An Introduction to CELLULAR AUTOMATA

Hélène Vivien

Acknowledgment

*

It is a pleasure to thank Serge Grigorieff for all his aid. It was he who first suggested to me the study of cellular automata and who then supplied me with a steadily increasing bibliography on the subject. Chapters 1 to 13 are the result of my efforts to understand and put some order into this growing heap. He read several preliminary drafts, forever suggesting complements and improvements, making sure that certain precious results by workers in the field were not forgotten. Without his constant help, his advice and the numerous discussions I had with him, this book would never have been written. I am happy to express my deep appreciation for all his help and encouragement.

Introduction

In common language, an automaton is a self propelled object, a mechanical puppet, performing jerky movements from one position to another, head turned left, arm up, ..., as Olympia in Offenbach's opera "The tales of Hoffman". Mathematically, it is only an abstract object, characterized by its state at any instant and (possibly) changing state at different points of time. We are thus given a set of states and a set of instants which will be the set of non-negative integers. The passage from one state to the next is governed by the internal law of the object.

A "cellular automaton" is an automaton which is formed of a large number of cells, identical small automata, evolving in a synchronized manner and interdependent : the state of a cell at a given time depends on the state of its neighbouring cells at the previous instant. The state of the automaton is the juxtaposition of the states of its component cells. The macroscopic evolution is thus the result of innumerable microscopic movements. Even if the basic component is very simple, with very few states, if the architecture of the whole is simple and regular, and the rules that govern local interactions are simple also, they result in an extremely complex and intricate global behavior.

The first one to study such systems is Stanislaw Ulam (1909-1984) who, after World War II, uses the very first computers to simulate examples of cellular automata [55]. His cells are the squares of an a priori infinite plane chessboard, and have as neighbours the four adjoining squares or the eight surrounding ones. The observation of the overall patterns formed by the states was a game, but this game suggested the idea of some universe in reduction, complex, having its own laws, and which could serve as a model for our real world.

His colleague at Los Alamos, John Von Neumann (1903-1957) is captivated by the human brain and the mechanisms of life, one of whose chief characteristics is its capacity to reproduce itself. At that time, in an industrial world, machines capable of fabricating less sophisticated machines were common. But Von Neumann tries to conceive one which can reproduce itself, a sort of assembly system, capable of reaching out for necessary parts. The technical difficulties, however, seem insuperable.

Ulam suggests he should avoid the impediments of the material world and take cellular automata as models for achieving his aim. An idea which seems quite natural nowadays, when we think that living organisms are made of cells. Thus Von Neumann pursues his search of an autoreproductive machine within Ulam's framework. In so doing, he tackles all the problems which will be at the origin of the future research in cellular automata.

In his idea the machine must comprise different parts, a "universal constructor", capable of constructing any machine from a description, then the "description" of the machine itself. There he comes across a logical problem (about the place of the description in the machine), whose solution he finds in

the works of Alan Turing (1912-1954) : with the supervision of a third part, a "universal Turing machine", the process of reproduction is decomposed into two successive phases, first the construction, then the duplication of the description. At last, in 1952, Von Neumann succeeds in building his machine [59]. It is a plane cellular automaton, the neighbours of each cell are the four adjoigning squares, it has no less than 200,000 cells and 29 possible states (among which the "empty" state). After some time the plane contains two replicas of the machine, in the same configuration as it was at the start. This machine is complicated, it contains a universal constructor and a universal Turing machine. But it is now proved that there is no logical contradiction in the concept of a self reproducing machine, and that self-reproduction does not need supernatural means. His machine will be improved by numerous successors, Moore in 1961 [40], Burks in 1964 [59, 4], Codd in 1968 [5]. It will be discovered later that the reproduction of actual living organisms follow the pattern of Von Neumann's solution. The DNA of a cell contains the complete description of the cell minus its DNA, the task of construction being entrusted to the ribosomes, that of supervision to the enzymes.

While Von Neumann and his successors tried to find rules for a self-reproductive machine, John Conway (b.1937) builds his "Game of Life" with an entirely different intuition [7, 17, 18, 43]. As Ulam, he thinks that the checkered plane evolving with simple rules is a reduced universe, nevertheless as rich and complex as our real world. Trying out rules securing a balance in complexity he finally works out extremely simple rules : there are only two states, a cell may be live or dead. To revive or survive, a cell must have enough living neighbours, 2 or 3, but not too many, with 4 it is stifled. After having chosen his initial configuration, the player has only to watch the automaton unfold its evolution, and he is greeted with the enchanting spectacle of patterns that flicker, patterns that sail along, patterns that destroy others, patterns that reproduce themselves, ... But twelve more years were actually needed to exhibit a pattern capable of self-reproduction as well as universal computability and constructability, in 1982.

Stephen Wolfram (b.1959) in his turn, and with the idea that cellular automata could act as models for chaos systems in theoretical physics, studies one-dimensional models, i.e. lines of cells, with 2 states (white,black) or 3 (grey added). White is the "quiescent" state, (a quiescent state has the property that a quiescent cell having quiescent neighbours remains quiescent). In this case the evolution with time of the c.a's may be represented by plane diagrams. All the possible transition rules may be duly numbered and exhaustively examined. Systematical observing of the evolution of commonplace initial configurations leads him to distinguish 4 classes of behavior, of increasing complexity : the configurations tend to become 1)entirely white, 2)stable or periodic, 3)chaotic, 4)original, complicated and mortal in a stable scenery. There is no precise definition of these types of behavior. (Nevertheless, Culik and Yu will prove later that the problem of determining wether some c.a belongs to one particular class or not is undecidable [10]). In 1983-84, he publishes several papers [61, 62, 63, 64], famous because of the visionary ideas he develops, namely that

the whole universe could well be governed by some simple program.

In the course of his works on c.a's, Von Neumann had also met with a crucial problem, the problem of synchronization : *with information exchange progressing from cell to cell, how can some cell give an order to all the other ones at the same time.* One measures the importance of this question to orchestrate a collective task. It is Myhill [41], in 1957, who expresses this problem in the vivid terms of the "Firing Squad" : find a (one-dimensional) c.a such that, starting with a configuration where all the cells (the soldiers) are resting, except one (the general), it comes to a configuration where all cells are suddenly in the same (fire) state. This problem became famous with the works of Moore (1964) [40] and Minsky [39]. A great number of solutions and generalizations of this problem have been studied. The nicest and most economical in states is Mazoyer's, the most general one, for networks of automata with communication delays, was given by Jiang in 1990.

Long after Von Neumann had built his machine, which comprised a universal Turing machine, A.R.Smith, in the 70's, demonstrates elementarily that one-dimensional c.a's with only left and right neighbours are equivalent to Turing machines. The simulation of the latter by c.a's immediately implies the undecidable character of most aspects of c.a behavior.

Turing machines are interesting because of their rudimentary constitution, but they are unpractical devices, and despairingly inefficient in realizing the slightest task. In a c.a, a great number of cells work simultaneously, what is called parallelism, which gives them power and speed. Thus, for a certain number of simple problems, very quick c.a algorithms have been found, as Atrubin's one which multiplies two integer numbers in real time. What is worth noticing is that these algorithms often use elementary methods. Whereas in general, efficient algorithms for computers are very tricky. Could the reason not be that c.a's can reproduce what we write on our two-dimensional paper ?

On a sheet of paper, we also solve many problems by using elementary geometry. Well, precisely, one-dimensional c.a's manage to reproduce these solutions using "signals", that is states that progress more or less quickly and leave straight trajectories in the diagrams. The most surprising fact is that c.a's, which are discrete systems, producing discrete space-time diagrams, seem to always faithfully implement the geometric solutions of the continuous world.

Another advantage of the geometric solutions is that when a c.a mimics a geometric solution, we have a mathematical proof of the task it accomplishes.

Do we have other methods to find c.a algorithms ? If the set of states happens to be equipped with an algebraic structure, and the rules for transition are algebraically defined, that is in very particular circumstances, algebra may be used.

Otherwise, we must admit the fact : the complex behavior of c.a's remains very difficult to master. This difficulty probably explains that massive parallelism has not been much used till now for building real machines. A first attempt was the "connexion machine" of D.Hillis [26], capable of quick and reliable computations. But this enterprise was not successful.

The structural efficiency of c.a's does not prevent from trying to speed them

up, as we speed up Turing machines. As a matter of fact the first attempts at doing this used a roundabout way through a special category of Turing machines. Ibarra, Kim and Moran in 1985 [28], then Choffrut and Culik [11], thus proved that language recognition could, roughly, be speeded up by an arbitrary factor. Recently, Nicolas Reimen and Mazoyer have set up a proper c.a method [46]. It is based on a weak speeding up result, needing a preliminary transformation of the languages, which must be expressed with syllables. They had the idea that the c.a could do this transformation itself. After that, remained the technical difficulty of switching all the cells from this first task to the work of recognition, for which an auxiliary synchronization was needed.

With this brief retrospect, one understands that no one book can cover all the approaches to cellular automata. The only aspect treated in the books of Burk and Codd [4, 5] is self reproduction. All the other aspects are studied in numerous and scattered papers. The main ambition of this work is gathering and unifying a certain number of existing results about c.a algorithms, which naturally leads to complements and improvements. Our point of view is resolutely geometrical. Elementary geometry is used

- to find algorithms (for language recognition, slowing down, speeding up)
- to divide problems in smaller problems and progressively reduce them to micro-problems (divide and conquer methods)
- to link partial solutions, that is solutions on different subsets of the cells
- to simulate delays by distances
- finally, we even present c.a's the states of which are geometrical pieces.

The order of the book is from the simple to the more complex, introducing a question as soon as we have the frame to put it and the means to solve it.

