Recursivity in Field-Based Programming: the Firing Squad Example

Luidnel Maignan
LIAFA, Université Paris-Diderot
France

Jean-Baptiste Yunès
LIAFA, Université Paris-Diderot
France

***Abstract*—In cellular automata, the well-known firing squad synchronization problems have many solutions usually provided as explicit transition tables, and explained in terms of idealized continuous signals and their collision. However, very few proofs exist despite of the large amount of work on these problems. In this presentation, we take the spatial computing point of view and provide a field-based description of a solution. On the cellular automata part, this provide a understandable and formal construction of a very general solution from which a proof seems to be derivable almost directly. On the spatial computing part, this provides an example of recursive field functional, with a kind of tail-recursivity leading to a strictly finite system.**

I. Introduction

A. Firing squad and signal-based programming

In cellular automata, the firing squad synchronization problem (FSSP) [2], [12], [13] may be stated as follows: *find a finite transition function having a given (fire) state such that, starting from arbitrary sized line of cells where all but one cell (the general) are quiescent, all cells enter for the first time in this state synchronously*. A classical solution is to send signals at different speed so that they first collide at the middles of the space, then at the quarter of the space, then at the eighth of the space, and so on, until an accumulation point is reached. For example, if one sends two bouncing signals from the leftmost cell, one at speed 1 and another at speed $\frac{1}{3}$, these two signals will collide at the middle of the space. However, this is only an idealized presentation since actual solutions have to deal with peculiarities appearing when applying these continuous concepts to the discrete cellular space, the parity of the space being the simplest example. Solutions based on these type on intuition are therefore obtained by solving these peculiarities by hand by iterative correction of the transition table.

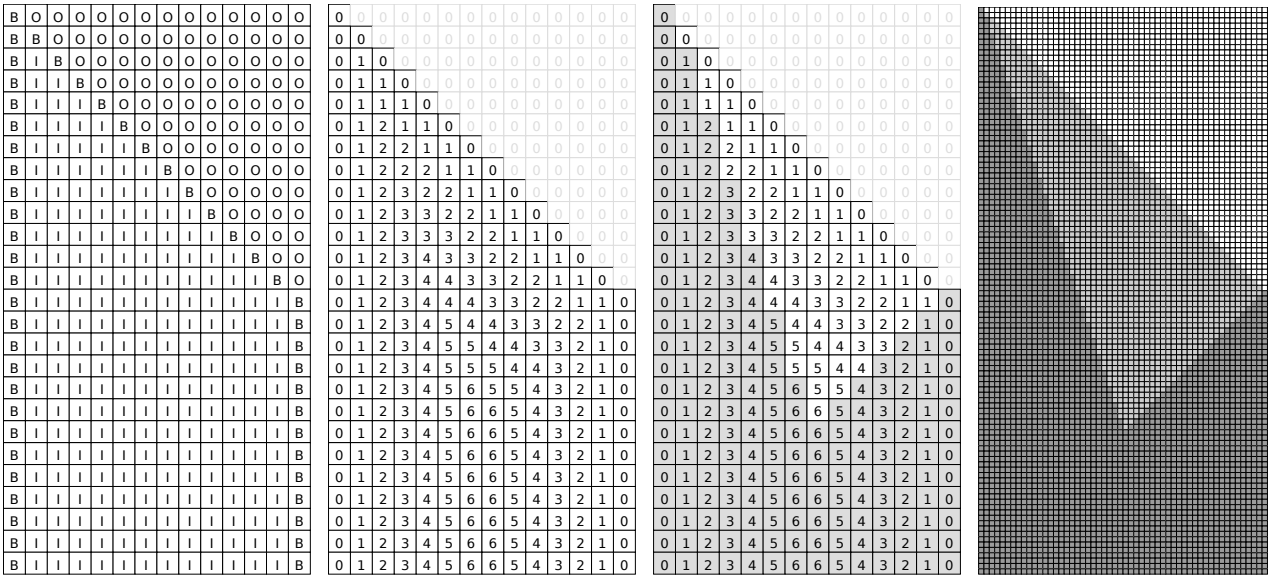
Despite these difficulties, this basic idea has been generalized into many solutions for the classical problems and for some generalizations. One can consider synchronizing with general at any arbitrary position [1], [18], [20], many generals synchronous or not [17], synchronizing 2D-spaces [3], [6], [17], 3D-spaces [16], graphs [5], [15], and variants with different constraints on shape of the space, that may also be dynamic to some extent [4]. However, the drawback of the method is that proofs are hard to obtain directly from the solution, and mistakes has also been found in some cases, using large experiment on many initial configuration. Only a very poor number of proofs of correctness [11], [14], [19] exist.

