second-order cellular automata into lattice gases

Tommaso Toffoli (tt@bu.edu)
Electrical and Computer Engineering, Boston University, Boston, MA 02215
Silvio Capobianco (capobian@mat.uniroma1.it)
Istituto Matematico, Università di Roma "La Sapienza"
Patrizia Mentrasti (mentrasti@mat.uniroma1.it)
Istituto Matematico, Università di Roma "La Sapienza"

1 Introduction

To undo a reversible process, intuitively all one has to do is

Recipe 1

Perform the inverse operations in the reverse order.

This is one of those fortunate cases in mathematics where a concept is so simple and clear that a formal prescription (see §3 for the systems of interest to us) has little to add to the intuitive recipe.[1]

Unfortunately, the way certain dynamical systems are customarily presented[2] makes it effectively impossible, *even if the system is presumed to be invertible*, to use the above recipe as means to automatically derive the system's inverse dynamics—and thus to "run the system in reverse"—or, if the recipe turns out to be inapplicable, use this fact to conclude that the system is not, after all, invertible. What's worse, that way of presenting the system makes it difficult to see properties, such as conservation laws, that are intimately tied to the invertible nature of the system and that would become immediately manifest if the system were given a more appropriate presentation.

This is specifically the case with second-order cellular automata[8]—of which one of the best known is the Ising spin model of ferromagnetism (cf. [5]). Though certain similarities of behavior between specific second-order cellular automata and specific lattice gases had been noticed,[3] no definite correspondence rules between the two classes of systems has yet been proposed. In this paper we give an explicit construction for transforming such cellular automata into lattice gases. In the latter, invertibility is a *structurally manifest* feature; thus, either the inversion scheme of Recipe 1 is immediately applicable—with all the advantages that that entails, or one can conclude that the cellular automaton was not invertible.

Incidentally, for the purposes of this paper we had to come up with the "right" definition of 'lattice gases' (which as far as I know had never been formally defined). We propose this definition as the canonical one.

2 Preliminaries

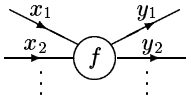
Computation is the exercise of *function composition* in a context where the set of building blocks—*interaction* and *interconnection* primitives—is specified once and for all, so that the originality of the composition lies not in the introduction of novel components but in obtaining the desired behavior by using only the given primitives (though in as large a number of occurences as desired). In brief, what is constrained is not the *number* of elements to be used but only their *kind*.

We shall deal with functions $f : X \to Y$ in which both the domain X and the codomain Y are Cartesian products of sets, namely, $X = X_1, X_2, \ldots$ and $Y = Y_1, Y_2, \ldots$, so that the mapping $x \overset{f}{\longmapsto} y$ takes the form

$$\begin{aligned} &(x_1, x_2, \ldots) \overset{f}{\longmapsto} (y_1, y_2, \ldots), \\ \text{or} \quad &(y_1, y_2, \ldots) = f(x_1, x_2, \ldots). \end{aligned} \tag{1}$$

In other words, both argument and result of a function may be ordered collections of variables rather than individual variables.

It will be convenient to express the above mapping in graphic form as



. (2)

The mapping itself can be thought of as an *equation* relating its input and output variables. A system of n *simultaneous* equations will be expressed by combining n diagrams like (2) into a **function composition graph**, i.e., a directed graph in which each arc represents a *variable* and each node

[1] Incidentally, my students always get a kick from seeing me go directly from

$$y = \frac{2a-1}{1+\sqrt[3]{x^2-4}} \qquad \text{to} \qquad x = \pm\sqrt{(1/\tfrac{y}{2a-1} - 1)^3 + 4},$$

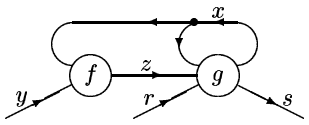
thus solving for x in a single pass. Of course I start with y and undo one-by-one—beginning with the last—the operations that had led from x to y.