Chapter 1 starts with the finite linear c.a's. They are precisely the required framework for introducing the Firing Squad Synchronization Problem, and giving the first and simplest solution, that of Minsky.

Chapter 2 presents Mazoyer's solution which is elegant and sophisticated. The needed display of technics also introduces to the abstract notions to be developed in later chapters.

Chapter 3 clarifies some important notions related to semi-infinite linear c.a's : inputs and outputs, computing a function, recognizing a language. It also gives fundamental examples of language recognition.

It is then possible to relate c.a's with Turing machines, which is done in Chapter 4, which ends with Atrubin's real time algorithm to multipliy two integer numbers.

Semi-infinite linear c.a's are the required framework for many developments in

the next chapters.

Chapter 5 gives a precise definition of a signal. Different notions then appear, those of waves and networks of signals, with the beautiful example of Fischer's c.a for recognizing prime numbers, and the gap theorem for waves.

In chapter 6, slowing down is studied. It may seem absurd at first view. But we shall see that it is useful, next to speeding up, for intelligently driving our c.a's. This incidentally leads us to build a c.a which computes particular word morphisms.

Chapter 7 is an exhaustive study of speeding up, for recognizers and for synchronizers.

Chapter 8 is devoted to the study of the family of synchronization times. The richness of this family allows for flexible use of the synchronization process.

Chapter 9 introduces the geometrical c.a's, i.e. c.a's the states of which are geometrical pieces. Numerous examples give new and elegant methods for speeding up.

Chapter 10 is a general study of n-dimensional c.a's and of the languages that they recognize. It is based on Cole [6].

In the last three chapters, communication delays are introduced. They considerably complexify the behavior of c.a's, in particular their synchronization.

Chapter 11 considers a line of two automata. Solutions for synchronizing these two automata are presented, and they are proved to be almost optimal.

Chapter 12 considers a finite line of automata. A c.a synchronizing such a line is presented, which mimics Minsky's method of chapter 1.

In chapter 13 we first give the general definition of a network, without or with communication delays. We show how automata placed at the nodes of the network may solve graph problems. We then use this possibility to transform the network into a fictive line. Applying the result of the preceding chapter to this line, we obtain a c.a for synchronizing the network.

Contents

Chapter 1

Finite lines - Synchronization

Once and for all, we shall write c.a for cellular automaton !

1.1 Introduction to cellular automata

1.1.1 Single automata

A finite automaton is the mathematical model of some machine whose state may change in time, the set of possible states being finite. Its behaviour is the succession of its states throughout time. Its characteristical features are the set of states, Q, and the rules for their changes. The number of states being finite, there is no question of a continuous course, so the time-scale will be $\mathbb{N}$. Changes are ruled by the function δ mapping state of automaton a at time t, denoted $< a, t >$, on state at time $t + 1$:

$$< a, t + 1 >= \delta(< a, t >).$$

The behaviour of the automaton is studied from an initial state, q_0, it is the sequence :

$$q_0, \delta(q_0), \delta^2(q_0), \ldots, \delta^n(q_0), \ldots$$

As just described, isolated and independent from any outer surrounding, our automaton presents very little interest ; indeed, if k is the number of states in Q, two out of the $k + 1$ states from time 0 to time k must be the same :

$$\exists i, j \quad 0 \leq i < j \leq k \quad \textit{such that} < a, i >=< a, j >,$$

so that a behaves periodically, with period no greater than k. Our automaton is a mere clock.

1.1.2 Interaction with the outer world

The automaton will gain interest in communicating with the outer world, or other automata, or both.

To begin with, let us give our automaton an input and an output :

- the set of possible inputs will be called the *input alphabet* denoted X, and the state of a at a certain time will depend not only on its state but also on its input at the preceding time :

$$< a, t+1 >= \delta(< a, t >, x(t))$$

where $x(t)$ denotes input at time t

- the set of possible outputs wil be called the *output alphabet* , noted Y, the output at a certain time depending only on the state :

$$y(t) = \sigma(< a, t >)$$

σ being the *output function*

- in these two cases "alphabet" is an appropriate denomination because inputs/outputs naturally gather into words.

Figure 1 presents the successive states of a piled up above state at time 0, so the time scale is upwards.

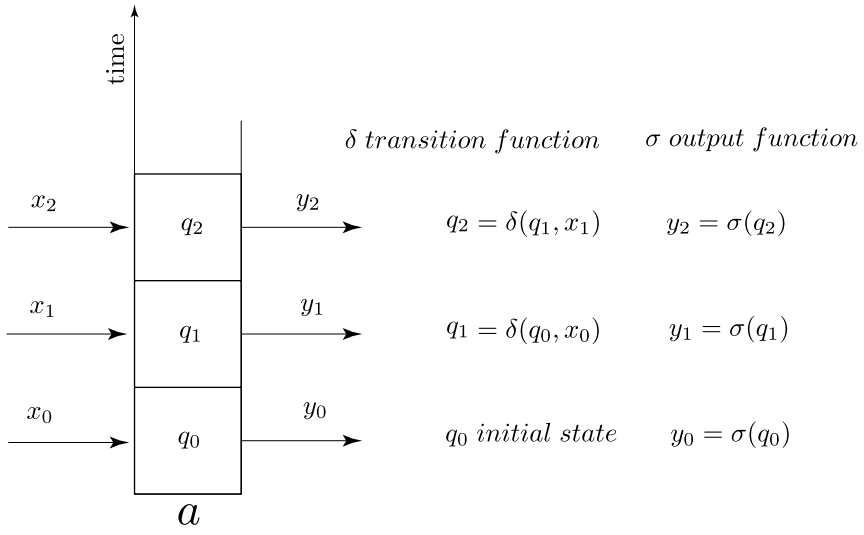


Fig. 1 : diagram of a single automaton a with input and output.

With finite input alphabet and only two outputs, (1 and 0, or else "accepted" and "rejected"), we have the classical theory of *finite automata*.

With finite output alphabet, we have the theory of *General Sequential Machines* .

1.1.3 Interactive automata

Our automaton may also communicate with a second automaton, or several others, which we shall assume to be all identical and identical to the first one. It then becomes the generic element of a colony, hence the name cell given to each such automaton. In this colony, the state of each cell depends on the state of some of the others, usually the nearest neighbours . We immediately imagine the great many varieties of such colonies, called cellular automata (c.a for short), that we can fancy to study.

If the basic cell has an input and an output, the cellular automaton will be able to communicate with the outside. This gives us a glimpse on the numerous situations for which cellular automata may serve as models, from cell clusters to computer networks ...

We shall begin our study with the simplest possible c.a : a finite number n of cells, $c_1, \ldots, c_n$, assembled in a line, where the state of each cell at time $t+1$ depends on its proper state at time t and the states at time t of its two left and right neighbours :

$$< c_i, t+1 > \; = \; \delta(< c_{i-1}, t >, < c_i, t >, < c_{i+1}, t >)$$

for $1 < i < n$. The mapping δ is called the *transition function.*

For $i = 1$ or n, no neighbour will be considered a particular state (not in Q) called "border" state, denoted β, so that the transition function δ always has three arguments :

$$< c_1, t+1 > \; = \; \delta(\beta, < c_1, t >, < c_2, t >)$$

$$< c_n, t+1 > \; = \; \delta(< c_{n-1}, t >, < c_n, t >, \beta)$$

Such a c.a will be called *one-dimensional (or linear), finite, of scope1 (or with first neighbours' neighbourhood).*

And with not more than these simplest of c.a's shall we meet with a fundamental question, the synchronization problem, and get familiar with a number of notions and techniques. But to begin with

1.1.4 A little vocabulary

Let now $\mathcal{A}$ denote some linear c.a of length n with set of states Q. An n-tuple of states of the cells $c_1, \ldots, c_n$ will be called a *configuration* (of states) of c.a $\mathcal{A}$. The set of all possible configurations of $\mathcal{A}$ is Q^n. Configuration of $\mathcal{A}$ at time t will be denoted $< \mathcal{A}, t >$ or $< (c_1, \ldots, c_n), t >$. Thus

$$< \mathcal{A}, t >= (< c_1, t >, \ldots, < c_n, t >).$$

The function Δ which maps a configuration of states of the c.a on the configuration of states at the next time is the *global transition function.* It is the

result of the local transition of each of the cells : if $< \mathcal{A}, t >= (q_1, \ldots, q_n)$ then

$$\begin{aligned} < \mathcal{A}, t+1 > &= \Delta(< \mathcal{A}, t >) \\ &= (\delta(\beta, q_1, q_2) \ldots \delta(q_{i-1}, q_i, q_{i+1}) \ldots \delta(q_{n-1}, q_n, \beta)). \end{aligned}$$

A *calculus* of the c.a is a sequence of configurations starting from an initial configuration, to which the global transition is repeatedly applied, that is a sequence of configurations of the c.a at times 0,1,2, ... This sequence is represented by the *space-time diagram* (s.t.d for short) , illustrated in Figure 2 (where $n = 3$).