B. Field-based approaches in spatial computing

Spatial computing considers massively distributed architectures as (programmable) spaces and promote the use of spatial concepts to ease the programming of such architectures. Data structure are therefore spatially extended objects, as can be seen in languages such as PROTO and MGS. In particular, the concept of *fields* is an important one: it is an object which associates a value to each point in space and specifies the local evolution of these values in time. In contrast with cellular automata, the values evolution is not a closed system but an open one, i.e. it may depend on values determined by other input fields. Fields can therefore be composed together to form more complex fields or fully determined (closed) systems, as functions can be composed to obtain more complex functions or programs.

In [7]–[10], the concept of fields has been applied to cellular automata in order to solve algorithmically different dynamic geometric problems in a modular way, using a common set of fields. In particular, the distance field is present in all of them and is the primary building block to collect spatial information in a finite-state manner. While all these problems are geometrical, it was postulated that non-geometrical problems could also be tackled with the same building blocks, and this paper is here to provide such an example. Also, while fields are really manipulated as functions, no cases analogous to recursive functions arose naturally from previously considered problems, this paper is again here to provide such an example, along with a notion of tail-recursivity.

Indeed, we propose to apply the same methodology to provide an algorithmic description of the FSSP. By algorithmic, we mean decomposing the problem into easier sub-problems, solving each sub-problem by a field, and composing the fields together to obtain a cellular automata solving the problem. In this process, each sub-problem and field will have a simple and fully independent semantic, contrary to signals whose meanings depend on all the signals present in the system. Ultimately, we show that we recover the classical concepts of modularity, reusability, semantic decomposition, etc in the context of cellular automata. Benefits of this approach are directly observable by the generality of the provided solution, and by how easier and intelligible a proof for this system seems to be compared to previous solutions.



(a) Region field (b) Distance field (c) Fixity field (d) Size 54

Fig. 1. The three fields describing the initial region and its middle

II. A Field-Based Description of the FSSP

A. An algorithmic decomposition

We recall that the main idea is to divide the space into two equals regions and to proceed recursively until all regions have size 1.

Naturally, the first step is to *identify the middle of the physical space*. To do this we introduce three fields. The first field, R^0, represents the discovery of the region to be cut. The second field, D^0, will provide some distance information deduced from R^0, such that this distance will eventually allows us to detect the middle. The third one is a boolean field, F^0 that indicates the correctness of the values of R^0 and D^0. Indeed, R^0 and D^0 are dynamic fields, which means that R^0 discovers the space from time to time and so its derived field D^0 also updates accordingly. Eventually R^0 stabilizes when it corresponds to the whole physical space and leading in turn to the stabilization of D^0 (see Section II-B).

Now that we have a region and its middle, we introduce another collection of three fields: R^1 that represents the discovery of the two regions induced by the previous cut of the space, D^1 the distance field deduced from R^1 such that the middles of the two regions will be detected, and F^1 the corresponding correctness field. As the reader might guess, this extends to a recursive schema which defines R^ℓ, D^ℓ and F^ℓ in terms of $R^{\ell-1}$, $D^{\ell-1}$ and $F^{\ell-1}$ (see Section II-C).

This obviously implies that we need an unbounded number of fields. However, we will later explain how this can be reduced to a finite system (see Section II-D).