[2] A **presentation**—of a group, an algebraic structure, a dynamics, and so forth—is a definition of it by means of a *structural* description ("how to put it together from parts"), as contrasted to an exhaustive raw tabulation ("a photograph of the finished product") or to a *functional* description ("what it does or is supposed to do"). A group, for example, may be presented—origami-like—by means of generators and relations; a finite-state machine is usually presented as a sequential network—an assembly of "gates" and "flip-flops"—whose size, in most practical cases, is much less that that of the full transition table.

[3] E.g., the Ising spin-glass dynamics [1].

$$\begin{cases} (x,y) \overset{f}{\longmapsto} z \\ (x,z,r) \overset{g}{\longmapsto} (x,s) \end{cases} \tag{3}$$

will be written as



. (4)

Remark that

- There is no implicit time dependency in (3) or (4); the variables represent single, timeless values—not time sequences. Specifically, diagram (4) *does not* represent a sequential circuit.

- In (4) we found it convenient to use as identifiers for the two nodes the two functions f and g respectively associated with them. In general, however, the same function may appear on more than one node, so that some other identifier may have to be used for distinguishing nodes.

- In (3), the variable x that appears as an output in the second equation g occurs as an input of both f and g. This circumstance is indicated in (4) by the ancillary "fanout" node (represented by a simple dot) which inputs x and outputs two copies of that variable. A more explicit (and physically more realistic) way of representing this situation is to treat the branches of the fanout node as two new variables, u and v, and add to (3) a third equation relating u and v to x, as follows,

$$\begin{cases} (u,y) \overset{f}{\longmapsto} z \\ (v,z,r) \overset{g}{\longmapsto} (x,s) \\ x \overset{I_2}{\longmapsto} (u,v) \end{cases} , \tag{3'}$$

where I_n is the n-**fanout** function $X \overset{I_n}{\longrightarrow} X^n$ defined by

$$x \overset{I_n}{\longmapsto} (\underbrace{x,\ldots,x}_{n}). \tag{5}$$

By using convention (3') for (3) and the corresponding graph (4)—where the fanout node thus becomes  to reflect the third equation of (3')—one obtains a **one-to-one** function composition scheme (cf. [6]), in which *at most one* input variable is identified with any given output variable; intuitively, any branching of signals can only take place at a node. Whether such a convention is used tacitly or explicitly, it is clear that, without loss of generality, *any* function composition scheme can be assumed to be one-to-one. A consistent reminder of one for each variable.[5]

Borrowing terms from relativity, let's call **signals** the arcs of a function composition graph and **events** its nodes. We say that event g **follows** event f if the two events coincide or if there is a sequence of signals that form a directed path (in terms of arcs' orientation) from f to g. (In (4), for example, event g follows f because signal z points from f to g, and conversely f follows g as per signal x.) The relation 'follows' is clearly *reflexive* and *transitive*.

A function composition graph is **causal** if the 'follows' relation is *anti-symmetric*, and thus induces a *partial order* between events. The graph is then *acyclic* (Fig. 1); intuitively, it admits of no "time loops." We shall call such a graph a **computational network**.

Even though a computational network may extend indefinititely over space and time—like a cellular automaton or a lattice gas—to model effective (in Turing's sense) computation one usually requires the network to be *locally finite*.[6]

For special classes of computation, the network may be required to obey additional constraints of topological or group-theoretical nature—such as locality of interconnection, number of dimensions, regularity, invertibility.

An arbitrary assignment of values to all the signals of a computing network is called a **history** on that network. A history is a **computation** if the values flowing in and out of each event satisfy the functional relationship specified by the event itself. In other words, a computation is a solution of the system of simultaneous equations that specifies the network.

