in a thin-layer of gel, therefore there is no diffusion term for v. To integrate the system we use Euler method with five-node Laplasian operator, time step $\triangle t = 10^{-3}$ and grid point spacing $\triangle x = 0.15$, with the following parameters: $\phi = \phi_0 + A/2$, $A = 0.0011109$, $\phi_0 = 0.0766$, $\varepsilon = 0.03$, $f = 1.4$, $q = 0.002$.

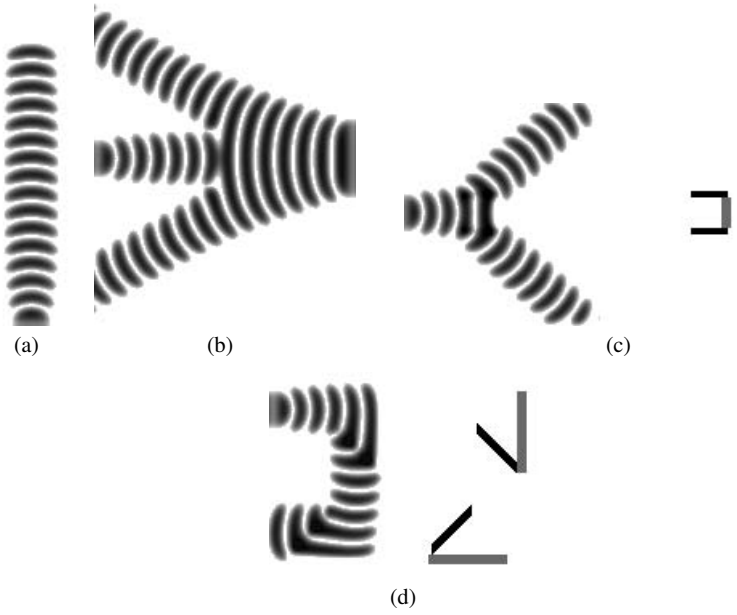


Fig. 12 Basic operations with signals. Overlay of images taken every 0.5 time units. Exciting domains of impurities are shown in black, inhibiting domains of impurities are shown in gray. (a) Wave fragment traveling north. (b) Signal branching without impurities: a wave fragment traveling east splits into two wave fragments (traveling south-east and north-east) when collides to a smaller wave fragment traveling west. (c) Signal branching with impurity: wave fragment traveling west is split by impurity (shown on the right) into two waves traveling north-west and south-west. (d) Signal routing (U-turn) with impurities: wave fragment traveling east is routed north and then west by two impurities (shown on the right). An impurity-reflector consists of inhibitory (gray) and excitatory (black) chains of grid sites. From [2].

Chosen parameters correspond to a region of "higher excitability of the sub-excitability regime" outlined in [46] that supports propagation of sustained wave fragments (Fig. 12a). These wave fragments are used as quanta of information in

our design of CB logical circuits. The waves were initiated by locally disturbing initial concentrations of species, e.g. ten grid sites in a chain are given value $u = 1.0$ each, this generated two or more localized wave fragments, similarly to counter-propagating waves induced by temporary illumination in experiments. The traveling wave fragments keep their shape for around $4 \cdot 10^3$–10^4 steps of simulation (4-10 time units), then decrease in size and vanish. The wave's life-time is sufficient however to implement logical gates; this also allows us not to worry about 'garbage collection' in the computational medium.

We model signals by traveling wave fragments [10, 46]: a sustainly propagating wave fragment (Fig. 12a) represents TRUE value of a logical variable corresponding to the wave's trajectory (momentarily wire). To demonstrate that a physical systems is logically universal it is enough to implement negation and conjunction or disjunction in spatio-temporal dynamics of the system. To realize a fully functional logical circuit we must also know how to operate input and output signals in the system's dynamics, namely to implement signal branching and routing; delay can be realised via appropriate routing.

We can branch a signal using two techniques. Firstly, we can collide a smaller auxiliary wave to a wave fragment representing the signal, the signal-wave will split then into two signals (these daughter waves shrink slightly down to stable size and then travel with constant shape further $4 \cdot 10^3$ time steps of the simulation) and the auxiliary wave will annihilate (Fig. 12b). Secondly, we can temporarily and locally apply illumination impurities on a signal's way to change properties of the medium and thus cause the signal to split (Fig. 12c).

A control impurity, or reflector, consists of a few segments of sites which illumination level is slightly above or below overall illumination level of the medium. Combining excitatory and inhibitory segments we can precisely control wave's trajectory, e.g. realize U-turn of a signal (Fig. 12d).

A typical billiard-ball model interaction gate [27, 36] has two inputs —x and y, and four outputs —$x\bar{y}$ (ball x moves undisturbed in absence of ball y), $\bar{x}y$ (ball y moves undisturbed in absence of ball x), and twice xy (balls x and y change their trajectories when collide to each other). We were unable to make wave fragments implement exact billiard-ball gate, because the interacting waves either fuse or one of the waves annihilates in result of the collision to another wave.

However, we have implemented a BZ (non-conservative) version of billiard-ball gate with two inputs and three outputs, i.e. just one xy output instead of two. This BZ collision gate is shown in Fig. 13.

Rich dynamic of BZ medium allows us also to implement complicated logical operations just in a single interaction event. An example of a composite gate with three inputs and six outputs is shown in Fig. 14. As we see in Fig. 14, some outputs, e.g. $\bar{x}yz$, are represented by gradually vanishing wave fragments. The situation can be dealt with by either using very compact architecture of the logical gates or by installing temporary amplifiers made from excitatory fragments of illumination impurities.

As know from results of computer simulations and experimental studies, classical excitation waves merge or annihilate when collide one with another. This may

(a) (b)

Fig. 13 Two wave fragments undergo angle collision and implement interaction gate $\langle x,y\rangle \to \langle x\bar{y},xy,\bar{x}y\rangle$. (a) In this example $x = 1$ and $y = 1$, both wave fragments are present initially. Overlay of images taken every 0.5 time units. (b) Scheme of the gate. In upper left and bottom left corners of (a) we see domains of wave generation, two echo wave fragments are also generated, they travel outwards gate area and thus do not interfere with computation. From [2].

complicated implementation of non-trivial logical circuits in classical excitable media. Wave-fragments however behave a bit differently, more like quasi-particles.

In computational experiments with exhaustive analysis of all possible collisions between localized wave-fragments [4] we classified all the collisions as (Fig. 15): quasi-elastic reflection of wave-fragments (Fig. 15a), pulling and pushing of a wave-fragment by another wave-fragment (Fig. 15bc), sliding of one wave-fragment along refractory trail of another fragment (Fig. 15d), and translation of wave-fragment along one axis by another wave-fragment (Fig. 15e). Examples of two types of collision are shown in Fig. 16.