Note 1.1.1 *Concerning this diagram, we insist that the notations :* (c, t)*, for a site in the diagram, and* $< c, t >$ *for the state of this site, be clearly distinguished. The first one is the place reserved in the diagram for information about cell* c *at time* t*, the second one is the state of cell* c *at time* t*.*

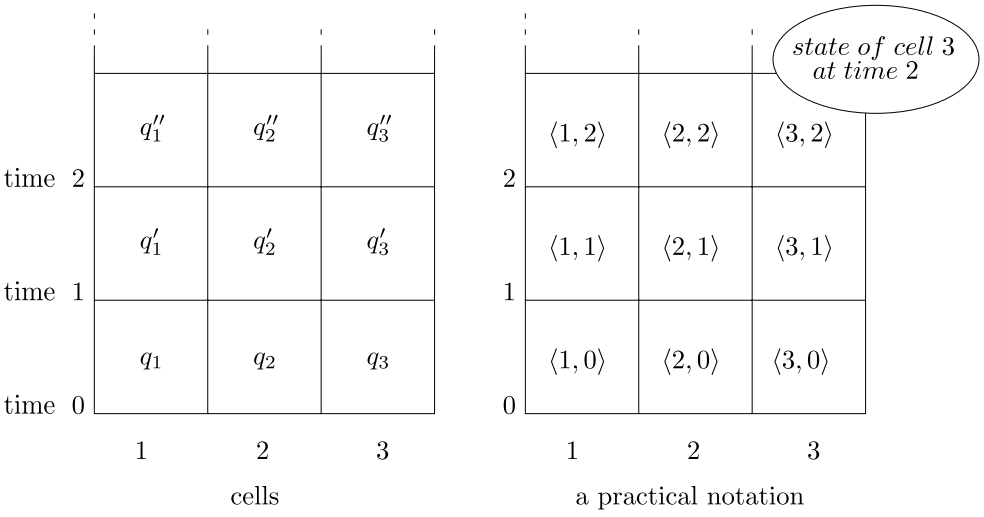


Fig. 2 : space-time diagram of a 3-cell linear c.a

Among the possible states of the cells we shall always find a special and quite important state, called the *quiescent state*, denoted e and satisfiying :

$$\delta(e, e, e) = e \quad \delta(\beta, e, e) = \delta(e, e, \beta) = e.$$

A set of cells in quiescent state will remain quiescent unless some neighbouring cells are active and interfere, or the outer world intrudes through the inputs.

The c.a will be said to be *synchronized* in state q at time t if all of the cells are in this same state q at that time :

$$\forall i \quad 1 \leq i \leq n \quad < c_i, t >= q.$$

1.2 The firing squad synchronization problem

Abbreviation for this problem is F.S.S.P. The question is :

> does there exist a set Q containing, (next to the quiescent state e), a state G, a state $*$, and a mapping $\delta : Q^3 \longmapsto Q$
>
> such that
>
> each line of automata L_n, whatever its length n, starting in configuration $(G, e, \ldots, e)$, will be synchronized at some time $T(n)$ in state $*$, which should not appear anywhere before ?

(See Figure 3).

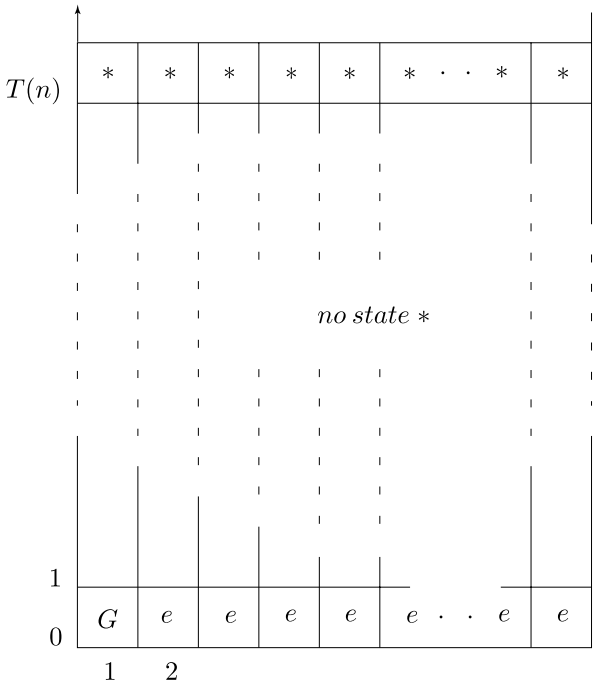


Fig. 3 : the F.S.S.P

Function $T : \mathbb{N}^* \longmapsto \mathbb{N}^*$ will be called the synchronizing time for the lines. It is the number of transition steps necessary to pass from the initial state of the line to the $*$ state, as well as the precise time when synchronization occurs if the starting time was 0.

We can imagine cell 0 is a general (state G), the other cells are soldiers, all alike, each of them communicating only with his two neighbours, and indeed ignoring how many they can be. The general himself communicates only with

the soldier at his right. At time 0, as all soldiers sleep, the general gives an order. The question is : is it possible to find rules for them to act so that they finally all fire (state $*$) at the same time ?

For all its wording in the shape of a macabre, but quite evocative joke, this problem is crucial and synchronization is, as we shall see later on, an essential tool in the domain of cellular automata.

We insist that this synchronization problem does not relate to some c.a considered by itself, but to some family of c.a's made up from the same generic cell (whose set of states is Q), having the same structure (finite line and scope 1), and the same transition function δ.

To synchronize some particular c.a of the family, for instance line L_n of length n, is trivial : just consider the following automaton which counts up to n

$$Q = \{e,\ \ G = q_1,\ \ q_2\ , \ldots\ ,\ \ q_n = *\}$$

$$\delta(?, q_i, ?) = q_{i+1} \quad \text{and} \quad \delta(q_i, e, e) = q_{i+1}\ , \qquad \text{for} \quad i = 1, \ldots, n-1$$

where ? denotes any state in Q. Synchronization is achieved at time $n-1$, which is actually the time needed for the last cell n to be drawn out of quiescent state.

In the actual problem, *all lengths of lines are considered*, Q and δ have no knowledge of any length, what we express by saying that the "general" ignores the number of soldiers in the line.

1.3 Minsky's solution to the FSSP

It is the simplest one. It will lead us to introduce the notion of a signal, in its most elementary form, that of a "threadlike" or "threadthin" or "thickness 1" signal, on quiescent background.

1.3.1 Intuition

Minsky proceeds by a divide-and-conquer strategy : he breaks a line in two half-lines, which are in turn broken in two and so on, until all lines have length 1 and then synchronization occurs.

How can we break the line in two ? Better intuition is gained when considering a continuous universe where the geometry of the solution can appear free from the constraints of a discrete world. This was really the means by which the idea first came (see Figure 4) : in (x, t) axis-system, a straight line of slope 3 and a straight line of slope 1 reflected by the vertical line of abciss l meet at point of coordinates $(l/2, 3l/2)$. The half-lines thus determined will be broken by the same way, indefinitely.

The initial line of length l is broken into segments of length $l/2i$ at time

$$\frac{3l}{2} + \frac{3l}{2^2} + \cdots + \frac{3l}{2^i} = 3l(1 - \frac{1}{2^i}).$$

As i becomes infinite, length of the segments tends to 0 and the limit time is $3l$. In Minsky's solution for the discrete case, fractioning of the line will cease sooner, so we expect a time less than $3n$.

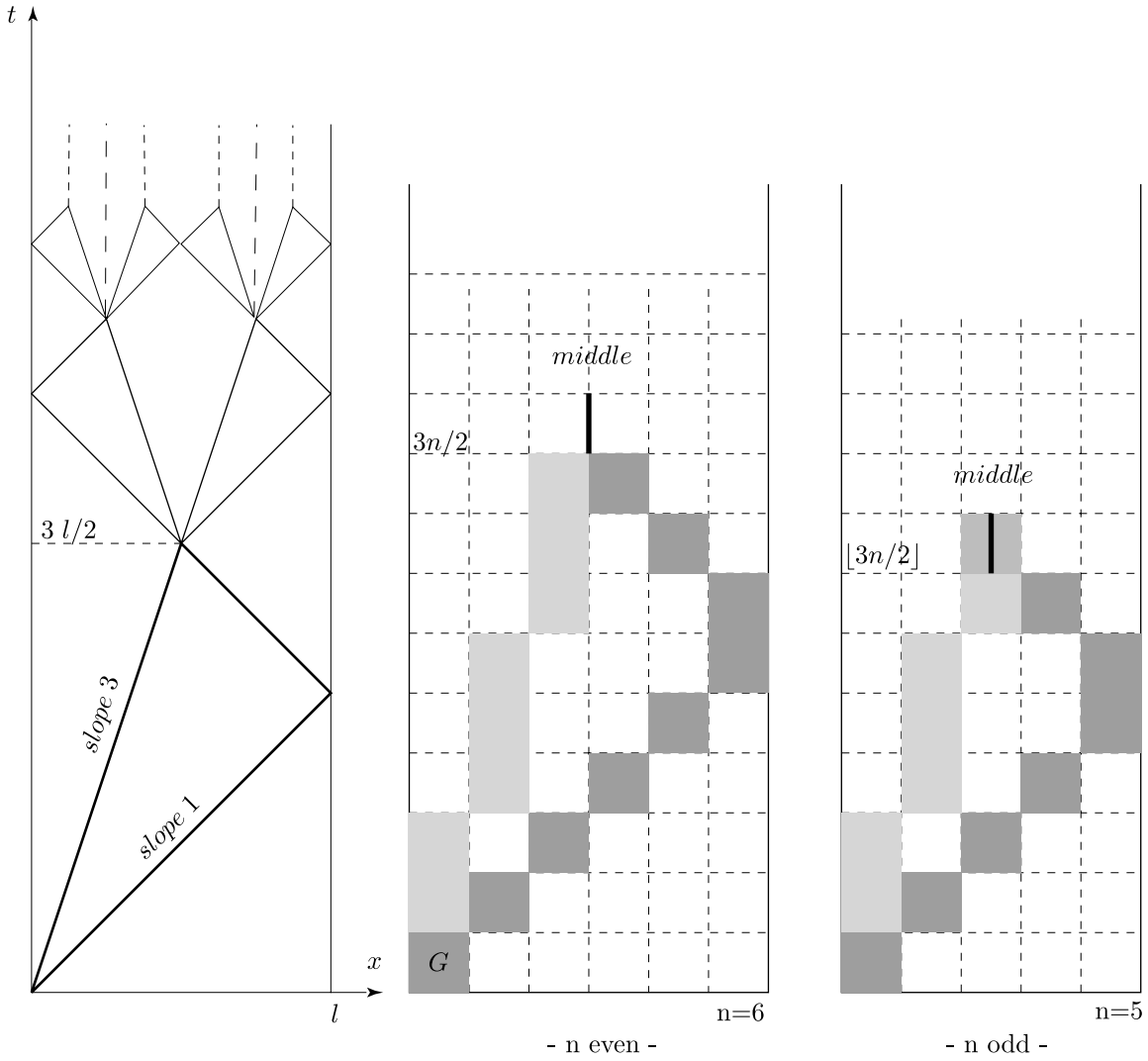


Fig. 4 : Minsky's solution, from the geometrical idea to its discrete realization.

