On the Collision-Propagation and Gather-Update Formulations of a Cellular Automata Rule

Bastien Chopard[1], Jean-Luc Falcone[1], Ranaivo Razakanirina[1], Alfons Hoekstra[2], and Alfonso Caiazzo[2]

[1] University of Geneva, Switzerland
bastien.chopard@cui.unige.ch
[2] University of Amsterdam, The Netherlands

Abstract. We consider two formulations of a cellular automata: the first one uses a gather-update paradigm and the second one a collision-propagation paradigm. We show the equivalence of both descriptions and, using the latter paradigm, we propose a simple way to define a Cellular Automata on a graph with arbitrary topology. Finally, we exploit the duality of formulation to reconsider the problem of characterizing invertible cellular automata.

1 Introduction

Traditionnally, a 1D, nearest neighbors Cellular Automata (CA) defined on a discrete spatial domain D is expressed as [3,9]

$$s(r,t+1) = F(s(r-1,t), s(r,t), s(r+1,t)) \qquad \forall r \in D \tag{1}$$

where F is the update rule, $s(r,t)$ the state of site $r \in D$ at time $t = 0, 1, 2, \ldots$. The extension to higher dimensions or larger neighborhood is straightforward. Eq. (1) refers to what we call here a *gather-update* (GU) formulation. The values $s(r-1,t)$, $s(r+1,t)$ of the neighbors are first read by cell r (gather operation) and then combined through F to update $s(r)$ at the next time step. This formulation is standard in the CA community.

However, in the community of lattice gas automata (LGA) and lattice Boltzmann (LB) [3] a different formulation is preferred. In LGA or LB, the state $f(r,t)$ of a cell is multi-valued, associating one value with each neighbor. For instance, in 1D, $f(r,t)$ is a three-value column vector

$$f(r,t) = (f_0(r,t), f_1(r,t), f_2(r,t))^T \tag{2}$$

The quantity f_0 is an information only known to the cell itself whereas f_1 and f_2 are data intended to the left and right neighbors.

The update scheme for this formulation is called the *collision-propagation* paradigm. During the propagation phase, the state f_i at location r is moved to the state f_i at location $r+v_i$, i.e. $f_i(r) \to f_i(r+v_i)$. In 1D, with $v_0 = 0$, $v_1 = 1$, $v_2 = -1$ the effect of propagation is to shift the states $i = 1$ to the right and those with $i = 2$ to the left. In general, this streaming of data can be expressed as

$$f^{in}(t+1) = Pf^{out}(t) \tag{3}$$

where P is the propagation operator and it contains the information about the neighborhood topology. Here we consider $f(t)$ as the whole set of states at time t. Thus P takes a full configuration and creates a new one in which each internal state is shifted to the appropriate direction. Note that we have introduced the upperscript *in* and *out* to better distinguish incoming and outgoing information.

After propagation, f can be updated according to the chosen evolution rule. We call this phase the *collision* and we denote by C the operator which transform $f^{in}(r,t+1)$ into $f^{out}(r,t+1)$ for all $r \in D$

$$f_i^{out}(r,t) = C_i\left(f^{in}(r,t)\right) \qquad \text{or, in short} \qquad f^{out}(t) = Cf^{in}(t) \tag{4}$$

By combining eqs. (3) and (4) we get

$$f^{in}(t+1) = PCf^{in}(t) \qquad \text{or} \qquad f^{out}(t+1) = CPf^{out}(t) \tag{5}$$

which we call the collision-propagation (PC) formulation. In what follows, f will denote either f^{in} or f^{out}, depending on the context.

Note that at the boundary of the domain D, some of the f_i may not be properly defined after the propagation step. In our 1D example, f_1 at the left-most r and f_2 at the right-most r will be unknown. Boundary conditions must be then supplied before collision can be applied. We call B the operator acting upon the configuration f and providing the required information, which is obviously problem dependent. Then, eq. (5) becomes $f^{in}(t+1) = PCBf^{in}(t)$.