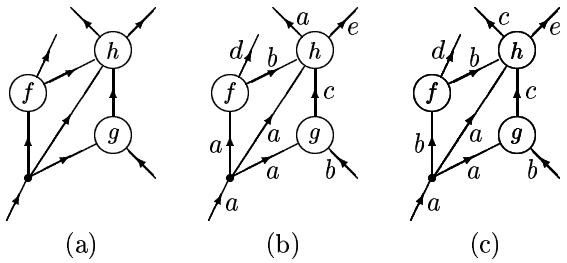


Figure 1: (a) A computational network. The function associated with each node is indicated; while the set of possible values associated with each arc is not explicitly indicated, it is implied by the functions of which that arc is an output or input. (b) A history on that network. The letters on the arcs denote specific constants. If the constants around every node satisfy the equation at that node (cf. (1)), than the history is a computation. (c) Another history on the same network. This is certainly not a computation, as the fanout equation (5) is not satisfied at the fanout node.

[4] In graph-theoretical terms, one would say that this is a *colored* and *labeled* graph. The "color" of an arc is the *set* associated with it (e.g., the set X in $f : X \to Y$); note that by 'variable' one simply means a symbol whose possible values range over that set. The "label" of a node is the *function* associated with it, which must have as domain the Cartesian product of the sets associated with its input arcs and as codomain the product of the sets associated with its output arcs.

[5] For example, in (4) there are three arrowheads around the fanout node, indicating three distinct variables—even though their values are asked to coincide by the third equation of (3').

[6] Typically, (a) Events are chosen from a given, finite repertoire of functions each having a finite number of input and output variables; and (b) Sets over which variables can range are chosen from a given, finite, repertoire of finite sets.

invertible functions. If that is the case, the **inverse** network, S^{-1}, is that obtained by

- reversing the orientation of all signals, i.e., exchanging the role of input and output variables, and
- replacing the function at each event by the inverse function—thus transforming an equation like (1) to

$$(y_1, y_2, \dots) \stackrel{f^{-1}}{\longmapsto} (x_1, x_2, \dots). \tag{6}$$

If H is a history on network S (Fig. 2a), a history on the inverse network S^{-1} is the **inverse history**, H^{-1}, if values on homologous signals in the two networks coincide (Fig. 2b).

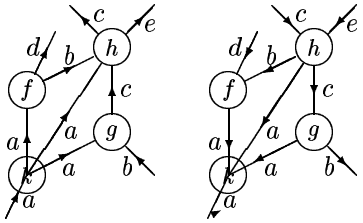


Figure 2: (a).

We conclude these preliminaries with a lemma that is but the precise expression of Recipe 1 when 'order' means *partial order*—thus showing that this recipe can be applied to *concurrent*[7] as well to *sequential* computation.

Lemma 1 *Let S be an invertible computational network. If a history H on S is a computation, then the inverse history H^{-1} on the inverse network S^{-1} is itself a computation—naturally called the* inverse computation.

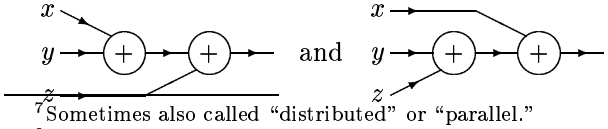
Intuitively, given the schematics for an invertible computer, this can be turned into the schematics for the inverse computer by a simple, strictly local transliteration.[8]

The proof is immediate if one considers that, if f is invertible, any assignement of values that turns (1) into an identity also turns (6) into an identity; consequently, any solution of the system of equations underlying an invertible computational network like Fig. 2a is also a solution of the system underlying Fig. 2b.

3 Nature of the problem

We shall now state the nature of the problem tackled by the present paper, first in the simplest possible setting—where the solution is, as we shall see, elementary—and then in the intended setting, i.e., invertible second-order cellular automata.

In essence, the issue will be one of *structure* vs *function*. For example, the two networks



, (7)

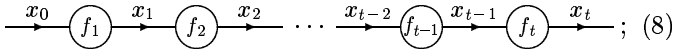
[7] Sometimes also called "distributed" or "parallel."

[8] Each element—arc or node—is converted to a new element indipendently of the rest of the network.

same function of x, y, z—have a different *structure*; in fact, they represent two distinct algorithms or "mechanisms" for computing $x + y + z$: in the first algorithm, x is added first; in the second, last.