1.3.2 First and fundamental signals

In the discrete space-time diagrams continuous lines are approximated by discrete *signals* : one or several states which, by way of the transition, propagate through the cells along a precisely outlined path. Let us give a few examples :

Example 1 : a state A_r with transition rules

$$\delta(e,e,A_r) = e \;\; \delta(e,A_r,e) = e \;\; \delta(A_r,e,e) = A_r$$

gives way, if the starting configuration is one cell in state A_r amidst quiescent cells, to a signal of slope 1, advancing rightwards one cell per time unit (see on Figure 13). Observe that this is the maximum possible speed.

Symmetrically, a state A_l with transition rules

$$\delta(e,e,A_l) = A_l \;\; \delta(e,A_l,e) = e \;\; \delta(A_l,e,e) = e$$

gives way to a signal going leftwards at maximal speed 1.

Example 2 : 3 states B_{r1}, B_{r2}, B_{r3} with rules

$$\delta(e,B_{r1},e) = B_{r2} \quad \delta(B_{r1},e,e) = \delta(e,e,B_{r1}) = e$$

$$\delta(e,B_{r2},e) = B_{r3} \quad \delta(B_{r2},e,e) = \delta(e,e,B_{r2}) = e$$

$$\delta(B_{r3},e,e) = B_{r1} \quad \delta(e,B_{r3},e) = \delta(e,e,B_{r3}) = e$$

give way, if the starting configuration is one cell in state B_{r1} amidst quiescent cells, to a signal of slope 3 in the space-time diagram, advancing at a rate of one cell every three time units (see on Figure 13).

In Minsky's c.a we shall also have the symmetrical signal B_{l1}, B_{g2}, B_{g3} .

Example 3 : reflection of signals on the border

A_r reflects on the right border and becomes A_l . Here two reflections are possible, a slow reflection (figure 5a) with rules :

$$\delta(A_r,e,\beta) = A_r \qquad \delta(e,A_r,\beta) = A_l$$

$$\delta(e,A_l,\beta) = e \qquad \delta(A_l,e,\beta) = e$$

and a fast reflection (figure 5b) with, instead of the first two rules, the only rule :

$$\delta(A_r,e,\beta) = A_l$$

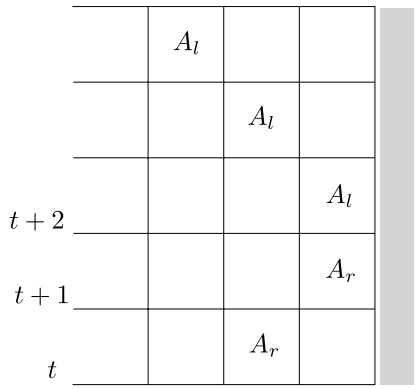


Fig. 5a : slow reflection

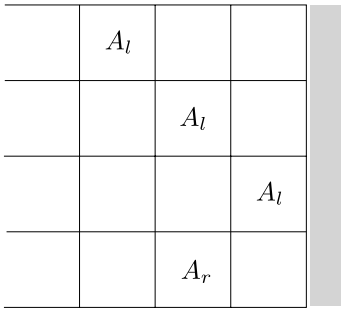


Fig. 5b : fast reflection

In Minsky's c.a we shall have a symmetrical reflection of A_l against the left border.

Example 4 : A_l and A_r when they meet

A_r and A_l reflect against one another, rules are

$$\delta(e, A_r, A_l) = A_l \qquad \delta(A_r, A_l, e) = A_r$$
$$\delta(e, A_l, A_r) = e \qquad \delta(A_l, A_r, e) = e$$

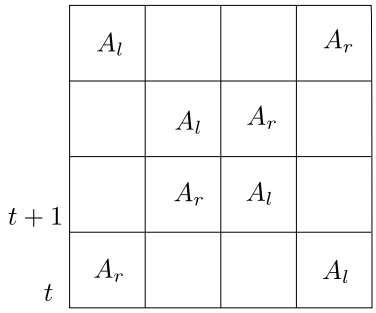


Fig. 6a

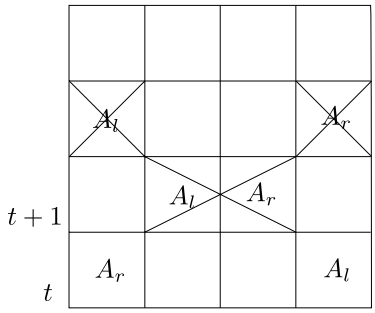


Fig. 6b : impossible

Looking at Figure 6a we see two possible ways of expressing these rules. We can say that A_l acts as a right border for A_r , and A_r as a left border for A_l . But we could also and more simply say that A_r and A_l cross each other without interfering. The first expression is more suitable in dealing with Minsky's c.a, as we shall see.

A natural question is "could we have a fast reflection of these two signals against one another", as shown in Figure 6b ? The answer is no, because

at time $t+1$ cell c (resp.$c+1$) cannot (scope is of one cell) perceive A_l (resp.A_r) approaching.

This is the reason why, in Minsky's c.a, we cannot use the fast reflection on the borders, as we shall see in section 1.3.3.

Example 4 : generation of signals

Let us say : state G (for symmetry reasons we shall henceforth prefer notation G_r) generates two signals, A_r and (B_{r1}, B_{r2}, B_{r3}). This lacks precision, so we can add that in this process G_r plays the part of states A_r and B_{r1} (Figure 7). Rules are

$$\delta(\beta, G_r, e) = B_{r2} \qquad \delta(G_r, e, e) = A_r.$$

We already guess that, in Minsky's c.a, we shall have a symmetrical generation by a state G_l.

	B_{r3}		
	B_{r2}		
	B_{r1}		A_r
B_{r3}		A_r	
B_{r2}	A_r		
G_r			

		B_{l3}	
		B_{l2}	
A_l		B_{l1}	
	A_l		B_{l3}
		A_l	B_{l2}
			G_l

Fig. 7

1.3.3 How to present transition function of a c.a ?

Through these four examples, it appears that transitions of a c.a can be described in different manners :

- by listing the rules. When they are many, we group them into tables, one table for each central state , the table associated to state q describing the binary function $(l, q, r) \mapsto \delta(l, q, r)$, as in Figure 11

- by portions of space-time diagrams (see preceding figures), where several rules appear together. These portions can be smaller or bigger, from four sites showing one rule (Figure 8), to a complete space-time diagram !

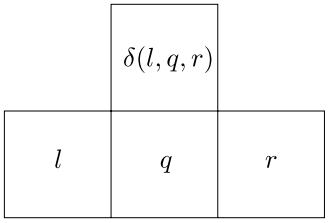


Fig.8 : diagram of one rule

- by sentences, as : A_r is a rightwards speed 1 signal ...

Exhaustive listing of the rules seems the more precise presentation, but it is quite unreadable. Portions of space-time diagrams are more pleasant and suggestive but may often contain repetitions (they certainly must contain no contradictions !) and we may forget particular cases. Sentences may lack precision, but are quick and quite evocative. In building c.a's we shall seldom use the listing of rules, but this is certainly what is pertinent for a computer program.

1.3.4 Minsky's solution in case $n = 2^p$

Coming back to our problem, let us begin with the very simple case when n is a power of 2 : $n = 2^p$ (Figure 9).

Signals B_r and A_l meet on both sides of the middle line at time $3n/2 - 1$. We decide that this meeting produces at the next time, on the same cells, states G_l and G_r, which leads to transitions

$$\delta(e, B_{r3}, A_l) = G_l \qquad \delta(B_{r3}, A_l, e) = G_r$$

where Gg is a new state, which behaves symmetrically to G_r : it generates signals B_g and A_l, as announced in example 4.

Next we must take care that everything goes on at the middle as if it were a new border for each half line, and this is clear if signals A_r and A_l reflect against each other as we have seen in example 3 (see Figure 9, times 19, 20).