The above structure is at the heart of the Complex Automata (CxA) approach recently proposed by us [5] for coupling several CA's together. The description of all the CA's in terms of the P, B and C operator provide a generic way to define coupling between different models (see also [1,6]).

2 Cellular Automata on a Graph Topology

CA are usually defined on a regular lattice of cells. This is quite natural if the CA represents a spatial domain. But some problems are more efficiently described by a complex network [2]. This is the case of many applications in economy, social science, epidemiology or system biology.

There is no well established way to define a dynamical evolution of a system whose structure is a complex network or a graph. Clearly, CA have been quite successful to model complex dynamical systems on a regular topology and it is natural to extend the definition of a CA to any set of interconnected cells. We call such an extension a CAG (Cellular Automata on a Graph). Here we use the collision-propagation (CP) paradigm to express it.

Informally a CAG is defined as a triple (V, E, C) where (V, E) is a *directed* graph with V the set of nodes and E the set of edges. We assume that each node r contains internal state variables $f_i^{in}(r)$ and $f_i^{out}(r)$. The number of $f_i^{in}(r)$ is equal to the in-degree k_{in} of node r and the number of $f_j^{out}(r)$ to its out-degree k_{out}. In addition we may have two extra states $f_0^{in}(r)$ and $f_0^{out}(r)$.

The quantity C is the collision operator which acts synchronously and locally at each node $r \in V$ and computes the outgoing values from the incoming ones

$$f_j^{out}(r) = C_j(r) f^{in}(r) \qquad 1 \le j \le k_{out}$$

Note that now the action of C may depend on the node r for the simple reason that different nodes may have a different number of neighbors.

The propagation P as well as the neighborhood are naturally defined from the graph edges in E: assume there is an edge in E from node r_0 to node r_1. In node r_0 we suppose this outgoing edge is labeled with index i_k. In r_1, let us say this incoming edge is labeled i_ℓ. Then, the propagation P will move $f_{i_k}^{out}(r_0, t)$ to $f_{i_\ell}^{in}(r_1, t+1)$.

As before, the C and P operators may be supplemented by an operator B to define boundary conditions. Note however that in a graph, all existing entering edges are expected to come from an existing node (a bit like in a periodic system) and P does not create any missing information. In a CAG, interaction with the external world is then naturally implemented with special nodes having given boundary values and only outgoing edges.

From the above discussion, it is clear that the CP formalism easly describes a CAG with the same compact relation as before, that is $f^{in}(t+1) = P f^{out}(t)$ and $f^{out}(t+1) = C f^{in}(t+1)$.

A software environment has been recently developed to implement a CAG [8]. Its output is illustrated in Fig. 1. The application we have considered is a simple economical model on a complex network. The links of the network represents the possible interactions between idealized agents trading goods against money. Here we assumed that interactions between persons obey a scale-free topology.

In this application, the operator P implements the exchanges of good and money between connected pairs of agents. Based on a local and self-adapting price $p(r, t)$, the operator C computes how an agent's fortune is split among the sellers in its neighborhood in order to buy their goods. In a second phase C also computes how much goods each seller gives to each of its buyer in exchange of its money.

This simple market model and the simulation results are discussed in details in [8]. Here we only want to stress the behavior observed in Fig. 1. Depending on the initial condition and the graph topology, we can see the emergence of submarket, i.e. subgraphs that result from the deletion of the transaction between some pairs of agents. It is indeed observed that two agents r_1 and r_2 that are not in the same *strongly connected component* of the graph will gradually reduce their interaction until the flux of money or goods that traverse the links $r_1 \rightarrow r_2$ or $r_2 \rightarrow r_1$ drops to zero. As time goes on, the dynamics reaches a stationnary state. An interesting result of this model is that the local prices in each emergent submarket converge to a unique value, but a different one in each submarket [8].