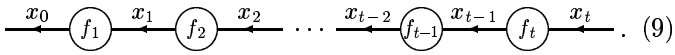
The concept of invertibility defined in the previous section and captured by Recipe 1 is based on structure, and is stronger than one based on function. We will sometimes emphasize this by saying that a network A which is the inverse (in that sense) of network B is the **structural** inverse of B. A **functional** inverse of a system, on the other hand, will be a system that displays the reverse partial order for the values that make up a history, regardless of its internal structure.

Let transformations $f_1, f_2, \dots, f_t$ be successively applied to a system, whereby starting from state x_0 we obtain a final state x_t. Explicitly,



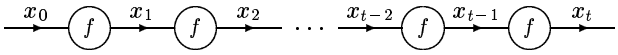
; (8)

If $f_1, f_2, \dots, f_t$ happen to be *invertible*, then we go back from x_t to x_0 by applying the inverse transformations in the reverse order as per Recipe 1:



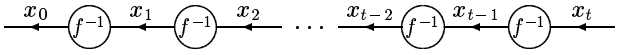
. (9)

An even simpler picture is provided by the *time-independent* dynamics $x_i \stackrel{f}{\longmapsto} x_{i+1}$, where all events are occurrences of the same function f,



, (10)

whose inverse $x_{i+1} \stackrel{f^{-1}}{\longmapsto} x_i$ yields

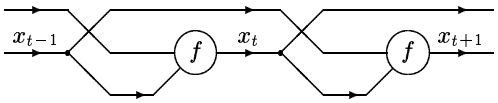


. (11)

Let us now consider the case of a *second-order* dynamical system, defined by the second-order recurrence relation

$$(x_{i-1}, x_i) \stackrel{f}{\longmapsto} x_{i+1}, \qquad x_i \in X, \tag{12}$$

or, equivalently, by the following indefinitely-extended computational network N:



. (13)

Whatever the function f, this network cannot possibly be an *invertible* computational network (as defined in §2). In fact, event f is of the form $X^2 \stackrel{f}{\longrightarrow} X$; since its domain and codomain have different number of elements, f cannot be invertible.[9] A similar argument applies to the fanout nodes, since the 2-fanout function is of the form $X \stackrel{I_2}{\longrightarrow} X^2$.

[9] We exclude from consideration, of course, the degenerate cases where X is the empty set or a singleton; in both cases the only candidate for f is trivial and trivially invertible, and similarly for the fanout function.

values into the two right-going signals x_{t-1} and x_t, the recurrence relation (12) would uniquely specify successive values for the entire forward sequence $x_{t+1}, x_{t+2}, \ldots$. On the other hand, if one tried to extend *backwards* the sequence $\ldots, x_t, x_{t+1}, \ldots$ by traveling along arcs in the reverse direction and trying to solve (12) or (5) for the inputs in terms of the outputs whenever encountering a node, one would in general meet *ambiguity* (underspecified outcomes) when traversing backwards an f node, and *conflict* (overspecified outcomes) when traversing backwards a fanout junctions.[10]

Yet, in spite of the the above disclaimers, the dynamical system (12) may still be "reversible" in some obvious sense. This happens when there exists a function g such that the sequence items

$$x_t, x_{t-1}, x_{t-2}, \ldots, x_1, x_0,$$

successively generated by the recurrence relation

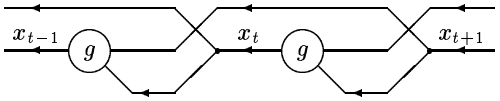
$$(x_{i+1}, x_i) \stackrel{g}{\longmapsto} x_{i-1}, \qquad x_i \in X, \tag{14}$$

from the initial values x_t, x_{t-1}, are the *same* as the items

$$x_0, x_1, \ldots, x_{t-2}, x_{t-1}, x_t$$

generated by (12) from initial values x_0, x_1, but in the *reverse* sequential order.

For such a g, the computational network $\bar{N}$ corresponding to recurrence relation (14), namely,



, (15)

is clearly a *functional* inverse of (13)—even though the latter is not structurally invertible.