B. Initial region and its middle

The *initial region field* R^0 is defined using three states O ("*outside*"), B ("*border*") and I ("*inside*"). O is the quiescent state of the field. A cell in state O will turn into B as soon as one of its neighbors is in state B. The "border" state is used to mark the border of the region currently discovered, which at this step must finally correspond the whole physical space. A cell in state B which does not coincide with a physical border of the space updates its state to I. With the help of a given static boolean field $Border^0(x)$ that states for each x if it is a physical border or not, the field R^0 is formally defined by:

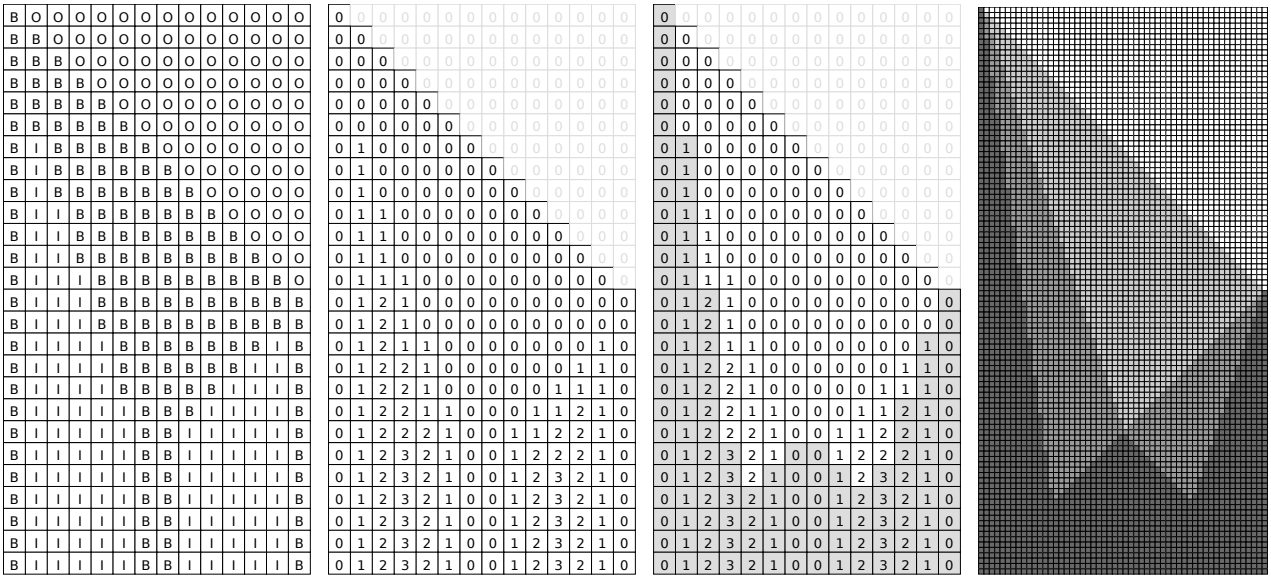
$$R^0_t(x) = \begin{cases} B & \text{if } R^0_{t-1}(x) = O \wedge \\ & \quad \exists y \in N(x); R^0_{t-1}(y) = B \\ I & \text{if } R^0_{t-1}(x) = B \wedge \neg Border^0(x) \\ R^0_{t-1}(x) & \text{otherwise.} \end{cases} \tag{1}$$

From any initial condition of the form $BO\dots O$, the evolution of R^0 produces a space-time diagram like the one depicted in Fig. 1(a).

Now that we have the field that, after some time, represents the whole initial region, we want to determine its middle point. To do so, we use the fact that the middle of a region is the further inner point from both "borders". So, we build the *distance field* D^0, which associates to each cell its distance to latest observed nearest "borders" of the region. As the middle is necessarily an inner cell, the value of D^0 of any cell that is not inside is defined as 0. One can note that this is coherent with the fact that a "border" is obviously at distance 0 from a "border". For an inner cell its distance to the latest observed nearest "border" is obviously 1 plus the smallest distance to the latest observed nearest "border" of its neighbors. This is formally defined by:

$$D^0_t(x) = \begin{cases} 0 & \text{if } R^0_t(x) \neq I \\ \min_{y \in N(x)} 1 + D^0_{t-1}(y) & \text{otherwise} \end{cases} \tag{2}$$

This rule has been extensively studied as a generic building



(a) Region field (b) Distance field (c) Fixity field (d) Size 54, level 0 and 1

Fig. 2. The three fields describing the level 1

block in cellular automata that solve different geometric problems (see [7]–[10]). It is sufficient to detect non-strict local maxima from the distance field to obtain the middle cell(s) of the region. This is illustrated in Fig. 1(b) which shows how region and distance fields evolve.