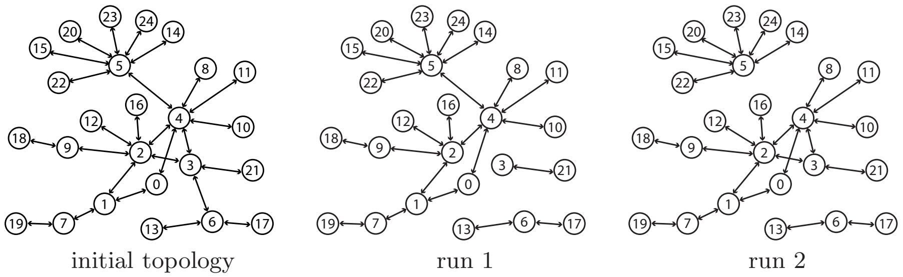

Fig. 1. Left panel: the initial graph topology. Middle and right panels: the resulting emergent submarkets with different initial money distributions.

3 Equivalence between the Gather-Update and Collision-Propagation Formulation

In this section we show that the PC and GU formulations, despite their conceptual differences, are mathematically equivalent.

In GU, all the neighbors of cell r_0 will gather the *same* information from r_0. When the neighbors of r_0 need different data, the CP formulation is more natural. This is the case in the above economical model in which a cell r_0 gives a different amount of money or goods to each of its neighbors. The same happens in LGA/LB models because a different amount of particle flows to the different neighbors. In the PC formulation, the flux of data between neighbors is made explict whereas it is not in the GU model.

Another difference between the two formulations is that the GU combines in one operation the reading of the neighbors state and the update of the site. On the other hand the CP approach clearly disentangles the *local* part of the rule (C) and its *non-local* part (P), thus giving a formulation which is closer to the computer implementation, at least in the case of a parallel code.

The translation from one formulation to the other is straightforward. Let us first assume we have a rule in the CP approach where the state $f(r)$ has, say, $q+1$ components. We write $f(r) = (f_0(r), f_1(r), \ldots, f_q(r))^T$ with $f_i(r)$ the quantity that will be send to neighbor $r + v_i$ ($v_0 = 0$). From $f^{out}(t+1) = CPf^{out}(t)$, we have

$$f_i^{out}(r) = C_i(f_0^{out}(r), f_1^{out}(r - v_1) \ldots f_q^{out}(r - v_q))^T \qquad \forall i \quad 0 \le i \le q$$

In the GU approach, a multi-valued state can be easily defined as $s(r,t) = f^{out}(r,t)$. Since s has $q+1$ components, the evolution rule F must be a set of $q+1$ rules $F_0, \ldots, F_q$ such that

$$s_i(r, t+1) = F_i(s(r,t), \ldots, s(r - v_q, t))$$

By definition of the GU formalism, F_i is a function of the entire state of the neighbor cells. Therefore, in order for F to match the CP formulation, one has to restrict F to one component only of the neighbor's state. By introducing a *selection* operator S_i such that $S_i s = s_i$, we see that the equivalence between a CP rule and a GU rule is obtained by choosing $s(r,t) = f^{out}(r,t)$ and

$$\begin{aligned} F_i(s(r,t),\ldots,s(r-v_q,t)) &= F_i(S_0 s(r), S_1 s(r-v_1), ..., S_q s(r-v_q)) \\ &= C_i(S_0 s(r), S_1 s(r-v_1), ..., S_q s(r-v_q)) \end{aligned} \tag{6}$$

As an example, let us consider the simple 1D CA in which C is the identity. By applying CP, the states f_1 move east and the states f_2 move west. In accordance to eq. (6) this rule can also be written as

$$s_1(r,t+1) = s_1(r-1,t) \qquad s_2(r,t+1) = s_2(r+1,t) \tag{7}$$

Thus the identity rule in CP implements simultaneously two rules in the GU formulation, namely the so-called *east* and *west* rules.