4 One-Dimensional Cellular Automata

This section deals with cellular automata in general and discrete signals in particular. It is shown both how they compute in the classical understanding and how they can be used to implement some phenomena relevant to massively distributed computing but without any sequential counterpart.

Cellular automata (CA) were introduced as a discrete model for parallel, local and uniform phenomena, whether of engineering, physical or biological nature. They often provide a medium where signals naturally appear and are thoroughly used to understand the global dynamics. The other way round, a correct handling of such signals is the key to compute and to design special purpose CA.

A cellular automaton works in the following way. The space is regularly partitioned in cells. All the cells are identical and are regularly displayed as an infinite array (having as many dimensions as the modeled space). Each cell can be in finitely many states and evolves according to its state and the states of the surrounding cells —*locality*. All the cells are identical and behave similarly —*uniformity*. The cells are all updated *synchronously*, like a *massive parallel* process sharing a single clock. By essence, CA form a discrete model: discrete time, discrete space and discrete values.

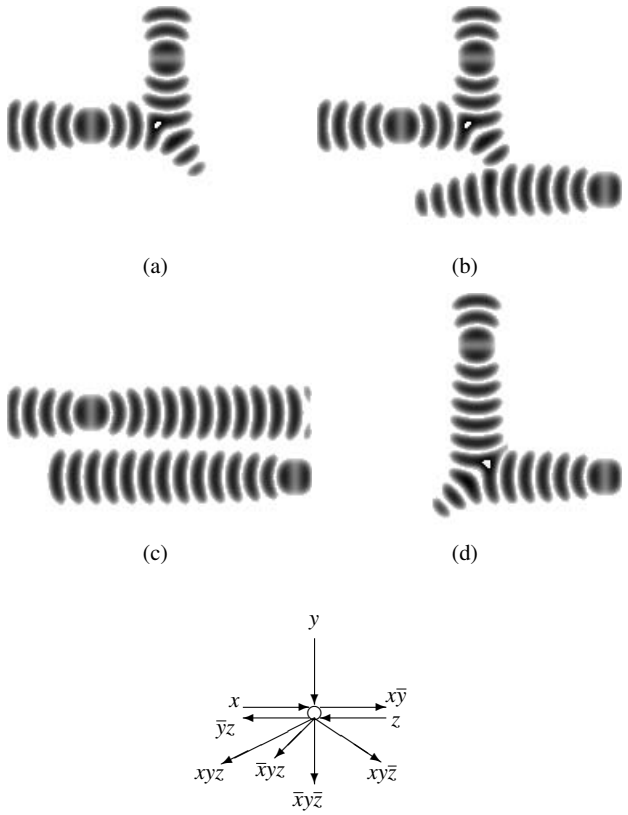


Fig. 14 Implementation of $\langle x,y,z\rangle \to \langle x\bar{y},\bar{y}z,xyz,\bar{x}yz,\bar{x}y\bar{z},xy\bar{z}\rangle$ interaction gate. Overlay of images of wave fragments taken every 0.5 time units. The following combinations of input configuration are shown: (a) $x = 1$, $y = 1$ $z = 0$, north-south wave collides to east-west wave. (b) $x = 1$, $y = 1$, $z = 1$, north-south wave collides to east-west wave, and to west-east wave. (c) $x = 1$, $y = 0$, $z = 1$, west-east and east-west wave fragments pass near each other without interaction. (d) $x = 0$, $y = 1$, $z = 1$, north-south wave collides to east-west wave. (e) Scheme of the gate. From [2].

Generally, the array is considered to extend infinitely in all directions so that there is no boundaries to deal with. However, borders can be encoded by special states which are never changed and make the two sides independent. Another way to simulate a CA on a computer is to consider that outside of a finite range, all cells are in the same stable state (called a *quiescent state*). Finite/periodic configuration can also be generated by displaying the cells to be on a ring (or torus): the last and first are then neighbors.

Previous collision based systems are CA, or better say are simulated by CA, some of them working on hexagonal lattices. Computation in dimension two and above

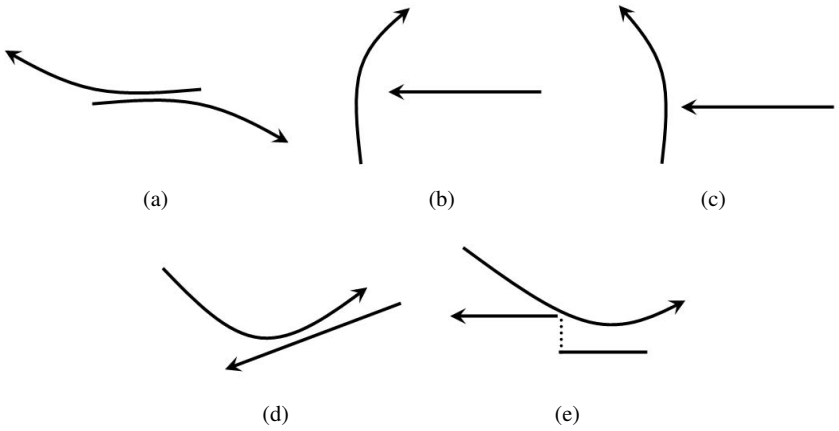


Fig. 15 Schematic representation of interaction between wave-fragments. (a) reflection, (b) attraction, (c) repulsion, (d) sliding, (e) shifting-sliding. From [4].

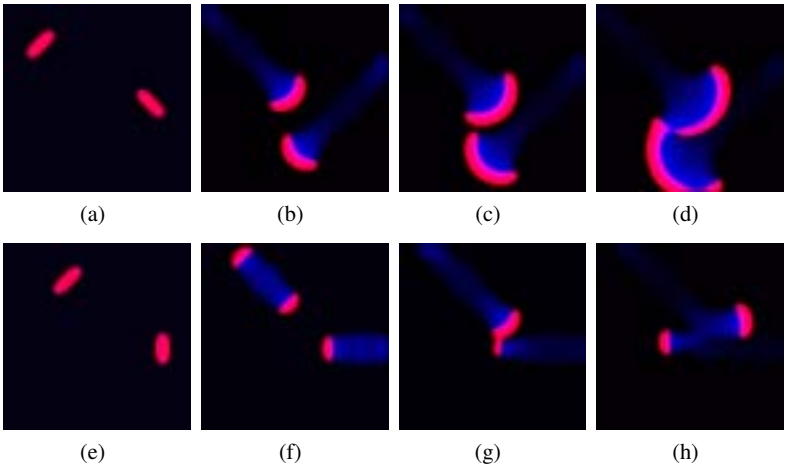


Fig. 16 (a)–(d) Sliding: snapshots of wave-fragment moving South-West colliding to wave-fragment traveling South-East. Center of the initial excitation of South-West wave-fragment is shifted southwards by 22 sites. From [4]. (e)–(h) Slide-shift: snapshots of wave-fragment moving South-East colliding to wave-fragment traveling West.

can be done strait-forwardly as already presented by bit encoding and implementation of logic gates.