A final piece is required with respect to synchronization: we need to know when values provided by the region and distance fields are finals. Hence, we define a boolean field F^0 which associates to each cell a boolean indicating if its respective region and distance values are definitely correct. It is possible to determine the appropriate value by a simple case analysis. First, no field's value of an "outside" cell is considered correct (it has not been discovered yet). "Border" cell field's values are correct only if they coincide with a physical border. For "inside" cells, we know by construction that they are really inside so this value is always correct, but we still need to ensure that the distance value is also correct. To determine the distance value correctness, we use the fact that the region only grows, which implies that distance values only increase. And, from the point of view of a cell x, this means that once a neighbor is both correct and minimaly-valued in its neighborhood, it will remain such forever. This ensures that the distance value of x will not evolve anymore since it correspond to this fixed value + 1 as specified in Eq. (2). Altogether, this leads to the following formal definition, whose evolution is illustrated in Fig. 1(c).

$$F_t^0(x) = \bigvee \begin{cases} R_t^0(x) = B \wedge Border^0(x) \\ R_t^0(x) = I \wedge \exists y \in N(x); \\ \quad D_t^0(x) = 1 + D_{t-1}^0(y) \ \wedge \ F_{t-1}^0(y) \end{cases} \tag{3}$$

C. Subsequent regions and middles

Now that the initial region is identified and that enough information has been built to divide it, let us proceed by adding new fields to obtain the division and provide sufficient information to recurse.

First, let us clearly identify what we want to build. From Fig. 1, it should be clear that we are going to build one region starting from the left and another starting from the right. However, we shall prevent ourselves to trust our eyes too much, but try to describe what we want by definition.

Let us come back on what we have done for the initial region and do nearly the same here. Given the predicate $Border^0(x)$, what we built is a region field whose values are, after some time, $R^0(x) = B$ for x's that are physical borders, and $R^0(x) = I$ for x's that are not physical borders and so inner cells.

In the region field R^1, we want to obtain as borders all the "borders" obtained at the previous level and new ones corresponding to the middle(s) cell(s) finally obtained at the previous level. Thus, we consider as borders of the two regions all correct x's such that $R^0(x) = B$, and all x's that correspond to correct non-strict local maxima of D^0. We also want to have $R^1(x) = I$ everywhere x is correct and is neither a "border" nor a maximum among its neighbors in D^0. This naturally leads to the following recursive formal definition of the two predicates $Border$ and $Inside$ for any level $l > 0$:

$$Border_t^{\ell+1}(x) = \bigvee \begin{cases} R_t^\ell(x) = B \wedge F_t^\ell(x) \\ \forall y \in \{x\} \cup N(x); \\ \quad D_{t-1}^\ell(x) \geq D_{t-1}^\ell(y) \ \wedge \ F_{t-1}^\ell(y) \end{cases} \tag{4}$$

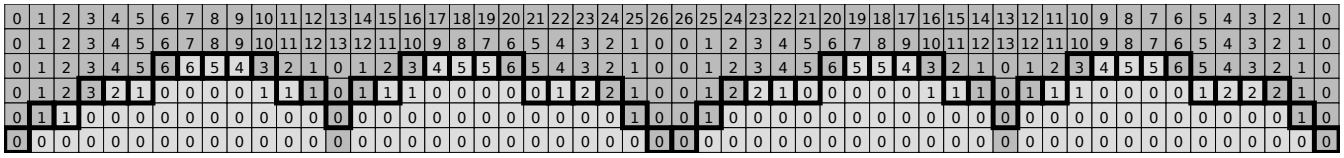


Fig. 3. Stack (level 0 on top) of all field values computed at time 96. Line length is 54.