We can now consider the inverse problem, i.e. how to write a GU rule in the CP form. Let us assume we have

$$s(r,t+1) = F(s(r,t), s(r-v_1,t)\ldots, s(r-v_q,t))$$

where s and F are possibly multi-component quantities. To obtain a CP version, the first step is to replicate s for all $q+1$ neighbor directions. We thus introduce an *expansion* operator E such that $f^{out}(r) = Es(r)$ means $f_i^{out}(r) = s(r)$, for $0 \le i \le q$. We can also define the inverse of E, the projection operator Π such that $\Pi f^{out}(r) = f_0^{out}(r) = s(r)$. The propagation can then be applied to $f^{out} = Es$. To match the behavior of F in the GU formulation, one has to choose $C_i = F$, for all $0 \le i \le q$. It is indeed easy to check that

$$Es(t+1) = f(t+1) \qquad \text{or} \qquad \Pi CPEs(t) = s(t+1)$$

To illustrate this construction we take the *east* CA rule $s(r,t+1) = s(r-1,t)$ which simply moves information to east. The rule is $F(s(r), s(r-1), s(r+1)) = s(r-1)$. On a spatial configuration $\ldots abc\ldots$ it acts as follows

$$\ldots abc\ldots \xrightarrow{F} \ldots zab\ldots$$

where z is the state of the cell on the left. In the CP case, with the collision $C_0(f_0,f_1,f_2) = C_1(f_0,f_1,f_2) = C_2(f_0,f_1,f_2) = f_1$ we have for the vector $f = Es$

$$\ldots \begin{pmatrix} a\\a\\a \end{pmatrix} \begin{pmatrix} b\\b\\b \end{pmatrix} \begin{pmatrix} c\\c\\c \end{pmatrix} \xrightarrow{P} \ldots \begin{pmatrix} a\\z\\b \end{pmatrix} \begin{pmatrix} b\\a\\c \end{pmatrix} \begin{pmatrix} c\\b\\d \end{pmatrix} \ldots \xrightarrow{C} \ldots \begin{pmatrix} z\\z\\z \end{pmatrix} \begin{pmatrix} a\\a\\a \end{pmatrix} \begin{pmatrix} b\\b\\b \end{pmatrix} \ldots$$

which is, as expected, $Es(t+1)$.

4 Invertible CA

The question of finding the inverse of a CA rule has been discussed in several papers [4], but mostly in the framework of a GU formulation. In the CP model, there is an easy way to build an invertible rule, using the physical interpretation of a collision process followed by particle motion [3,7]: assume we have $f(n) = (PC)^n f(0)$ for some given C operator. After n iterations let us apply a still to be specified local transformation A to the configuration $f(n)$ and apply, for n steps, another collision operator C'. A rule is said *invertible* if we then obtain back the initial configuration $f(0)$, up to the transformation A. This condition reads $(PC')^n A(PC)^n f(0) = Af(0)$ or, simply, $(PC')^n A(PC)^n = A$ since it is true for any $f(0)$. With

$$(PC')^n A(PC)^n = (PC')^{n-1} PC'APC(PC)^{n-1}$$

we see, by induction over n that condition $(PC')^n A(PC)^n = A$ is equivalent to

$$A = PC'APC \tag{8}$$

In other words, if we can find A and C' such that the above condition holds, the CA is invertible.

It is easy to check that a way to solve (8) is to impose

$$A = R^{-1}C \qquad PRP = R^{-1} \qquad C'R^{-1}C = R \tag{9}$$

for some *reverse* operator R having an inverse R^{-1}. This solution is inspired from LGA models of particles [3,9]: to reverse the particles motion, one needs to perform an extra collision, reverse their velocity (R does it) and then run the LGA n times to finally obtain the initial state with reversed velocities.