From now on, only one dimensional space is considered because on the one hand, in higher dimension can —up to some point— be treated alike or correspond to what has been exemplified in previous sections and because on the other hand, dimension one is particularly restrictive and needs special focus.

The reader interested in CA might consult [29, 32, 45] for various topics not covered here.

4.1 Signals in CA

In space-time diagrams, or orbits, configurations are concatenated as iterations go. They are very important in oder to comprehend the dynamics. Since the temporal coordinate is added, this leads to an array with one more dimension than the underlying space.

A signal is anything periodic in the space-time diagram. If the periodicity is in two directions, then can fill the whole space-time and is referred as a *background*, the substrata upon which signals move. The frontier between two backgrounds is a signal as long as it is periodic. Figure 17 provides an example with a complex background (alternatively 0111, 1000, 1101 and 0010 repeating) on the left. On the right, signals are revealed by an ad hoc filtering. This example illustrates the variety in width and pattern of signals. As it can be seen, collisions between signals can be pretty complicated.

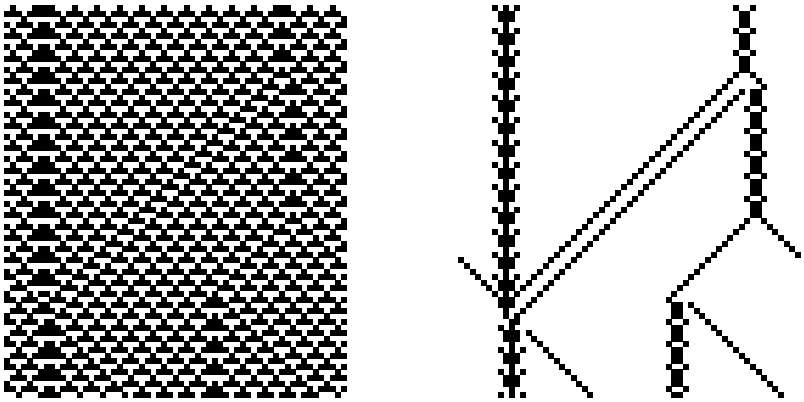


Fig. 17 Filtering to exhibit signals and backgrounds on rule 54 (inspired from [13, Fig. 7]). Time is increasing upwards.

4.2 Computing in one dimensional systems

In dimension two and above, as soon as it is possible to carry information around (with signal), to make information crossing and duplication and to process it (with collision) in the proper way to yield a sufficiently large set of logical gates then computation is straightforwardly possible.

In dimension one, this is not so easy because there is no way for a signal to go around another one or some static artifact. This means that signal propagation is really an issue and, for example, handling garbage signals can be cumbersome. One way to cope with it is to have various kinds of signals not identically affected by artifacts (some would pass unaffected while others interact).

This means that to use the bits and gates scheme, one has to be very careful with available signals and have a very clever positioning of the logic circuitry which is always thought of as two dimensional (time often provides the extra dimension for displaying it). For example, the construction of Ollinger [39] provides a small CA that is able to compute with circuit implementation where displaying the circuit is not straightforward.

Another way to tackle computation is to use other formalisms. One classical way is to go back to Turing machines, another one is to implement minimal rewriting systems like cyclic tag systems.

4.2.1 Turing machine

A Turing machine is a finite automaton acting on a potentially infinite array of symbols. The computation starts with the automaton in its initial state and the input written on the tape. The tape is accessed through a read/write head. Each iteration, the automaton reads the symbol under the head, rewrites a symbol, changes its state and moves the head left or right. A Turing machine is basically one-dimensional as its tape, so that it naturally fits on a one dimensional CA.

The simulation is rather simple: signals encoding the symbols are motionless and one signal amounting for the state of the automaton is moving round updating the symbols. Collisions are as follows: a moving state meet a stationary symbol, this leaves a new stationary symbol and a moving new state. (Fig. 24 in the next section provides a clear illustration of this in a continuous setting.) This was implemented by, for example, Lindgren and Nordahl [34].

4.2.2 Cyclic tag systems

Another way to define computation is to rely on a word which is rewritten until the system halts. The input is given as the starting word and the output is the final word. Connexion with Turing machine is straightforward when considering the tape as a finite word and the state of the automaton as an extra symbol where is the head.

Cyclic tag systems (CTS) is a particular case of many Turing-universal rewriting systems. A CTS considers a binary word together with a circular list of binary words (*appendants*). Each iteration the first bit is removed from the word and if it is 1 then the first appendant of the list is added at the end of the word, then the list is rotated. The computation stops when the word is empty or a special halting word (denoted `h` in the example) is activated. An example is provided on Fig. 18.

```
1011          011:h:011:01
 011011       h:011:01:011
  11011       011:01:011:h
   1011011    01:011:h:011
    01101101  011:h:011:01
     1101101  h:011:01:011
      101101
```

Fig. 18 Example of computation of a CTS (given on the first line).

Cyclic tag systems were proven able to perform any computation [14] and even to do so in polynomial time [51] and were used to built very small universal Turing machines [38]. They have been implemented in one dimensional CA with collision/signal based computing by Cook [14] to produce computation universal CA with only two states (it is impossible to compute with less). The construction relies on signals moving and colliding that are seek and classified in order to work on a more abstract level.

As far as minimal CA are concerned, it is worth mentioning that 4 states are enough to get a one dimensional CA able to simulate any other one dimensional CA [44]. This is not Turing universality since the simulation encompasses infinite configurations.

4.3 Signal specific issues

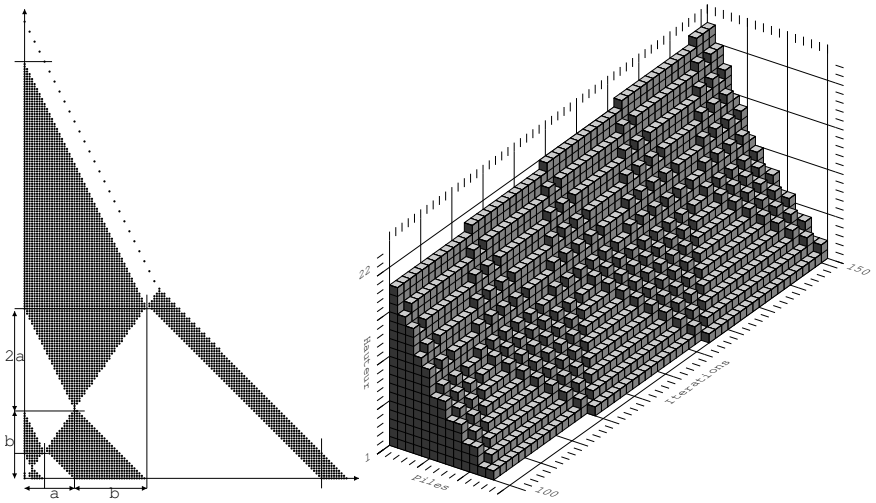
Since CA form a model for physical phenomena and also for parallel computing architectures, other issues arise. One is to understand the underlying dynamics when used as a discrete model. The other is to design CA for special purposes.