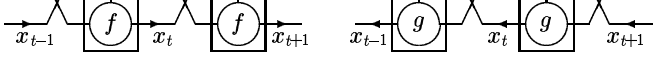
What properties must f and g satisfy in order to make up such a complementary pair? And, assuming that f and g are indeed such a pair, is there any way that we can somewhat restructure the corresponding networks (13) and (15) so as to obtain a pair of networks that, while retaining their original functional behavior, are strict *structural* inverses of one another? In sum, can we after all still manage to "explain" in terms of Recipe 1 the special functional relationship between systems (12) and (14)?

The following construction achieves that goal.

The first step of the construction is merely cosmetic preparation. In Fig. 3, where we show networks N and $\bar{N}$ (from diagrams (13) and (15)) side-by-side, the fanout junctions have been drawn closer to the nodes that follow them and a box has been drawn around each pair of nodes (remember that fanout is a node).

The second step involves a change in notation. Provided that networks N and $\bar{N}$ are functional inverses of one another, we shall rewrite the contents of the f node, which is presently a function of the form $f : X^2 \to X$, as a function

[10]This lack of invertibility of the nodes of network (13) at a logic level has a well-understood counterpart at the *physical* level[6].



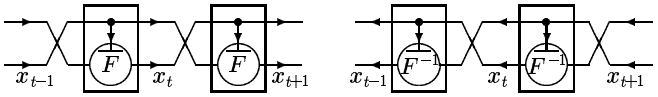
(a) network N (b) network $\bar{N}$

Figure 3: The networks N and $\bar{N}$ shown side-by-side. The fanout junctions have been drawn closer to the nodes that follow them, but without altering the network structure. Also, a box has been drawn around each pair of nodes.

of the form $F_i : X \to X$—in other words, one of the two arguments will be replaced by a parameter (the latter, of course, is really an argument in a different form). To this purpose, let us consider a collection $\{F_i\}$ of *invertible* functions of the form $F_i : X \to X$, where the index i as well ranges over X. If we define f and g in terms of the F_i as follows

$$\begin{aligned} f(i,j) &\equiv F_i(j), \\ g(i,j) &\equiv {F_i}^{-1}(j), \end{aligned} \tag{16}$$

we can formally rewrite N_1 and $\bar{N}_1$ as in of Fig. 4.



(a) network N (b) network $\bar{N}$

Figure 4: If networks N and $\bar{N}$ are functional inverses of one another, then the functions f and g can be replaced respectively by function families $\{F_i\}$ and $\{{F_i}^{-1}\}$, where the index i is a parameter that takes the place of the first argument. Now the variable represented by the vertical arrow sets this parameter (and thus determines which of the F_i is used for F) instead of being submitted to f as an ordinary argument as in Fig. 3. In the diagram, this is suggested by having the vertical arrow "reach into" the node so as to affect F itself. In spite of the different notation, the boxes in this figure and those in Fig. 3, if viewed as "black boxes," have identical behavior.

Proof. If f and g are defined as above in terms of $\{F_i\}$, then

$$(x_{t-1}, x_t) \stackrel{f}{\longmapsto} x_{t+1} \iff (x_{t+1}, x_t) \stackrel{g}{\longmapsto} x_{t-1} \tag{17}$$

and thus N and $\bar{N}$ are functional inverses of one another In fact we can set up the following chain of equivalences

$$\begin{aligned} f(x_{t-1}, x_t) = x_{t+1} &\iff F_{x_t}(x_{t-1}) = x_{t+1} \\ &\iff x_{t-1} = {F_{x_t}}^{-1}(x_{t+1}) \\ &\iff g(x_{t+1}, x_t) = x_{t-1}. \end{aligned} \tag{18}$$

The converse is also true; that is, if N and $\bar{N}$ are functional inverses, then there must exist a collection $\{F_i\}$ having the above properties.

Proof. Put $F_i(x) = f(i,x)$.