$$Inside_t^{\ell+1}(x) = F_t^\ell(x) \wedge R_t^\ell(x) \neq B \\ \wedge \exists y \in N(x);\ D_{t-1}^\ell(y) > D_t^\ell(x) \quad (5)$$

Given these two boolean fields, we can apply the same reasoning as before and, obtain nearly the same evolution rule as the initial region. We only need to change the use of $\neg Border^0(x)$ in Eq. 1 into $Inside_t^1(x)$ and the use of $Border^0(x)$ in Eq. 3 into $Border_t^1(x)$. Thus, we obtain the three additional fields describing the first level of division, and iterating this construction, for any level $l > 0$ we obtain the following recursive definition:

$$R_t^\ell(x) = \begin{cases} B & \text{if } R_{t-1}^\ell(x) = O \wedge \\ & \quad \exists y \in N(x); R_{t-1}^\ell(y) = B \\ I & \text{if } R_{t-1}^\ell(x) = B \wedge Inside_t^\ell(x) \\ R_{t-1}^\ell(x) & \text{otherwise.} \end{cases} \quad (6)$$

$$D_t^\ell(x) = \begin{cases} 0 & \text{if } R_t^\ell(x) \neq I \\ \min_{y \in N(x)} 1 + D_{t-1}^\ell(y) & \text{otherwise.} \end{cases} \quad (7)$$

$$F_t^\ell(x) = \bigvee \begin{cases} R_t^\ell(x) = B \wedge Border_t^\ell(x) \\ R_t^\ell(x) = I \wedge \exists y \in N(x); \\ \quad D_t^\ell(x) = 1 + D_{t-1}^\ell(y) \ \wedge \ F_{t-1}^\ell(y) \end{cases} \quad (8)$$

Fig. 2 shows how the three fields evolve at level 1 of the algorithm. In Fig. 2(a) we have one region that grows from the left and starts at the initial time, and another one that grows from the right and starts at time $n-1$ (n is the number of cells). The distance field D^1 evolves inside each region described by R^1.

One can observe that while in D^0 the non-strict local maxima spanned two cells, then in D^1 there is two non-strict local maxima that both span only one cell. This depends on whether the region's length is odd or even (Fig. 2(b)).

D. Reduction to a finite number of states

Now we face two problems. The first one is that distance fields are defined over integers and the other one that we obtained an unbounded number of fields. A detailed explanation of the reducability in finite state is out of the scope of this paper, but let us sketch the most important steps.

The first problem can be solved using a special property. If an integer field is Lipschitz-continuous, i.e. the difference of values between two neighbors is bounded, and the information used in the system only depends on this difference, then it can be transformed into a finite-state field (refer to [8] for all the details). An application of this result is that when the difference is at most 1, then only 3 states are required. With definitions given in Eq. 2 and Eq. 7, it's easy to remark that all the distance fields D^ℓ can therefore be represented with only 3 states each.

To solve the second problem we remark that *in some sense* the recursive schema is "tail-recursive". Indeed, tail-recursiveness is about conserving only the information that are required by the subsequent recursive calls. From the point of view of a cell x, if its field values at given level ℓ are correct, this means that they do not evolve anymore. If furthermore its field values at $\ell + 1$ are also correct and so are the values of its neighbors, then its values at level ℓ are no more useful and can be discarded. This is observable in Fig. 3 where fields values are represented for all cells at a given time. Values (x, ℓ) in darker gray are correct ($F_t^\ell(x)$ is true), and if the whole neighborhood at the next level is also gray, then (x, ℓ) can be "forgotten". By discarding all these gray values (and a little bit more with a much finer analysis), we obtain for each cell a *lowest useful level* represented by a bold surround in the figure. In fact, these are the only necessary values that need to be stored, along with their associated lowest level number (which can be represented with only three state thanks to the Lipschitz-continuous argument). Altogether, this shows that field values are uniformly bounded, and that only a finite number of fields is required. This imply that we finally describe the behavior of a cellular automaton.