4.3.1 Signals to understand dynamics

Once a simulation is running as a CA, one common way to understand the dynamics is to try to find regularities in the space-time diagram and observe them. They are often the key to the underlying dynamics and then to predict. (Although it might happen that what is observed is nothing but an artifact of the model and has no counterpart in reality.) This is made clear by considering two examples.

The first one models sand dripping in a one dimensional space. The dynamics is quite simple: there is nothing in the initial configuration and each iteration, one grain is added to the first pile/cell. Each time a pile has at least two more grains than the nest one, a grain falls.

If only grains dropping at odd time are colored in black, then their position is as on Fig. 19(a) after 10000 iterations and only the top grains will ever move. This strange disposal as well the two different slopes and precise long term behavior are explained by signals [21, 22]. These signals can be identified on Fig. 19(b) where the successive configurations (iteration 100 to 150) are set one after the other to form a volume. On this volume, triangles can be seen on the lower (as well as the upper parts), their frontiers are signals (which can be revealed with an appropriate filtering).



(a) Dotting even grains [21, Fig. 6] at iteration 10000

(b) Successive configurations [21, Fig. 3]

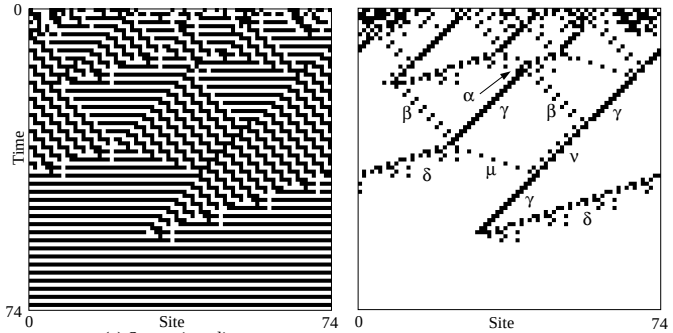
Fig. 19 One dimensional sand dripping.

Another example is provided by [15]. The aim is to generate a CA with 2 states and 5 closest cells neighborhood, such that, on a ring of any size whatever configuration it is started on, it always ends up blinking: all cells are 0 then all cells are 1, alternatively, forever. Instead of trying to built it straightaway, evolutionary programming is used: random CA are generated; they are ranked according to their "blinking capability"; then the best are kept, recombined and mutated to form the next generation. It then cycles through ranking and next generation until one "fully blinking" CA emerges.

The obtained CA is analyzed in term of signal (again with some filtering) as illustrated on Fig. 20. (The apparent non-connectivity of some signals comes from

the filtering and also because cells can influence one another up to distance two.) The evolutionary process goes in steps where various intermediate levels of "blinking" appear. Analyzing typical members of each level reveals the progressive apparition of signals and collision rules.

It is very important to notice how, in a context where signals were not asked for by the evolutionary process they indeed appear and are the key both to the desire dynamics and to the evolutionary process.



(a) Space-time diagram. (b) Filtered space-time diagram.

Fig. 20 Figure and filtering to highlight signals and analyze (from [15, Fig. 1]). Time is increasing downwards.

The previous example does not compute in the classical understanding, nevertheless, it provides a relevant dynamical global property.

4.3.2 Signals to generate a particular behavior

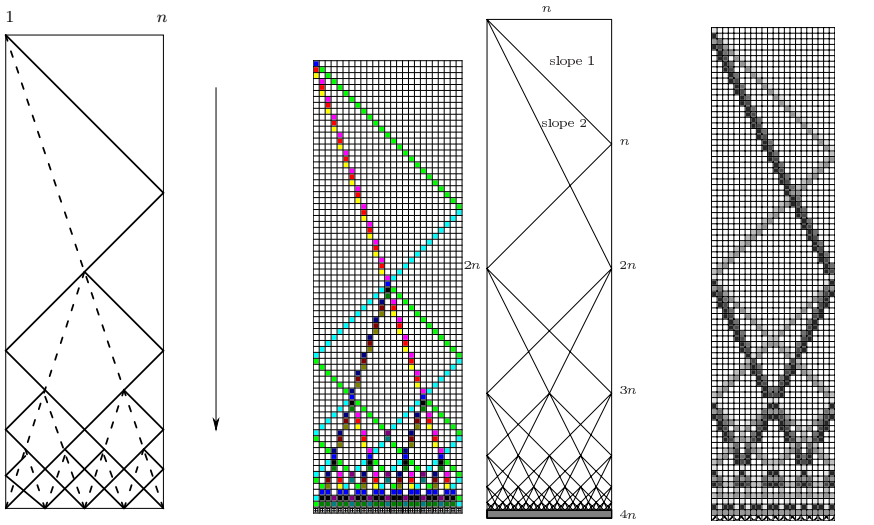
Computing in the usual understanding is a special behavior. But CA can also be thought of as a computing model on its own. Then primitives can be specially defined to take advantage of the huge parallelism. Signals are the key to manage it.

As already cited, the pioneer work of Fischer [25] generates prime numbers by a parallel implementation of the Sieve of Eratosthenes where they are indicated as no signal on the first cell at the corresponding times. Prime numbers can then be read on the space-time diagram by observing the state of the first cell.

Another complex behavior is the famous Firing Squad Synchronization problem (FSS). Starting from all but one cell in a quiescent state, one wants all the cells to enter simultaneously the same state —which has not been used before. Like if they would all blink for the very first time synchronously.

Each example of Fig. 21 shows the Euclidean conception and then the discrete implementation of a FSS solution. Both construction relies on a recursive cut in halves. In the discrete implementation, at some point, the granularity of space —a cell cannot be divided— is reached. Since CA are synchronous, this point is reached

simultaneously everywhere, ensuring the global synchronization of the entire array of cells.



(a) (b) $l = 25$ (a) The skeleton (b) A line of length 22

(a) divide and conquer in $3n$-steps [54, Fig. 2.3] (b) 8-states and 4n-steps [55, Fig. 1]

Fig. 21 FSS implementations.

4.3.3 Signals as a computational model on its own

Mazoyer et al. developed a computing model where multiplication (an on Fig. 22(a)), composition (an on Fig. 22(b)) and even iteration are graphically achieved. The programming system is achieved by using trellis of adaptable size that can be dynamically generated. The computation is carried out over the trellis. Composition is then achieved by having the next computation displayed on a new trellis. Recursion is provided by a scheme that dynamically starts another iteration providing each time an adapted trellis.