For each i, F_i is a function. In fact, let $x_1 = x_2$. Then, $f(i,x_1) = f(i,x_2)$ since f is a function; so $F_i(x_1) = F_i(x_2)$. Since x_1 and x_2 are arbitrary, F_i is a function.

For each i, F_i is invertible. Indeed, put $G_i(x) = g(i,x)$. G_i is a function for the same reason F_i is a function.

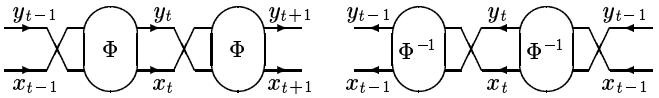
Fix a time t.

Fix x. Consider a history H such that $x_{t-1} = x$, $x_t = x$. Then $x_{t+1} = f(i,x) = y$. Let K be the history inverse of

$g(i,y) = x$. Since x is arbitrary, G_iF_i is the identity.

Now, fix y. Consider a history K that is the inverse of a history H where $x_{t+1} = y$, $x_t = x$. Then $x_{t-1} = g(i,y) = x$. But $y = f(i,x)$ since H is a history. Then, $F_i(G_i(y)) = f(i,g(i,y)) = f(i,x) = y$. Since y is arbitrary, F_iG_i is the identity.

As a final step, we shall replace the contents of each box in Fig. 3a by a *single*, invertible node, Φ, defined below, and each box in Fig. 3b by its inverse Φ^{-1}, thus obtaining new networks $\hat{N}$ and its inverse $\hat{N}^{-1}$ as in Fig. 5.



(a) network $\hat{N}$ (b) network $\hat{N}^{-1}$

Figure 5: The networks $\hat{N}$ and $\hat{N}^{-1}$. The latter is the *structural* inverse of $\hat{N}$ as well as a functional inverse.

(19)

involves a transformation from a one-variable, second-order recurrence scheme to a two-variable, first-order scheme; this transformation is close in spirit to the *Legendre transform* used in classical mechanics to go from the Lagrangian formalism (second-order differential equation in one variable) to the Hamiltonian formalism (first-order differential equation in two variables).

4 Scrap

But Recipe 1 applies equally well to the *most general* function-composition scheme, consisting, as explained in §2, of an acyclic directed graph (Fig. 1).

In this case the concepts of global time and "synchronous" global state are no longer applicable. However, a local state can still be defined on any spacelike surface, and the graph specifies the functional dependency between any two such states (x and y in Fig. 1).

If the network of Fig. 1 prescribes that $y = f(x)$, then the inverse relationship, $x = f^{-1}(y)$, is prescribed by the *same* network, provided that the order of evaluation is reversed (the arrows are followed backwards) and the functions at the nodes are replaced by their inverses. In other words, *functional* inversion is automatically obtained by performing a trivial *structural* inversion.

As a matter of fact, the difficulty we address in this paper arises with *any* second-order recurrence relation, such as the central-difference or "leapfrog" scheme

$$a_{t+1} = f(a_t) + a_{t-1}$$

(cf. [2]). A recurrence like the latter, however, involves only the time dimension, and can immediately be transformed into a first-order recurrence (where inversion *à la* 1 is immediate). Things are no longer so simple if besides time we also have *spatial* dimensions, as in cellular automata, and thus a recurrence over multiple indices.

Remark 1 *Given an arbitrary two-argument function $f : X \times Y \to Z$, for each $y \in Y$ consider the one-argument function $f_y : X \to Z$ defined by*

$$f_y(x) = f(x,y). \tag{20}$$

The collection f_y indexed by $y \in Y$ is equivalent to f itself; in fact, given x, y, and $f_y(x)$, (20) immediately yields $f(x,y)$. Intuitively, we have taken the table that defines f and broken it down into a collection of subtables, one for each possible value of the argument y.

Lemma 2 *The function $F : X \times Y \to Z \times Y$ defined by*

$$F(x,y) = (f_y(x), y)$$

is invertible iff each of the f_y is invertible.