It is interesting to note that eq. (9) includes Fredkins method [9] to produce a reversible rule in the GU approach

$$s(r, t+1) = F(\{s(r - v_i, t)\}_{i=0}^{q}) \oplus s(r, t-1) \tag{10}$$

where $\oplus$ denotes the logical xor and s is a Boolean state. The rule is reversible (self-inverse) for any F since it can be written as $s(r, t-1) = F(\{s(r-v_i, t)\}_{i=0}^{q}) \oplus s(r, t+1)$ due to the property of the xor.

Let us now transform (10) in a CP form. To keep the notation simple we consider a 1D case. Two states per cell must be stored: the current and previous time step. Let us call them s and $\bar{s}$. According to the procedure above, we construct

$$f(r, t) = E(s, \bar{s})^T = (s(r, t), \bar{s}(r, t), s(r, t), \bar{s}(r, t), s(r, t), \bar{s}(r, t))^T$$

where, by choice, components 3,4 are propagated to the right, components 5,6 to the left and components 1,2 are the rest states. Let us define the collision C as follows

$$\begin{aligned} C_i f &= F(f_1, f_3, f_5) \oplus f_{i+1} \qquad i = 1, 3, 5 \\ C_2 f &= f_1 \qquad C_4 f = f_5 \qquad C_6 f = f_3 \end{aligned} \tag{11}$$

Let us define R as the operator which, first, swaps the values which propagate in one direction with those propagating in the other direction and, second, swaps

f_i and f_{i+1} for $i = 1, 3, 5$. R can be expressed by a 6×6 matrix

$$R = \begin{pmatrix} B & 0 & 0 \\ 0 & 0 & B \\ 0 & B & 0 \end{pmatrix} \qquad \text{with} \qquad B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

It is easily verified that $B^2 = 1$ and $R = R^{-1}$. Let us now check whether condition (9) holds. We have

$$\begin{aligned} PRPf(r) &= PR(f_1(r), f_2(r), f_3(r-1), f_4(r-1), f_5(r+1), f_6(r+1))^T \\ &= P(f_2(r), f_1(r), f_6(r+1), f_5(r+1), f_4(r-1), f_3(r-1))^T \qquad (12) \\ &= (f_2(r), f_1(r), f_6(r+1-1), f_5(r+1-1), f_4(r-1+1), f_3(r-1+1))^T \\ &= Rf(r) \end{aligned}$$

Thus $PRP = R$. For the collision condition $C'R^{-1}C = R$ we choose $C' = C$ since the rule is expected to be its own inverse. By dropping the cell location r and writing F for $F(f_1, f_3, f_5)$ we have

$$\begin{aligned} CRCf &= CR(F \oplus f_2, f_1, F \oplus f_4, f_5, F \oplus f_6, f_3)^T \\ &= C(f_1, F \oplus f_2, f_3, F \oplus f_6, f_5, F \oplus f_4)^T \qquad (13) \\ &= (F \oplus F \oplus f_2, f_1, F \oplus F \oplus f_6, f_5, F \oplus F \oplus f_4, f_3)^T \\ &= (f_2, f_1, f_6, f_5, f_4, f_3)^T = Rf \end{aligned}$$

Thus $CRC = R$ and Fredkins method is a special case of eq. (9).

It should be noted that another way to solve eq. (8) is

$$A = PR \qquad PRP = R^{-1} \qquad C'R^{-1}C = R \qquad (14)$$

If we choose $R = P^{-1}$ (P^{-1} always exists since it corresponds to moving the information backwards) then (14) requires $C'PCP = 1$. Consider a 1D rule $s(r, t+1) = F(s(r-1, t), s(r, t), s(r+1, t))$ and its potential inverse $s(r, t+1) = G(s(r-1, t), s(r, t), s(r+1, t))$. We set $f(r) = (f_1, f_0, f_2) = Es(r) = (s(r), s(r), s(r))$ and we associate C to F and C' to G. Thus

$$C'PCPf = C'P \begin{pmatrix} F(s(r-1), s(r), s(r+1)) \\ F(s(r-1), s(r), s(r+1)) \\ F(s(r-1), s(r), s(r+1)) \end{pmatrix} = C' \begin{pmatrix} F(s(r-2), s(r-1), s(r)) \\ F(s(r-1), s(r), s(r+1)) \\ F(s(r), s(r+1), s(r+2)) \end{pmatrix}$$