5 Abstract Geometrical Computation

Abstract Geometrical Computation (AGC) is an idealization of collision based computing where particles/signals are dimensionless (time and space are continuous). Although the number of signals in a bounded space is expected to be finite, it is unbounded. Another way to consider AGC is as a continuous counterpart of CA as

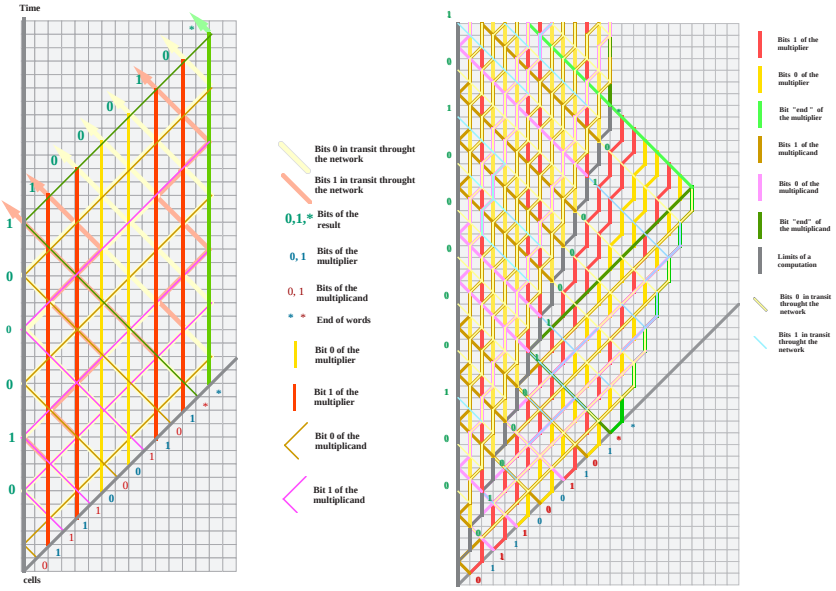


Figure 4: Multiplying 110011 by 10110.

(a) Multiplication [37, Fig. 4]

Figure 8: Computing $(ab)^2$.

(b) Composition of multiplications [37, Fig. 8]

Fig. 22 Mazoyer's system. Time is increasing upwards.

a limit when the size of the cells tends to zero. In CA, signals are almost always the key to understand and design, and indeed, in the literature, the discreteness of CA and of their signals is often put aside to reason at a more abstract level and left to technical details.

In abstract geometrical computation, dimensionless signals moves at constant speed. When they meet, they are replaced by others according to some rewriting rules. A *signal machine* (SM) gathers the definition of the nature of available signals (called *meta-signals*), their speeds and the collisions rules. There are finitely many meta-signals and each one is assigned a constant speed (velocity and direction). The constant speed may seem surprising, but each discrete CA signal has indeed a constant speed.

The space-time diagrams generated are continuous in both space and time and the traces of signals form line segments. For a given meta-signal, all these segments are parallel. In this section, only 1 dimensional AGC are considered so that the space-time diagram is always two dimensional with time increasing upwards as illustrated by Fig. 23.

Space-time diagrams can be much more complicated and, as presented below, use continuity up to implement the Zeno paradox and emulate the Black hole model of computation.

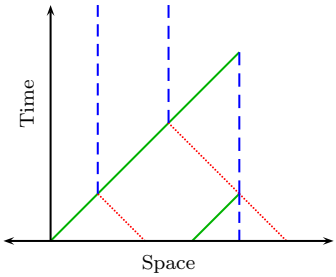


Fig. 23 Example of a space-time diagram.

5.1 Computing

Computing in the classical understanding is pretty easy. In fact, many constructions for CA directly translate to ACG and are even easier since discreteness does not have to be taken into account. Two ways to achieve computability are presented: Turing machines and Cyclic Tag Systems.

For Turing machines (as presented in Subsubsec. 4.2.1), the implementation is quite straightforward: static signals encode the tape —one signal per symbol— and one signal encodes the state of the automaton and the position of the head. The later moves forth and back from symbol to symbol as the read/write head. This is depicted on an example on Fig. 24. The construction is detailed in [20].

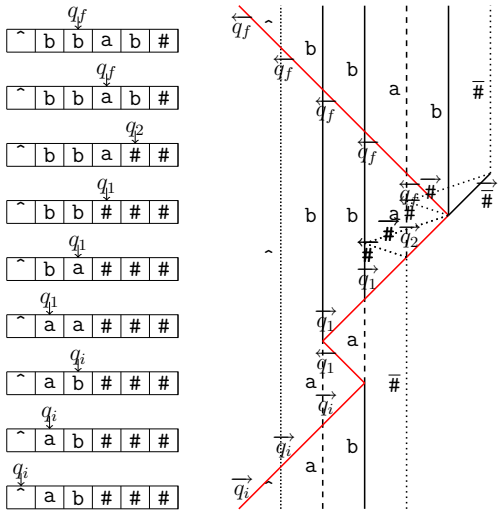
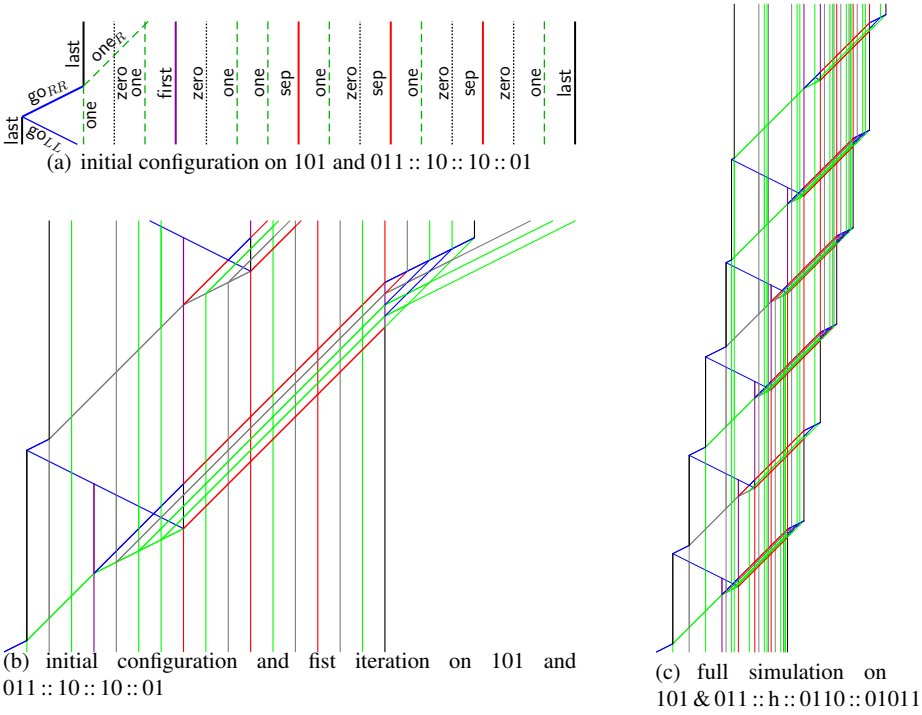


Fig. 24 Simulating a Turing machine.