Moreover, the above recipe applies just as well to *composite*, distributed systems, even indefinitely extended ones. If the overall evolution of a system is the result of *local* interactions, the history of such a system need no longer be thought of as a linear sequence of *global states*, connected by synchronous updating steps and totally ordered along a *global time*. Rather, updating of local states is carried out *concurrently*, by a lattice of partially-ordered local operations, as indicated in Fig. 6.

Figure 6: The spacetime history of a distributed system is generated by a partially ordered network of local interactions.

A dynamical process such as a digital computation or the assembly of a toaster can be thought of as applying a specified sequence of transformations $\tau_1, \tau_2, \ldots$ to a system, starting from a specified initial state x_0:

$$x_0 \xrightarrow{\tau_1} x_1 \xrightarrow{\tau_2} x_2 \xrightarrow{\tau_3} \ldots .$$

Note that this process only needs *one instance* of the system to operate: the system in state x_i disappears into the transformation engine, and re-emerges in state x_{i+1}. If we anticipate needing state x_i at a later time—for instance, to compare it with x_{i+1}—we must make a copy of it ourselves; that is, we must produce another instance of the system and set it to state x_i.

Having reached a state x_n, it is sometimes desirable (for instance, for debugging the program or the assembly process) to *backtrack* from that state, that is, to reconstruct the previous states in the order

$$x_{n-1}, x_{n-2}, x_{n-3}, \ldots$$

until some halting condition is met (or, possibly, all the way back to x_0).

Through what "communication channels" does one part of an extended system system make itself "felt" by the adjacent parts? This kind of question is routinely (though tacitly) dealt with when one models a system by a partial

grapher's equation[10] for a distribution of charges

$$\frac{\partial^2\rho}{\partial t^2} = \frac{\partial^2\rho}{\partial x^2} + a\frac{\partial\rho}{\partial t} \tag{21}$$

we essentially ask the charge density ρ at each place to "sense" the charge densities at certain neighboring places and neighboring instants of time and to "update" itself according to a specific prescription involving the values thus sensed. That this is what we are in fact doing in (21) becomes even more evident when, for the sake of numerical integration, we replace the differential equation by a finite-difference scheme. Note that a number of alternative schemes may be available for this purpose, offering different tradeoffs both in computational terms (rate of convergence, stability, amount of intermediate storage) and in terms of physical realism (is charge represented as granular or continuous? does charge show up in places that are inconsistent with the speed-of-light limit?[11])

In this paper we explore an issue of this nature involving two alternative computational schemes, namely, *cellular automata* and *lattice gases*.

Cellular automata and lattice gases are two well-knowm paradigms for discrete, fine-grained, space- and time-uniform dynamics—in essence, for computation on a *regular mesh* or, in Margolus's words[4], "crystalline computation."

The difference between these two schemes is captured in essence by Fig. 7, which shows side-by-side a three-neighbor cellular automaton and a three-signal lattice gas, both one-dimensional and shown evolving through time. We defer a discussion to §**??**.

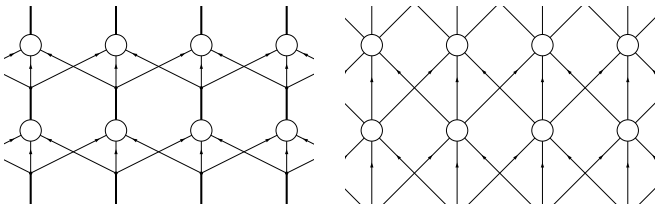


Figure 7: .

Even though there is much in common between the two paradigms, there are also nontrivial differences of substance—and these are of course reinforced by different usage patterns. Overall, cellular automata are preferred for phenomenological models of dissipative systems, where much of the microscopic detail is either poorly known or irrelevant (or both) and the emphasis is on emergent behavior. On the other hand, lattice gases are the models of choice when one is interested in strict accounting of physical resources (e.g., energy, momentum, particle species) or, more generally, strict accounting of *information*. As we shall see in a moment, the latter goes hand-in-hand with an *invertible* dynamics.