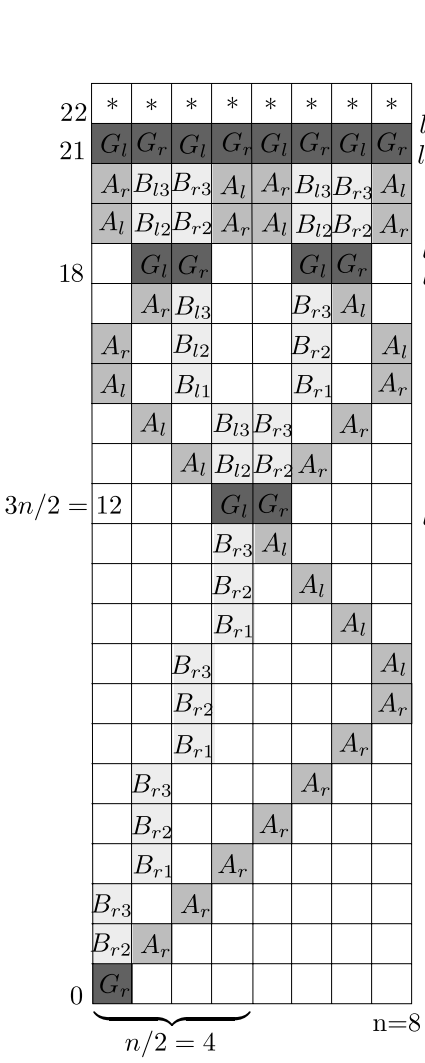


Fig. 9

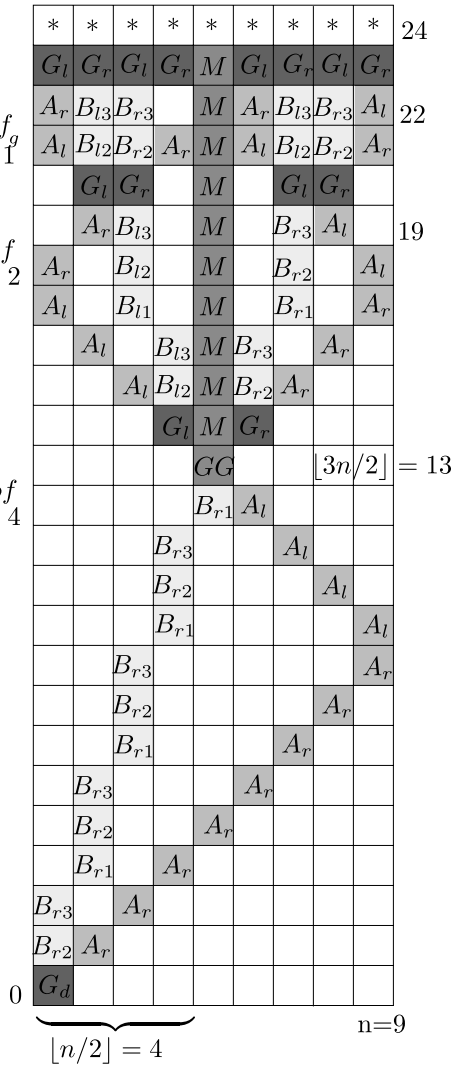


Fig.10

Let us now observe in this same Figure, representing case $n = 8$, the end of the process, when all lines have length 1 and are in state G_l or G_r. With rules

$$\delta(\beta, G_l, G_r) = \delta(G_l, G_r, G_l) = \delta(G_r, G_l, G_r) = \delta(G_l, G_r, \beta) = *,$$

we get synchronization.These rules lead to set for an original line of length 1

$$\delta(\beta, G_r, \beta) = *,$$

so that, in case $n = 1$, the synchronization time is $T(1) = 1$.

By the way, as examples of rules that we would probably not bother to mention in sentences but must add in the listing, are the rules expressing that signals B_r/B_l propagate normally, that is as if the background were quiescent, even if the background is not quiescent, such as

$$\delta(\beta, B_{r2}, A_r) = B_{r3} \qquad \delta(B_{g2}, B_{r2}, A_r) = B_{r3}$$

$$\delta(B_{r3}, e, A_r) = B_{r1} \qquad \delta(B_{r1}, e, A_r) = e \ldots$$

Always in Figure 9 recurrence formula for even n appears clearly

$$T(n) = \frac{3n}{2} + T(\frac{n}{2}).$$

Thus total synchronization time for $n = 2^p$ will be

$$\begin{aligned} T(2^p) &= \frac{3}{2}(2^p) + T(2^{p-1}) \\ &= \cdots \\ &= \frac{3}{2}(2^p + 2^{p-1} + \cdots + 2) + T(1) \\ &= 3.2^p - 2 = 3n - 2 \end{aligned}$$

For the case $n = 2^p$, we conclude that the 12 states :

$$e, *, G_r, G_l, A_r, A_l, B_{l1}, B_{l2}, B_{l3}, B_{r1}, B_{r2}, B_{r3}$$

are sufficient.

1.3.5 Minsky's solution for the general case

In the process of repeatedly dividing lines, about one line out of two will have odd length, so we must see how to break such a line.

At time $3\lfloor n/2 \rfloor - 1$ situation is (Figure 10)

B_{r1}	A_l
middle cell	*right neighbour*

For the next time, (which is $3\lfloor n/2 \rfloor$), we have two possible choices :

1. consider two half lines of length $\lceil n/2 \rceil$ and then set a two-sided general on the middle cell, which will belong to the two half lines. In this case, recurrence formula is

$$T(n) = \left\lfloor \frac{3n}{2} \right\rfloor + T(\left\lceil \frac{n}{2} \right\rceil)$$

2. consider two half lines of length $\lfloor n/2 \rfloor$ and then set up, one unit of time later, a state M (middle) with G_l on its left and G_d on its right. This is done via the following rules :

$$\delta(e, B_{r1}, A_l) = GG$$

$$\delta(e, e, GG) = G_l \quad \delta(e, GG, e) = M \quad \delta(GG, e, e) = G_r.$$

In this case, recurrence formula is :

$$T(n) = \left\lfloor \frac{3n}{2} \right\rfloor + 1 + T(\left\lfloor \frac{n}{2} \right\rfloor).$$

We choose the second formula, which will lead to much simpler calculations. (This choice was made in Figure 10).

At the beginning (Figure 12, time 0) we have a single G_r state. After the first breaking in two of the line, this G_r has disappeared, replaced by $G_l G_r$ (or $G_l M G_r$) (Figure 12, time 21) in the middle of the line. In the same way, at each successive breaking in two of the sublines, each G_l (resp.G_r) will be replaced by $G_l G_r$ (or $G_l M G_r$) in the middle of the subline it commands (Figure 12, time 32, 37). Moreover sublines, all along the process, all have the same length. It is then clear that the breaking in two process will stop when the sublines have length 1, configuration of the line then being a sequence of alternating G_l and G_r's, possibly separated by isolated M's.

We want this line to give a fire line at next time : as in the case of even lines and sublines, states G_l / G_r having two states G_r / G_l as left and right neighbours will give the fire state. But we have a little problem with the M states, because we don't want to get the fire state $*$ from the $G_l M G_r$ triples produced all along by state GG when odd sublines are broken in two . So we must complicate things a bit, by adjoining two states G_l^* and G_r^*, that state GG will produce when it approaches a border or an M state, that is when the present subline-length is 2, and next subline-length will be 1. More precisely, we set

$$\delta(\beta \ \ or \ \ M, e, GG) = G_l^* \qquad \delta(GG, e, \beta \ \ or \ M) = G_r^*.$$

These states will satisfy

$$\delta(G_l^*, M, G_r) = \delta(G_l, M, G_r^*) = \delta(G_l^*, M, G_r^*) = *.$$

Finally, all triples formed with states G_l, G_r, G_l^*, G_r^* and M, except $G_l M G_r$, will give the fire.

We shall sum up all the rules in tables, in Figure 11, which will definitely convince us that such tables, first do not help us understand how the c.a works, and second will be quite impossible to establish when the number of states increases.

We can count the states, which are now 16, because we have added GG, M, G_r^* and G_l^*.

e	β	M	e	GG	G_l	A_r	A_l	B_{r1}	B_{r2}	B_{r3}	B_{l1}	B_{l2}	B_{l3}
β			e	G_l^*	A_l	e	A_l	e	e	e			
M				G_l^*				e	e	e			
e	e		e	G_l	A_l	e	A_l	e	e	e			
GG	G_r^*	G_r^*	G_r										
G_r	A_r		A_r										
A_l	e		e								e		B_{l1}
A_r	A_r		A_r									A_r	
B_{l1}		e	e										
B_{l2}		e	e										
B_{l3}		e	e										
B_{r1}						e							
B_{r2}							A_l						
B_{r3}						B_{r1}							

3 symmetrical tables
around
symmetrical states
e, M, GG

M	G_r	G_l	G_r^*	G_l^*	B_{r2}	B_{r3}
G_l	M		$*$			
G_r		$*$				
G_l^*	$*$		$*$			
G_r^*				$*$		
B_{l2}					M	
B_{l3}						M

GG	e
e	M

6 tables to be completed by the 6 symmetrical tables obtained by mirror image and exchanging g and d states

G_r	β	e	M	G_l
β	$*$	B_{r2}		
M		B_{r2}		$*$
G_l	$*$	B_{r2}	$*$	$*$

G_r^*	β	M
M	$*$	$*$

A_r	β	M	e	B_{l1}	B_{l3}
β			e	e	G_l
e	A_l	A_l	e	e	G_l
B_{r2}	A_l	A_l	e		

B_{r1}	e	A_l
e	B_{r2}	GG

B_{r2}	e	A_d
β	B_{r3}	B_{r3}
M	B_{r3}	B_{r3}
e	B_{r3}	
B_{l2}	B_{r3}	B_{r3}

B_{r3}	e	A_l
β	e	G_l
M	e	G_l
e	e	G_l
B_{l3}	e	G_l

denotes rules that have
no symmetrical rule because
time 0 configuration
is not symmetrical

Fig. 11 : complete tables of transition for Minsky's solution.

In Figure 12 are represented the particular cases of the smallest lines and Figure 13 and 14 represent cases $n = 14$ and $n = 16$.