Cyclic Tag Systems are presented in Subsubsec. 4.2.2. The simulation is done by encoding, left to right, the word and then the circular list of appendants by parallel signals encoding the bits as shown on Fig. 25(a). Each iteration, the list is rotated to the right, but before that, if the erased bit of the word is 1 a copy is left. The copy is directly added at the right of the word as on Fig. 25(b) which present one full iteration. The full simulation of the example of Fig. 18 is given on Fig. 25(c). The rightward drifting corresponds to the erasure of the word from the left and to the rightward rotation of the list. Each group of oblique lines corresponds to one rotation of the list.

This simulation is detailed in [19]. The signal machine obtained is able to simulate any CTS, and in thus Turing universal. It is the smallest such one known (it has only 13 meta-signals).



(a) initial configuration on 101 and 011 :: 10 :: 10 :: 01

(b) initial configuration and fist iteration on 101 and 011 :: 10 :: 10 :: 01

(c) full simulation on 101 & 011 :: h :: 0110 :: 01011

Fig. 25 Simulating a cyclic tag system.

5.2 Geometric primitives

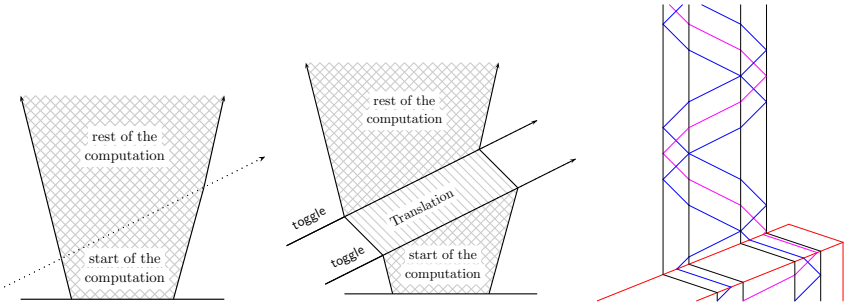
As for CA, AGC can also be considered as a model on its own with particular operators that might not have any classical computation counterpart (like FSS) nor discrete counterpart.

All the following constructions involve to add meta-signals and rules to a given SM so that the desire capability exists. Signals have to be added to the initial configuration in order to fire the effect. Rules can also be modified in order to dynamically fire it.

Space and time are continuous and scale-less so that since signals have no dimension, there is no limit on the scalability. If all signals are scaled spatially by a given coefficient, the whole computation is also scaled temporally. So that the duration of a computation is meaningless and complexity measures such as the maximal number of collisions with a causal link has to be used.

Spatial rescaling of the initial configuration is a static operation. It is also possible to do it during the computation. The construction relies on the ability to freeze a configuration.

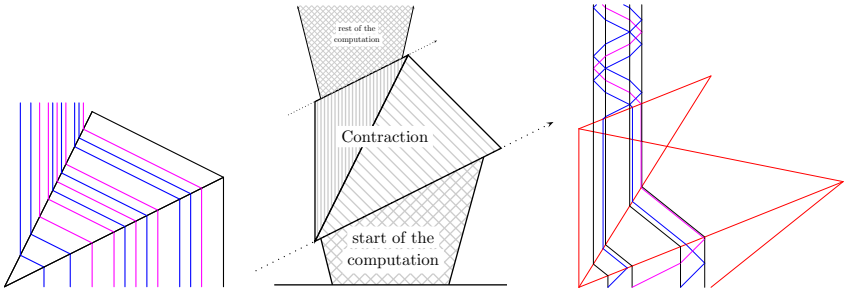
Freezing a computation is quite simple. One signal is added on one side and it crosses the entire configuration. Doing so, each time it meets a signal (or enters in a previously existing collision), it replaces it by another signal encoding the meta-signal. All the encoding signals have the same speed: they are parallel and do not interact and moreover the distance between them is preserved. The configuration is frozen and shifts. To unfreeze it, a signal comes from the side, crosses the configuration, and replaces each encoding signal by the encoded one (or the result of the collision). Freezing and unfreezing signals must have the same speed so that the configuration is restored exactly as it was, up to a translation. (And indeed they correspond to the same meta-signal toggle.) This is depicted on Fig. 26 where the scheme is presented as well as an example on the right.



(a) normal computation (b) translated computation (c) example

Fig. 26 Freezing and unfreezing.

Meanwhile a configuration is frozen into parallel signals it is possible to act on these signals. One simple trick is to change their direction. On Fig. 27(a), this is done twice so as to restore the original direction. (An extra signal is used on the right to delete the structure.) As a result, since different slopes are used to change direction, the distances between signals are scaled (here by one half). Adding freezing and unfreezing signals (automatically fired in due time) an artifact to scale down a whole configuration is generated as it can be seen on the Fig. 27(c).



(a) principle (b) scaled computation (c) example

Fig. 27 Scaling.

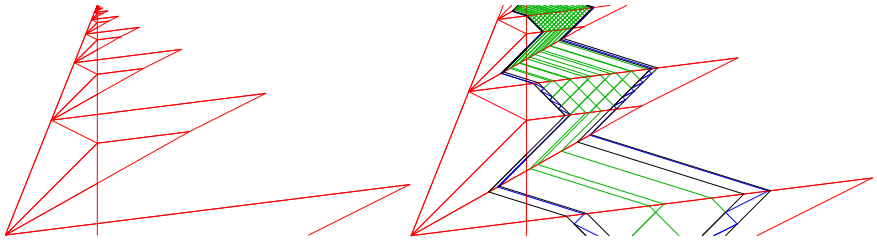
The initial configuration has been modified in order to start straightaway the effect. It is also possible to fire it dynamically during the computation when some special collision happens. This is used to iterate it.

5.3 Infinite acceleration and non recursive function

Scaling can be automatically restarted ad infinitum, since both space and time are continuous, this is not a problem —at least before the singularity. In the example of Fig. 28, a computation producing an infinite trellis extending indefinitely in both space and time in folded into the structure. The right signal and the middle one are very important since the folded computations should be bounded in order to ensure that is fully scaled.

A *singularity* refers to an accumulation of infinitely many collisions and signals to a given location. Figure 28(a) shows that the structure alone already create a singularity. This illustrates the Zeno paradox. The leftmost signals of Fig. 28(a) form an infinite sequence with infinitely (yet countably) many collisions although the time elapsed as well as the total distance crossed by signals is finite.