Then $C'PCP = 1$ if

$$G[F(s(r-2), s(r-1), s(r)), F(s(r-1), s(r), s(r+1)), F(s(r), s(r+1), s(r+2))] = s(r) \qquad (15)$$

which is the expected *non-local* relation expressing that rules F and G are inverse of each other. It is clear that rule *east* $F(s(r-1), s(r), s(r+1)) = s(r-1)$ and rule *west* $G(s(r-1), s(r), s(r+1)) = s(r+1)$ obey this inverse relation.

In general, the validity of eq. (15) requires to check all 2^5 values of $s(r-2)$, $s(r-1)$, $s(r)$, $s(r+1)$, $s(r+2)$. In 2D and a von Neumann neighborhood $2^{13} = 8192$ tests would be necessary to check whether $C'PCP = 1$. In contrast, the verification of the *local* condition $C'R^{-1}C = R$ requires only 3×2^3 operations in 1D and $5 \times 2^5 = 160$ in 2D.

5 Conclusions

We have formally discussed the relation between the collison-propagation (CP) and the gather-update (GU) formulation of a CA rule. The CP formulation is naturally adapted to situations where the flow of information depends on the neighbors. It is also well suited to couple several CA [5].

Furthermore we have introduced the concept of CAG (cellular automata on a graph) by applying the CP approach to irregular topologies and asymmetrical neighborhoods.

We have finally explored the conditions of finding the inverse of a CA rule using both formulation. We found that two classes of invertible CA can be identified: the information to inverse the rule is local (as in the Fredkins construction or in discrete fluid models); or the information to inverse the rule is non-local (shared by the neighbors) and the problem is numerically more intensive.

This work is supported by the European Commission (COAST project EU-FP6-IST-FET Contract 033664) and the Swiss National Science Foundation.

References

1. Caiazzo, A., Falcone, J.-L., Chopard, B., Hoekstra, A.: Error investigations in complex automata models for reaction-diffusion systems. In: Umeo, H., et al. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 260–267. Springer, Heidelberg (2008)
2. Caldarelli, G.: Scale-Free Networks. Oxford University Press, Oxford (2007)
3. Chopard, B., Droz, M.: Cellular Automata Modeling of Physical Systems. Cambridge University Press, Cambridge (1998)
4. Durand-Lose, J.: Representing reversible cellular automata with reversible block cellular automata. In: Discrete Mathematics and Theoretical Computer Sciences Proceedings AA (DM-CCG), pp. 145–154 (2001)
5. Hoekstra, A., Lorenz, E., Falcone, J.-L., Chopard, B.: Towards a complex automata formalism for multuscale modeling. Int. J. Multiscale Computational Engineering 5(6), 491–502 (2008)
6. Hoekstra, A.G., Falcone, J.-L., Caiazzo, A., Chopard, B.: Multi-scale modeling with cellular automata: The complex automata approach. In: Umeo, H., et al. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 192–199. Springer, Heidelberg (2008)
7. Marconi, S., Chopard, B.: Discrete physics, cellular automata and cryptography. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 617–626. Springer, Heidelberg (2006)
8. Razakanirina, R.M.: Cellular automata on graphs applied to financial flow simulation. Master's thesis, University of Geneva, Computer Science department, Master thesis (2007)
9. Toffoli, T., Margolus, N.: Cellular Automata Machines: a New Environment for Modeling. MIT Press, Cambridge (1987)