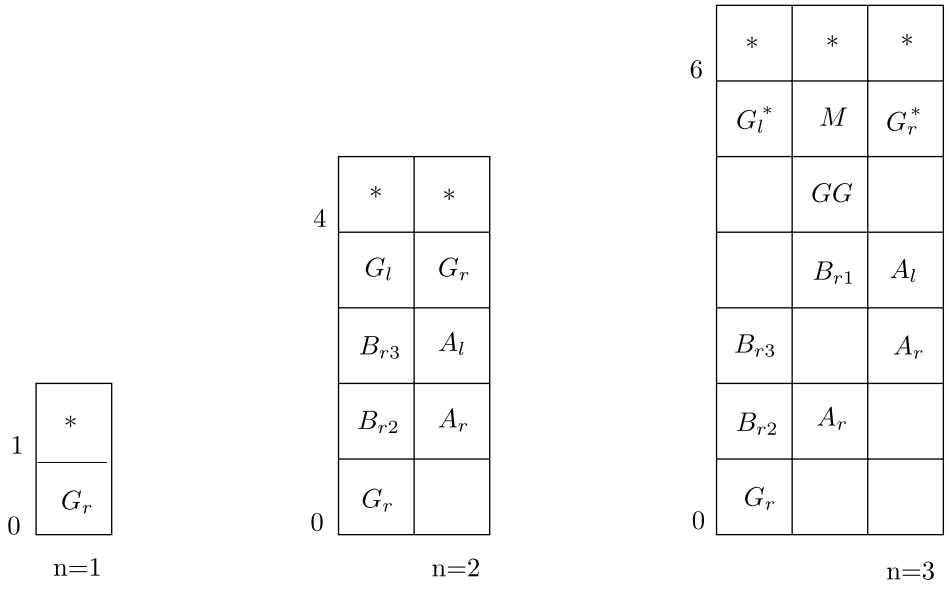


Fig. 12 : Minsky's solution for the very small lines

1.3.6 Synchronization time

Let us now calculate the time for synchronization. The recurrence formulas for even and odd n can be written :

$$T(2p) = 3p + T(p) \qquad T(2p+1) = 3p + 2 + T(p)$$

that is, for $\varepsilon = 0$ or 1

$$T(2p+\varepsilon) = T(p) + 3p + 2\varepsilon.$$

Let us write n in basis 2

$$n = 2^k + \varepsilon_{k-1}2^{k-1} + \varepsilon_{k-2}2^{k-2} + \cdots + \varepsilon_1.2 + \varepsilon_0 \qquad (\varepsilon_k = 1)$$

where $\varepsilon_i = 0$ *or* 1, $i = 0, \ldots, k-1$. Using Horner's classical factorization of polynomials (see Donald KNUTH [32]) n appears as the result of multiplications by 2 and additions of 1 (if $\varepsilon_i = 1$) or 0 :

$$n = \{[\ldots((2.1 + \varepsilon_{k-1})2 + \varepsilon_{k-2})\ldots]2 + \varepsilon_1\}2 + \varepsilon_0$$

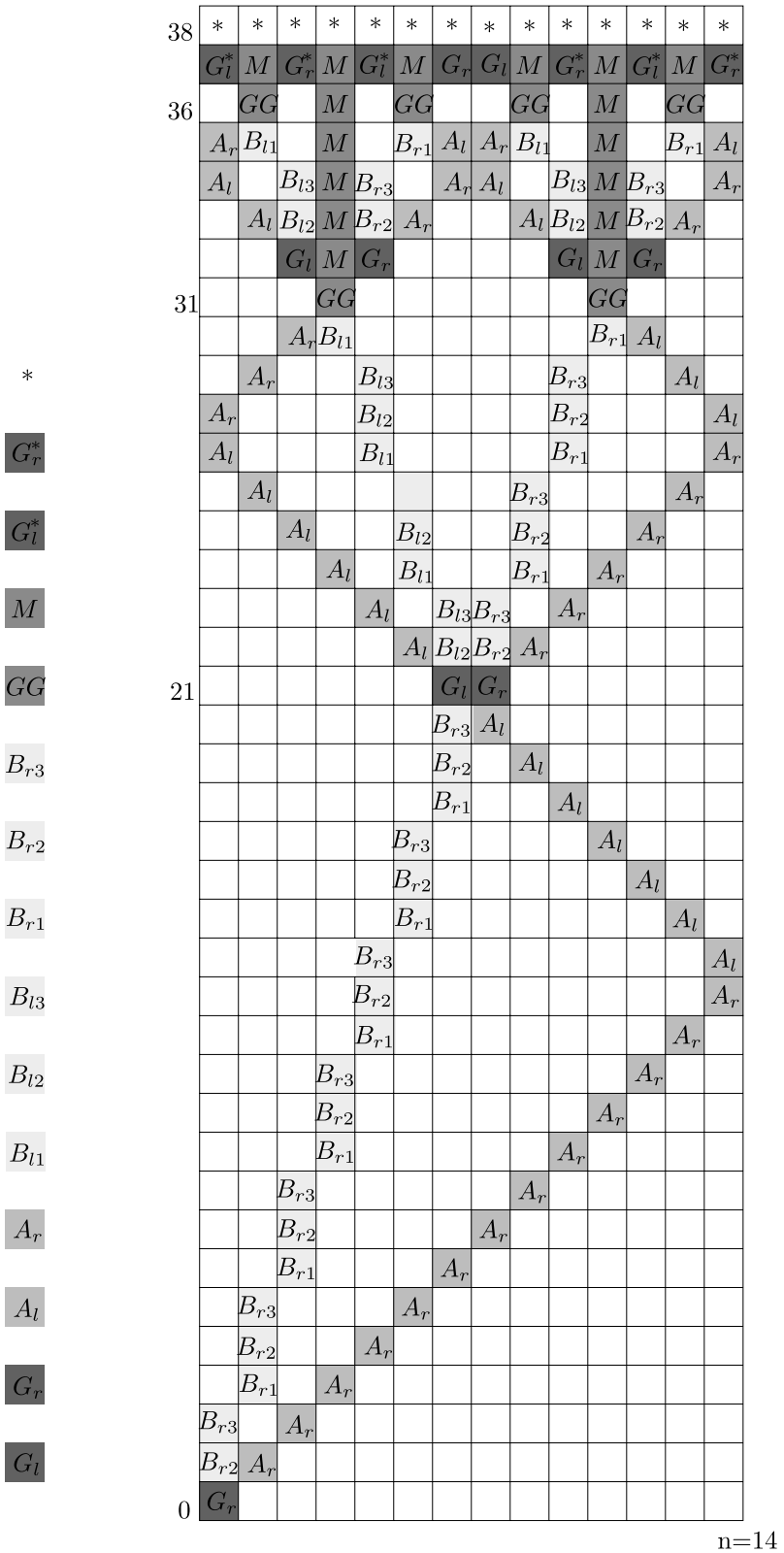

38
36
31
21
0
n=14
*
G_r^*
M
GG
G_r
G_l^*
M
GG
B_{r3}
B_{r2}
B_{r1}
B_{l3}
B_{l2}
B_{l1}
A_r
A_l
G_r
G_l

Fig. 13

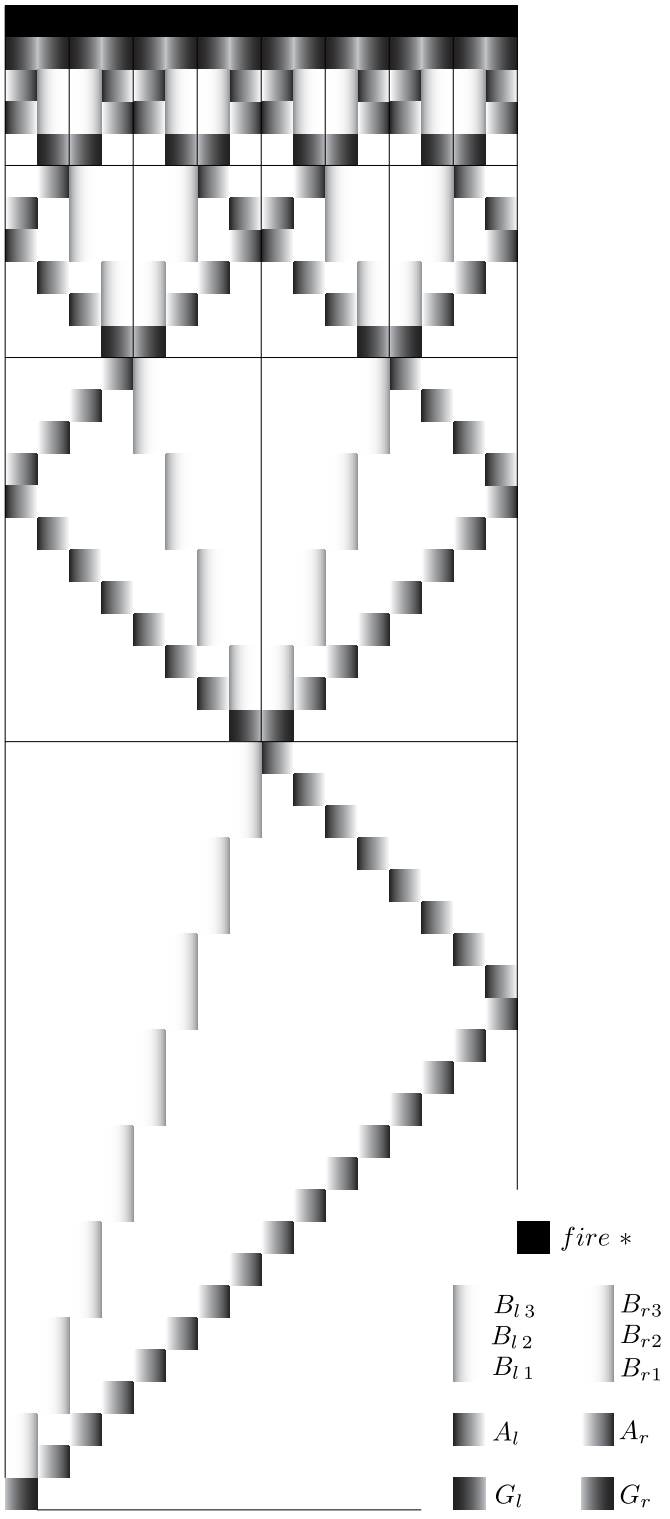


Fig. 14

From this we deduce :
for $n_1 = 2 + \varepsilon_{k-1}$

$$T(n_1) = T(1) + 3 + 2\varepsilon_{k-1}$$

for $n_2 = 2n_1 + \varepsilon_{k-2} = 2^2 + 2\varepsilon_{k-1} + \varepsilon_{k-2}$