III. Conclusion

Without any modification, the system described in this paper is much more general than one can think. Indeed, in all our description we never use the property that there is only one general on the left. Thus we can naturally expect that it is agnostic to such particularities, and this is exactly the case as one can observe in Fig. 4. We also never assumed that the wake-up of the cells happens one after the other from the general, so that removing the corresponding sub-system, one obtain a solution for arbitrary initial desynchronized configuration.

It seems also possible to compose the same fields in slightly different ways to obtain different kind of solutions or to extend this solution to higher dimensions. We can also expect that a proof of correctness of the solution for all sizes and all initial desynchronized configurations to be much more easier than for classical solutions, each field is simple and almost correct by construction, and so is their composition.

References

[1] Karel Culik. Variations of the firing squad problem and applications. *Information Processing Letters*, 30:153–157, 1989.

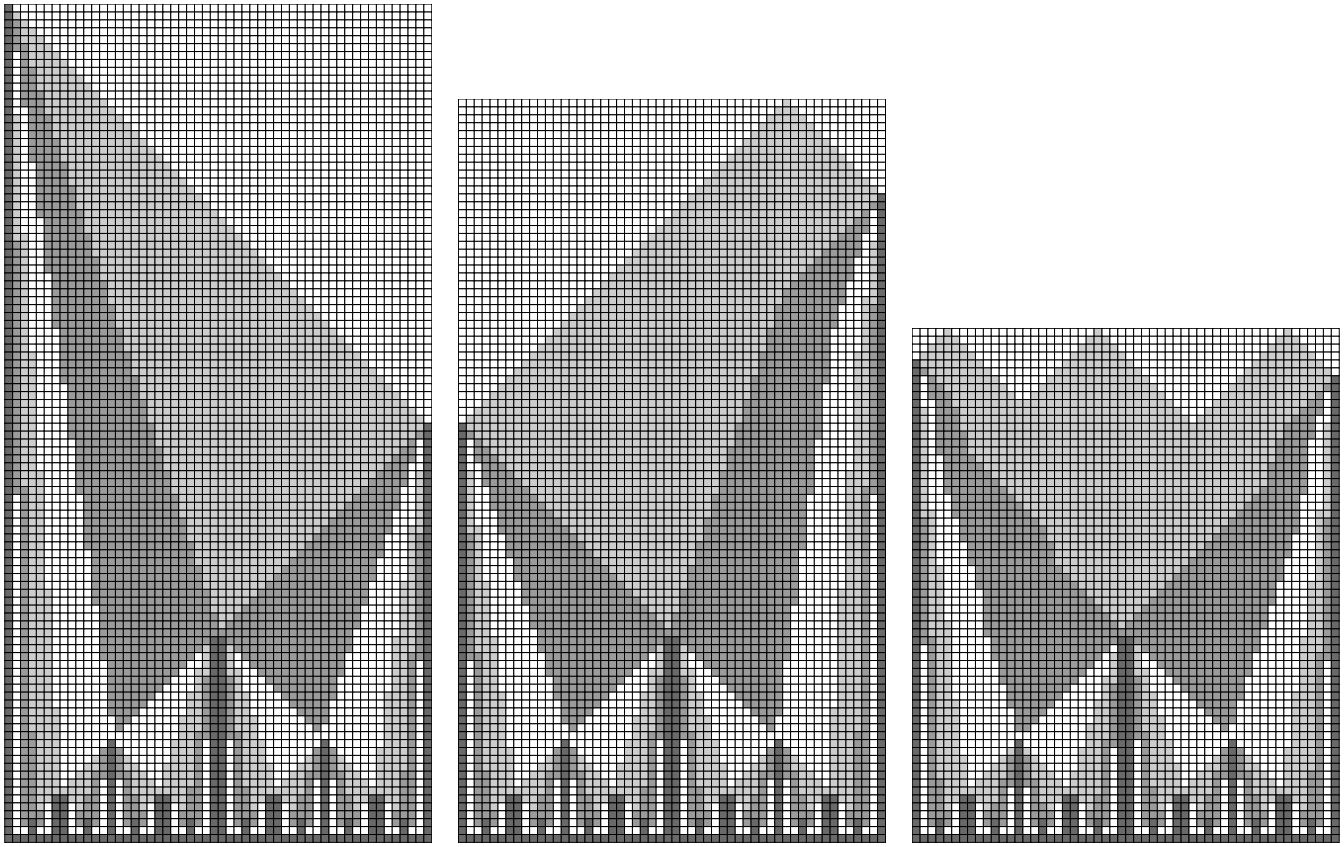


Fig. 4. Evolution of the complete system with different set of generals

[2] Eiichi Goto. A minimum time solution of the firing squad synchronization problem. Courses Notes for Applied Mathematics 298, Harvard University, 1962.

[3] Antonio Grasselli. Synchronization of cellular arrays: The firing squad problem in two dimensions. *Information and Control*, 28:113–124, 1975.

[4] G.T. Herman, W. Liu, S. Rowland, and A. Walker. Synchronization of growing cellular automata. *Information and Control*, 25:103–122, 1974.

[5] Tao Jiang. The synchronization of nonuniform networks of finite automata. *Information and Control*, 97:234–261, 1992.

[6] Kojiro Kobayashi. The firing squad synchronisation problem for two-dimensional arrays. *Information and Control*, 34:177–197, 1977.

[7] Luidnel Maignan and Frédéric Gruau. Integer gradient for cellular automata: Principle and examples. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 321–325, Washington, DC, USA, 2008. IEEE Computer Society.