[11] It is well known, for example, that if the evolution of a distribution of particles undergoing a random walk is modeled by the "heat equation," one obtains a finite probability for a particle to be found farther than ct from its origin.

of their conservative nature, conceptually belong to the second class—and thus would be expected to be modeled as lattice gases—are instead routinely formulated as cellular automata, albeit of a special kind—namely, *second-order* cellular automata. There are practical as well as historical reasons for that. Nonetheless, there are occasions where it would be desirable, perhaps for convenience of analysis or computational efficiency, to see those system formulated, if at all possible, in terms of lattice gases.

Though a similarity of behavior between certain second-order cellular automata and certain lattice gases had been noticed,[12] no definite correspondence rules between the two classes of systems has yet been proposed.

Of course, since they are computation universal systems, any cellular automaton can be simulated by an appropriate lattice gas and, conversely, any lattice gas can be simulated by a cellular automaton. However, these simulations are in general *non-isomorphic*, in the sense that the simulating system is in some sense "larger," than the simulated one, uses "more machinery," or consumes "expendable resources." In general, to run a faithful simulation one must use a dissipative system (which embodies the informational equivalent of a free-energy source and a heat sink) even if the simulated system is nondissipative. At the very least, one must allow, in addition to the simulated system's state variables, the use of ancillary variables that are "borrowed" as a temporary scratchpad at a certain stage of the computation and then restored to the original state for later reuse—thus "recycled[13] rather than "consumed."

In physical modeling, perhaps the most jealously guarded property is the *invertibility* of the dynamics,[14] In fact, all of the usual conservation laws of physics are but aspects, each one corresponding to a particular set of symmetries of the system under consideration, of the *invertibility* of the underlying dynamics.[15]

List of references

[1] Bennett, Charles H., and Norman Margolus and Tommaso Toffoli, "Bond-energy variables for Ising spin-glass dynamics," *Phys. Rev. B* **37** (1988) 2254

[2] Greenspan, Donald, "Digital studies and perspectives for N-body modelling in physics," *Int. J. Theor. Phys.* **00** (2003), in press.

[3] Jacopini, Giuseppe, and Patrizia Mentrasti and Giovanna Sontacchi, "Reversible Turing machines and poly-

[12] E.g., the Ising spin-glass dynamics [1].

[13] See, for example, [3].

[14] Historically, reversibility appeared in the natural sciences in the context of near-equilibrium thermodynamical transformations. As the connections between thermodynamics and mechanics were clarified it became important to distinguish between this *thermodynamical* reversibility, which always refers to macrostates, and the reversibility of the *mechanical laws*; the latter was then called 'microscopic reversibility'. To avoid confusion, it is now customary to use the wholly unambiguous mathematical term 'invertible' for a 'microscopically reversible' system.

[15] Specified by a group of *symplectic* transformations in classical mechanics and of *unitary* transformations in quantum mechanics.

Disc. Math. **3** (1990), 241–254.

[4] MARGOLUS, Norman, "Crystalline computation," Chapter 18 of *Feynman and Computation* (A. Hey, ed.), Perseus Books (1999).

[5] POMEAU, Yves, "Invariant in cellular automata," *Journal of Physics A* **17** (1984), L415.

[6] FREDKIN, Edward, and Tommaso TOFFOLI, "Conservative logic," *Int. J. Theor. Phys.* **21** (1982), 219–253.

[7] TOFFOLI, Tommaso, and Norman MARGOLUS, *Cellular Automat Machines: A new environemnt for modeling*, MIT Press 1987.

[8] TOFFOLI, Tommaso, and Norman MARGOLUS, "Invertible cellular automata: A review," *Physica D* **45** (1990) 229–253.

[9] TOFFOLI, Tommaso, "Cellular automata mechanics," Ph.D. thesis, The University of Michigan 1976.

[10] VLADIMIROV, V. S., *Equations of Mathematical Physics*, Mir 1984.

[11] VON NEUMANN, John, *Theory of Self-Reproducing Automata*, edited and completed by A. Burks, University of Illinois Press 1966.