$$T(n_2) = T(n_1) + 3n_1 + 2\varepsilon_{k-2}$$

...
for $n_i = 2n_{i-1} + \varepsilon_{k-i} = 2^i + 2^{i-1}\varepsilon_{k-1} + \cdots + \varepsilon_{k-i}$

$$T(n_i) = T(n_{i-1}) + 3n_{i-1} + 2\varepsilon_{k-i}$$

...
for $n = n_k = 2^k + 2^{k-1}\varepsilon_{k-1} + \cdots + \varepsilon_0$

$$T(n) = T(n_{k-1}) + 3n_{k-1} + 2\varepsilon_0$$

so

$$T(n) = T(1) + 3[1 + n_1 + \ldots + n_{k-1}] + 2[\varepsilon_{k-1} + \varepsilon_{k-2} + \ldots + \varepsilon_0] =$$

$$= 1 + 3[(1 + 2 + \cdots + 2^{k-1}) + \varepsilon_{k-1}(1 + 2 + \cdots + 2^{k-2}) + \cdots + \varepsilon_1] + 2(\varepsilon_{k-1} + \cdots + \varepsilon_0) =$$

$$= 1 + 3[(2^k - 1) + \varepsilon_{k-1}(2^{k-1} - 1) + \cdots + \varepsilon_1(2 - 1)] + 2(\varepsilon_{k-1} + \cdots + \varepsilon_0).$$

We finally obtain

$$T(n) = 3n - 2 - \sum_{i=0}^{k-1} \varepsilon_i = 3n - 1 - \sum_{i=0}^{k} \varepsilon_i = 3n - 1 - |bin(n)|_1$$

where $|bin(n)|_1$ denotes the number of 1's in the binary decomposition of n. We may notice that $\varepsilon_i = 1$ if and only if $\lfloor n/2_i \rfloor$ is odd ($\varepsilon_0 = 1$ if n is odd) so $|bin(n)|_1 - 1$ is also the number of times in the process when we have to break odd line or sub-lines (for k there are no more lines to break). This formula yields, in case $n = 2^p$ (no odd lines)

$$T(n) = 3n - 2$$

and in case the binary decomposition of n has only 1's, that is if $n = 2^{k+1} - 1$,

$$T(n) = 3n - 2 - k = 3n - 1 - \log(n + 1).$$

Let us just mention here results of the same sort of calculations using the first recurrence formula of 1.3.5 :

$$T(n) = \left\lfloor \frac{3n}{2} \right\rfloor + T(\left\lceil \frac{n}{2} \right\rceil).$$

Writing n as

$$n = 2^{k+1} - 0.2^k - \eta_{k-1}2^{k-1} - \eta_{k-2}2^{k-2} - \cdots - \eta_1.2 - \eta_0 \qquad (\eta_k = 0)$$

(where $\eta_k = 0, \eta_{k-1}, \eta_{k-2}, \cdots, \eta_1, \eta_0$ are the 1-complements of the binary digits of $n-1$) we obtain

$$\begin{aligned} T(n) &= T(1) + 3(n-1) + (\eta_{k-1} + \cdots + \eta_1 + \eta_0) \\ &= 3n - 2 + |bin(n-1)|_0 \end{aligned}$$

This time is a bit more than the preceding, so we had all the reasons to choose the second solution.

1.3.7 The 2 ends-FSSP

We shall end by mentioning the 2e-FSSP, which is the same problem except that there are two generals, one at each end of the line (2e stands for two ends). And we shall distinguish the synchronizing times for the two problems, from now on, by denoting them respectively T_{1e} and T_{2e} .

An obvious solution consists in first parting the line in two with 2 fast signals (of slope 1 and -1) sent respectively by the two generals. If length of the line is even, $n = 2p$, at time p cell p knows it is the last of the first half-line of length p, (while symmetrically cell $p+1$ knows it is the first of the other half-line). Two symmetrical simple Minsky synchronizations of the two half-lines can then be achieved in time :

$$\begin{aligned} T_{2e}(n) = p - 1 + T(p) &= p - 1 + 3p - 1 - |bin(p)|_1 \\ &= 4p - 2 - |bin(p)|_1 \\ &= 2n - 2 - |bin(n)|_1 < 2n - 2 \end{aligned}$$

If length of the line is odd, $n = 2p+1$, at time p cell $p+1$ knows it is the middle cell. Two simple Minsky synchronizations of the two overlapping half-lines of length $p+1$ can be achieved in time

$$\begin{aligned} T_{2e}(n) = p - 1 + T(p+1) &= p - 1 + 3(p+1) - 1 - |bin(p+1)|_1 \\ &= 4p + 1 - |bin(p+1)|_1 \\ &= 2n - 1 - |bin(p+1)|_1 < 2n - 1 \end{aligned}$$

As a matter of fact such synchronizing times that have complicated expressions are not very useful. But we shall come back to the 2e-FSSP problem later, in a more interesting case.

1.4 Minimal synchronizing time

1.4.1 Minimal synchronizing time for the FSSP

In this section we do not consider the line consisting of a single cell, for which the following proof is not correct. Thus, denoting L_n the line of length n, our

family of c.a's is $(L_n)_{n\geq 2}$. They all start with initial configuration $(G, e, \ldots, e)$. As a consequence of the transition rules for the quiescent state the end cell n of L_n necessarily remains in quiescent state from time 0 to time $n-2$ included (see Figure 15). At time $T_{1e}(n)$ the whole line, cell n included should be in fire state, so certainly

$$T_{1e}(n) > n-2, \quad \text{or} \quad T_{1e}(n) \geq n-1.$$

Before we go further, let us record a little obvious lemma :

Lemma 1.4.1 *If two lines have the same states on cells $c-1$, c, $c+1$ at time t (where $c-1$ / $c+1$ may be the left / right border) then they have the same state on cell c at time $t+1$.*

Let us now consider, together with L_n, a second longer line $L_{n'}$, $n' > n$: up to time $n-2$ included, the n^{th} cells of both lines are likewise in quiescent state. A simple iteration of the preceding lemma proves that all their cells from 1 to n are in the same states up to time $n-2$ included. Then all the sites $(cell\ c, time\ t)$ for

$$t = (n-2)+i \quad c \leq n-i \quad 1 \leq i \leq n-1$$

are also in the same states. In particular

cell number 1 at time $2n-3$.

Let us then consider some n' greater than $2n$ (Figure 15) : according to our previous result, cell 1 of $L_{n'}$ cannot be in fire state at time $2n-3$. Then cell 1 of L_n cannot either ! From this results that

$$T_{1e}(n) > 2n-3, \quad \text{that is} \quad T_{1e}(n) \geq 2n-2.$$

We shall see later that this is the greatest lower bound for the synchronizing times of the L_n's.

For $n = 1$, $T_{1e}(n)$ is of course greater than $2n-2 = 0$, but 0 is not the greatest lower bound, which must be at least 1 because state G is not the fire state !

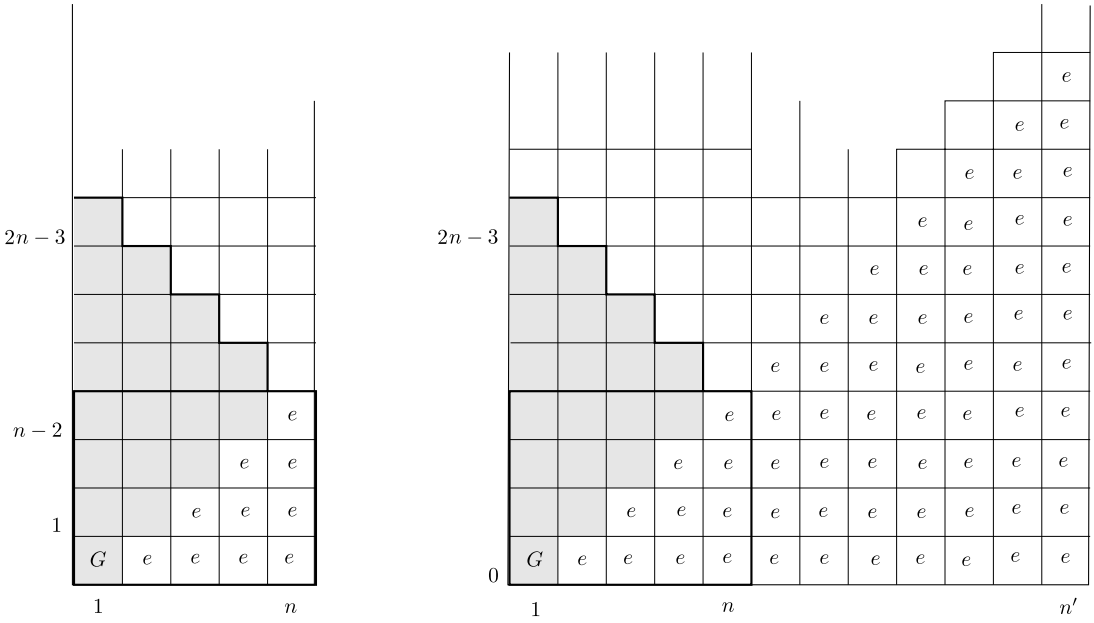
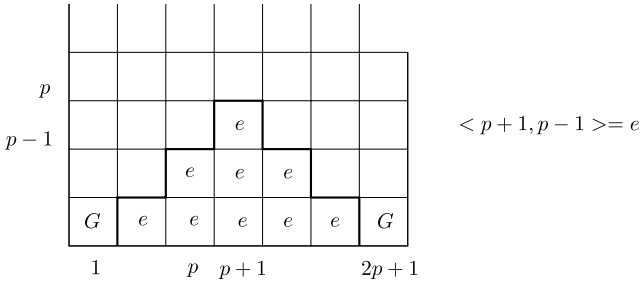