[8] Luidnel Maignan and Frédéric Gruau. A 1D cellular automaton that moves particles until regular spatial placement. *Parallel Processing Letters*, 19(2):315–331, June 2009.

[9] Luidnel Maignan and Frédéric Gruau. Convex hulls on cellular automata. In Stefania Bandini, Sara Manzoni, Hiroshi Umeo, and Giuseppe Vizzari, editors, *ACRI*, volume 6350 of *Lecture Notes in Computer Science*, pages 69–78. Springer, 2010.

[10] Luidnel Maignan and Frédéric Gruau. Gabriel graphs in arbitrary metric space and their cellular automaton for many grids. *ACM Trans. Auton. Adapt. Syst.*, 6:12:1–12:14, June 2011.

[11] Jacques Mazoyer. A six states minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50:183–238, 1987.

[12] Marvin Minsky. *Computation : Finite and Infinite Machines*. Prentice-Hall, 1967.

[13] Edward E. Moore. *Sequential machines, Selected papers*. Addison Wesley, 1964.

[14] Kenichiro Noguchi. Simple 8-state minimal time solution to the firing squad synchronization problem. *TCS*, 314:303–334, 2004.

[15] P. Rosenstiehl, J.R. Fiskel, and A. Holliger. *Intelligent Graphs : Networks of Finite Automata capable of Solving Graph Problems*. Graph Theory and Computing (R.C. Read Ed.) Academic Press, 1972.

[16] Ilka Shinahr. Two and three dimensional firing squad synchronization problems. *Information and Control*, 24:163–180, 1974.

[17] Helge Szwerinski. Time-optimal solution of the firing-squad synchronization problem for n-dimensional rectangles with the general at an arbitrary position. *Theoretical Computer Science*, 19:305–320, 1982.

[18] V.I. Varshavsky, V.B. Marakhovsky, and V.A. Peshansky. Synchronization of interacting automata. *Mathematical System Theory*, 4 n.3:212–230, 1969.

[19] Jean-Baptiste Yunès. An intrinsically non minimal-time Minsky-like 6-states solution to the firing squad synchronization problem. *RAIRO ITA/TIA*, 42(1):55–66, 2008.

[20] Jean-Baptiste Yunès. Known CA synchronizers made insensitive to the initial state of the initiator. *JCA*, 4(2):147–158, 2009.