All the signals and collisions that would have existed if there would have been no folding indeed exist, but in a bounded portion of the space and time diagram. (The



(a) folding structure (b) example

Fig. 28 Folding or infinite rescaling.

equivalence of the plane and a bounded part is somehow implemented.) So that up to the presence of the structure and of the frozen parts, and the different scales the computation is the same.

Any computation starting with finitely many signals can be folded. Inside the structure, the rescaling provide speedup (the closer the faster they collide). This speedup is unbounded and goes to infinity. Inside the structure there is a space-time where time-lines are infinitely accelerated compared to the outside. This is the first step to emulate the Black-hole model of computation [23, 28, 35].

The second step it to find a way for an atomic piece of information to leave the black hole, here the singularity. One meta-signal can be added and rules changed so that the new meta-signal is not affected by the structure but can be generated by the initial computation.

The final step is to add bounding signals on both side of the folding (called here horizonLe and horizonRi). At their collision point, the folded computation is entirely in the casual past as displayed on Fig. 29. Any signal leaving the folding would have been collected.

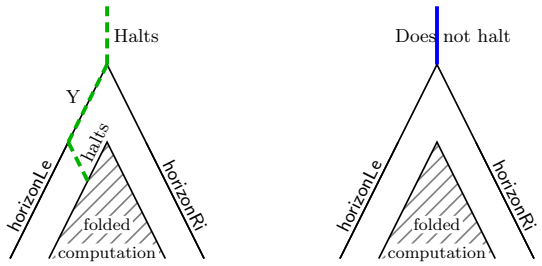


Fig. 29 Framing the folding to collect all signals exiting the folding.

This way the black hole model is emulated: there are two time-lines, one for the machine and one for the observer. The machine one is infinite. On the observer one, after a finite duration, the whole machine one is entirely in the casual past. A single piece of information can be sent by the machine to the observer and it has

to be sent after a finite (machine) duration. The ultimate date for the observer to receive anything from the machine is finite and known. To understand the power of this model just imagine that the machine only send a signal if its computation ends. So that the observer received a signal only if the it stops and after a duration the observer is aware of, any signal sent would have been received. So that just by checking its clock, the observer knows that the computation did not halt when it is the case. Altogether, the halting problem (whose undecidability is the cornerstone of Turing computability theory) is decidable!

This model, as well as AGC, clearly computes beyond Turing. In formal term it can decide Σ_0^1 formulae in the arithmetical hierarchy, i.e. a recursive/computable predicate prefixed by an existential quantifier. This includes, for example, the consistency of Peano arithmetic and Set theory, many conjectures such as the Collatz conjecture and Goldbach's conjecture.

Defining what happens at the singularity point is not easy especially since the accumulating set can be a line segment, a fractal curve or even a Cantor! (To understand this, take a continuous look at the FSS: what would happen at the bottom of Fig. 21?) For singularity on a single point, a careful continuation has been proposed in [20] that allows to climb the arithmetical hierarchy. There is also another use of isolated singularity as a way to provide limits for analog computation.

5.4 Analog computation

Unlike collision based computing and CA, with dimensionless signals in a continuous space, it is possible to encode reals as the distance between signals. In the AGC context, this distance is exact, i.e. it is a real number in exact precision. As long as signals are parallel, this distance is preserved. This allows to encode reals and to make some computations over them.

Since any real can be encoded exactly, the model ipso facto falls out of classical computability (because of cardinalities, there is no way to encode all the reals with natural numbers or finite strings). So that an analog model of computation has to be searched for.

With this encoding of real numbers, AGC is equivalent to linear Blum, Shub and Smale model (BSS) [12]. In the linear BSS model, variables holds (exact) real numbers and the operations available are addition, multiplication by a constant and branch according to the sign of a variable. This equivalence is true as long as the BSS machine has an unbounded number of variables (accessed through a context shift like moving a window over an infinite array of variables), there are finitely signals and there is no singularity [17].

If singularities are used in a proper way, it becomes possible to multiply two variables. Then the classical BSS model can be implemented in AGC [18]. The simulation is not possible in the other way round anymore since, for example, the exact square rooting can also be computed by AGC.

Like in the discrete case, a proper handling of isolated singularities of any order can be used to decide quantified (over natural but not real numbers) predicates in the BSS model and climb the BSS-arithmetical hierarchy [20].

References

[1] A. Adamatzky. *Novel materials for collision-based computing*. Springer, 2002.

[2] A. Adamatzky. Collision-based computing in Belousov-Zhabotinsky medium. *Chaos, Solitons & Fractals*, 21:1259–1264, 2004.

[3] A. Adamatzky, editor. *Collision based computing*. Springer, 2002.

[4] A. Adamatzky and B. de Lacy Costello. Binary collisions between wave-fragments in sub-excitable Belousov-Zhabotinsky medium. *Chaos, Solitons & Fractals*, 34:307–315, 2007.

[5] A. Adamatzky and A. Wuensche. Computing in spiral rule reaction-diffusion hexagonal cellular automaton. *Complex Systems*, 16(4), 2007.

[6] A. Adamatzky, A. Wuensche, and B. de Lacy Costello. Glider-based computation in reaction-diffusion hexagonal cellular automata. *Chaos, Solitons & Fractals*, 27:287–295, 2006.

[7] C. Anastassiou, J. W. Fleischer, T. Carmon, M. Segev, and K. Steiglitz. Information transfer via cascaded collisions of vector solitons. *Optics Lett.*, 26: 1498–1500, 2001.

[8] A. J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers*, EC-14(1):394–399, 1965.

[9] E. Banks. *Information and transmission in cellular automata*. PhD diss., MIT, 1971. Cited by [48].

[10] V. Beato and H. Engel. Pulse propagation in a model for the photosensitive Belousov-Zhabotinsky reaction with external noise. In L. Schimansky-Geier, D. Abbott, A. Neiman, and C. van den Broeck, editors, *Noise in Complex Systems and Stochastic Dynamics (Proc. SPIE 2003)*, 2003.

[11] E. R. Berlekamp, J. H. Conway, and R. L. Guy. *Winning Ways for your Mathematical Plays*, volume 2 Games in particular. Academic Press, 1982.

[12] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer, New York, 1998.

[13] N. Boccara, J. Nasser, and M. Roger. Particle-like structures and interactions in spatio-temporal patterns generated by one-dimensional deterministic cellular automaton rules. *Phys. Rev. A*, 44(2):866–875, 1991.

[14] M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15: 1–40, 2004.

[15] R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson. Evolving globally synchronized cellular automata. In L. J. Eshelman, editor, *International Conference on Genetic Algorithms '95*, pages 336–343. Morgan Kaufmann, 1995.

[16] M. Delorme and J. Mazoyer. Signals on cellular automata. In A. Adamatzky, editor, *Collision-based computing*, pages 234–275. Springer, 2002.

[17] J. Durand-Lose. Abstract geometrical computation and the linear Blum, Shub and Smale model. In S. Cooper, B. Löwe, and A. Sorbi, editors, *Computation and Logic in the Real World, 3rd Conf. Computability in Europe (CiE '07)*, number 4497 in LNCS, pages 238–247. Springer, 2007.

[18] J. Durand-Lose. Abstract geometrical computation with accumulations: beyond the Blum, Shub and Smale model. In A. Beckmann, C. Dimitracopoulos, and B. Löwe, editors, *Logic and Theory of Algorithms, CiE 2008 (abstracts and extended abstracts of unpublished papers)*, pages 107–116. University of Athens, 2008.

[19] J. Durand-Lose. Abstract geometrical computation: small Turing universal signal machines. In T. Neary, A. Seda, and D. Woods, editors, *International Workshop on the Complexity of Simple Programs, Cork, Ireland, December 6-7*. Cork University Press, 2008.

[20] J. Durand-Lose. Abstract geometrical computation 3: Black holes for classical and analog computing. *Natural computing*, 2009. to appear.

[21] J. Durand-Lose. Grain sorting in the one dimensional sand pile model. *Complex Systems*, 10(3):195–206, 1996.

[22] J. Durand-Lose. Parallel transient time of one-dimensional sand pile. *Theoret. Comp. Sci.*, 205(1–2):183–193, 1998.

[23] G. Etesi and I. Németi. Non-Turing computations via Malament-Hogarth space-times. *Int. J. Theor. Phys.*, 41(2):341–370, 2002. gr-qc/0104023.

[24] R. J. Field and R. M. Noyes. Oscillations in chemical systems. iv. limit cycle behavior in a model of a real chemical reaction. *J. Chem. Phys.*, 60:1877–84, 1974.

[25] P. C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12(3):388–394, 1965.

[26] E. F. Fredkin and T. Toffoli. Design principles for achieving high-performance submicron digital technologies. In A. Adamatzky, editor, *Collision-based computing*, pages 27–46. Springer, 2002.

[27] E. F. Fredkin and T. Toffoli. Conservative logic. *Int. J. Theor. Phys.*, 21, 3/4: 219–253, 1982.

[28] M. L. Hogarth. Non-Turing computers and non-Turing computability. In *Biennial Meeting of the Philosophy of Science Association*, pages 126–138, 1994.

[29] A. Ilachinski. *Cellular automata –a discrete universe*. World Scientific, 2001.

[30] M. H. Jakubowski, K. Steiglitz, and R. K. Squier. When can solitons compute? *Complex Systems*, 10(1):1–21, 1996.

[31] M. H. Jakubowski, K. Steiglitz, and R. K. Squier. Computing with solitons: A review and prospectus. *Multiple Valued Logic*, 6(5–6):439–462, 2001.

[32] J. Kari. Theory of cellular automata: a survey. *Theoret. Comp. Sci.*, 334:3–33, 2005.

[33] H. J. Krug, L. Pohlmann, and L. Kuhnert. Analysis of the modified complete oregonator (mco) accounting for oxygen- and photosensitivity of Belousov-Zhabotinsky systems. *J. Phys. Chem*, 94:4862–66, 1990.

[34] K. Lindgren and M. G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.
[35] S. Lloyd and Y. J. Ng. Black hole computers. *Scientific American*, 291(5): 31–39, November 2004.
[36] N. Margolus. Physics-like models of computation. *Phys. D*, 10:81–95, 1984.
[37] J. Mazoyer. Computations on one dimensional cellular automata. *Ann. Math. Artif. Intell.*, 16:285–309, 1996.
[38] T. Neary and D. Woods. Four fast universal Turing machines. *Fund. Inf.*, 2009.
[39] N. Ollinger. The quest for small universal cellular automata. In *ICALP '02*, number 2380 in LNCS, pages 318–329. Springer, 2002.
[40] D. Rand and K. Steiglitz. Computing with solitons. In *Encyclopedia of Complexity and Systems Science*. Springer, 2009.
[41] D. Rand, K. Steiglitz, and P. Prucnal. Signal standardization in collision-based soliton computing. *Int. J. of Unconventional Computing*, 1:31–45., 2005.
[42] P. Rendell. Turing universality of the game of life. In A. Adamatzky, editor, *Collision-based computing*, pages 513–540. Springer, 2002.
[43] J.-P. Rennard. Implementation of logical functions in the game of life. In A. Adamatzky, editor, *Collision-based computing*, pages 491–512. Springer, 2002.
[44] G. Richard and N. Ollinger. A particular universal cellular automaton. In T. Neary, D. Woods, A. K. Seda, and N. Murphy, editors, *The Complexity of Simple Programs*. National University of Ireland, Cork, 2008.
[45] P. Sarkar. A brief history of cellular automata. *ACM Computing Surveys*, 32 (1):80–107, 2000.
[46] I. Sendiña-Nadal, E. Mihaliuk, J. Wang, V. Pérez-Muñuzuri, and K. Showalter. Wave propagation in subexcitable media with periodically modulated excitability. *Phys. Rev. Lett.*, 86:1646–49, 2001.
[47] K. Steiglitz. Time-gated manakov spatial solitons are computationally universal. *Phys. Rev. E,*, 63:1660–8, 2001.
[48] T. Toffoli and N. Margolus. *Cellular Automata Machine - A New Environment for Modeling*. MIT press, Cambridge, MA, 1987.
[49] J. J. Tyson and P. C. Fife. Target patterns in a realistic model of the Belousov-Zhabotinsky reaction. *J. Chem. Phys.*, 73:2224–37, 1980.
[50] A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, 1966.
[51] D. Woods and T. Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '06), Berkeley, CA, USA*, pages 439–448. IEEE Computer Society, 2006.
[52] A. Wuensche. Glider dynamics in 3-value hexagonal cellular automata: The beehive rule. *Int. J. of Unconventional Computing*, 1:375–398, 2005.
[53] A. Wuensche and A. Adamatzky. On spiral glider-guns in hexagonal cellular automata: activator-inhibitor paradigm. *Int. J. Modern Phys. C.*, 17(7):1009–26, 2006.

[54] J.-B. Yunès. *Automates Cellulaires; Fonctions Booléennes*. Habilitation à diriger des recherches, Université Paris 7, 2007.

[55] J.-B. Yunès. Simple new algorithms which solve the firing squad synchronization problem: a 7-states 4n-steps solution. In J. Durand-Lose and M. Margenstern, editors, *Machine, Computations and Universality (MCU '07)*, number 4664 in LNCS, pages 316–324. Springer, 2007.