2n − 3
n − 2
1
G e e e e
1
n
2n − 3
0
1
n
n′

Fig. 15


p
p − 1
G e e e e e G
1
p
p + 1
2p + 1
< p + 1, p − 1 >= e

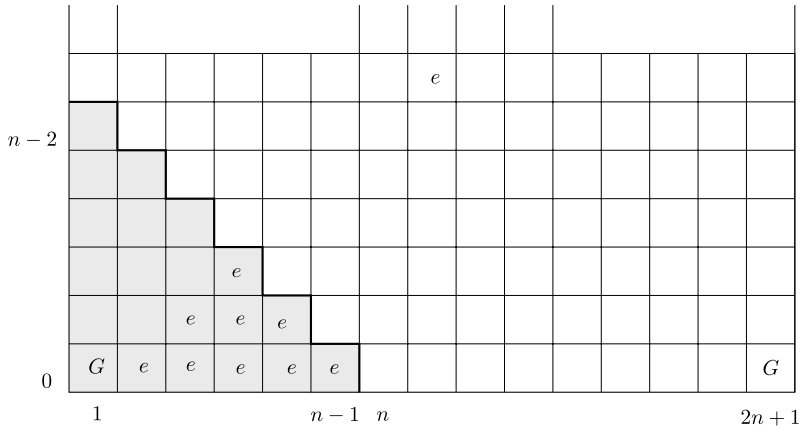

n − 2
0
G e e e e e G
1
n − 1
n
2n + 1

Fig. 16

1.4.2 Minimal synchronizing time for the 2e-FSSP

We shall reason in quite the same way for the 2e-FSSP (Figure 16).

Here the line consisting of one single cell is discarded because at least two cells are needed. Initial configuration of line L_n, $n \geq 2$, is $(G, e, \ldots, e, G)$. Let us first notice that for an odd line $n' = 2p + 1$, site (cell p+1,time p-1) has quiescent state, so :

$$T_{2e}(2p + 1) \geq p.$$

Consider now L_n and $L_{n'}$, $n' > n$. At time 0, their cells 1 to $n - 1$ are in the same states. Then their sites $(n - 1 - t, t)$ for t ranging from 0 to $n - 2$ are also in the same states, particularly cells 1 at time $n - 2$.

So now let us have $n' = 2n + 1$. As $T_{2e}(n') \geq n$ (from first result), cell 1 of $L_{n'}$ cannot be in fire state at time $n - 2$, so cell 1 of L_n cannot either. Thus

$$T_{2e}(n) \geq n - 1.$$

Again, we shall see later that this is the greatest lower bound for the synchronizing times of the $2e - L_n$'s.

1.5 A solution in minimal time $T_{1e}(n) = 2n - 2$

(More precisely $max(1, 2n - 2)$ if we should include line of length 1).

Several solutions have been imagined, with less and less states, each bringing new technical ideas :

- 1962, E.Goto, a few thousands of states
- 1966, A.Waksman, 16 states
- 1967, R.Balzer, 8 states
- 1987, J.Mazoyer, 6 states.

We choose to give Mazoyer's solution, which has the least number of states till now, and is particularly instructive, being obtained through two contrasted methods, the first a solid logical construction, followed by a craftsman tinkering. Final result is remarkably simple. We shall leave this for a separate chapter because explanations are really long.

Should we expect new and better solutions ?

A three-state (necessarily G, $*$ and e) solution cannot be dreamed of, as can be seen by examining the possible s.t.d's for lines of length 2 and 3.

A four-state solution can not be hoped for : J.B.Yunès [67] destroyed this hope, if ever we had it, by systematically exploring possibilities through a backtracking program.

Does there exist a five-state solution ? Till now this is an open problem. However, there are 5-state solutions for particular families of lengthes, for instance for lines of length 2^k for some k (J.MAZOYER, unpublished)

1.6 Minimal time solutions for the 2e-FSSP

The idea which comes naturally is that of developing any one of the minimal time solutions for the FSSP together with the symmetrical solution, thus obtaining an s.t.d formed by two halves, each being the s.t.d of a half line with one general, the symmetry axis working as border line for each (Figure 17).

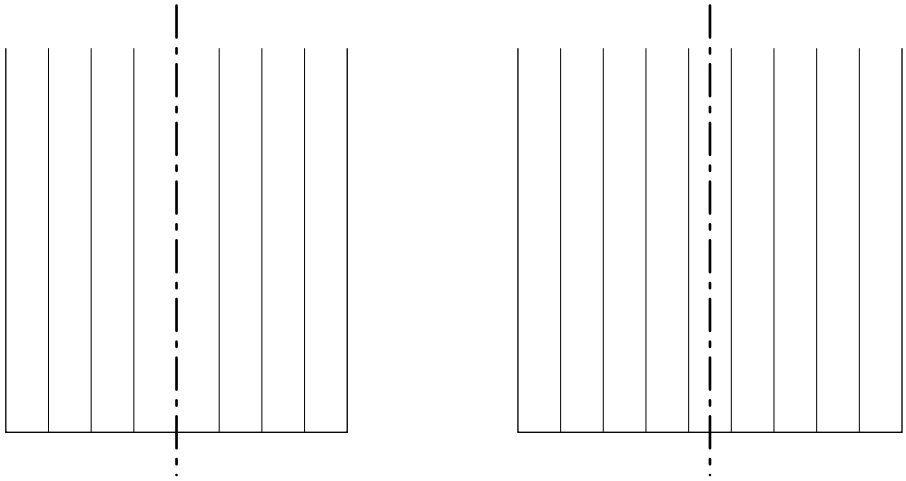


Fig. 17

Only the fitting together of these two halves will be somehow difficult, and we shall need for this a little skill in pottering with states. Two lemmas will help us, quite obvious but better clearly and distinctly expressed, and we shall also need c.a products. So let us prepare our tools.

1.6.1 C.a products

Let A_1 and A_2 be two linear c.a's with the same number n of cells, having sets of states and transition functions :

$$(Q_1, \delta_1) \quad \text{and} \quad (Q_2, \delta_2).$$

Their product is the c.a having set of states $Q_1 \times Q_2$ and as transition function the product of δ_1 and δ_2 :

$$\delta[(g_1, g_2), (q_1, q_2), (d_1, d_2)] = (\delta_1(g_1, q_1, d_1), \delta_2(g_2, q_2, d_2)).$$

It simply results from the simultaneous and parallel running of the two c.a's. We can draw its s.t.d by placing side by side, or on top of one another, the sites of the two s.t.d's. But why not, simpler still, keep the two s.t.d's side by side but separate ! Note that we can make the product of more than two c.a's.

If the resulting c.a's are often very complex, the product operation in itself is very easy. It is particularly useful and we shall use it extensively.

1.6.2 Merging and splitting of states

Technical lemma 1.6.1 (Merging of states) *It is possible to make one state out of two (or more), if there does not exist triples of states (eventually appearing in the s.t.d) that would thereby merge without their images by δ merging.*

For instance if there exist states q and q' such that $\delta(q_1, q, q') = p_1$ and $\delta(q_2, q, q') = p_2 \neq p_1$ we cannot merge q_1 and q_2. On the contrary such rules as

$$\delta(q_1, q, q') = p_1 \quad \text{and} \quad \delta(q_2, q, q') = p_1$$

or

$$\delta(q_1, q, q') = q_1 \quad \text{and} \quad \delta(q_2, q, q') = q_2$$

cannot hinder merging q_1 and q_2.

Technical lemma 1.6.2 (Splitting of states) *It is always possible to split a state in two (or more), we must just define transition for the new triples, in accordance with the original rules.*

For instance, if $\delta(p, q, q) = r$ and q is split into q_1 and q_2, we must define

$$\delta(p, q_i, r) = r \quad \text{for } i, j = 1, 2.$$

If $\delta(p, q, p) = q$ and we split q into q_1 and q_2, we must choose $\delta(p, q_1, p)$ and $\delta(p, q_2, p)$ in $\{q_1, q_2\}$.

1.6.3 First steps towards a minimal time solution to the 2e-FSSP

Our starting point is (Q, δ) a solution for the FSSP in minimal time $T_{1e}(n) = 2n - 2$ (and $T_{1e}(1) = 1$). We shall denote Δ the transition function of the solution we are trying to build.

Splitting e and G in original and newly appeared states

Out of state e of time 0 we create two quiescent states, e, e' as follows. We modify δ so that every transition $(p, q, r) \mapsto e$ with $(p, q, r) \neq (e, e, e)$ is replaced by transition $(p, q, r) \mapsto e'$. However, transition $(e, e, e) \mapsto e$ is not modified (Figure 16).

We also split state G into state G of time 0 and another state G' at all times different from 0. These precautions will spare us changing the initial configuration in the future.

Splitting the other states in left and right states

This we do for all states q of Q, except e, G and $*$:

$$q \quad \text{gives} \quad q_l \quad \text{and} \quad q_r.$$

$$(e' \quad \text{gives} \quad e_l \quad \text{and} \quad e_r).$$

$$(G' \quad \text{gives} \quad G_l \quad \text{and} \quad G_r).$$

Transition for the left triples will be δ, and for the right ones the symmetrical rule

$$\Delta(g_l, q_l, d_l) = \delta(g, q, d)_l \qquad \Delta(g_r, q_r, d_r) = \delta(d, q, g)_r$$

$$\Delta(\beta, q_l, d_l) = \delta(\beta, q, d)_l \qquad \Delta(g_r, q_r, \beta,) = \delta(\beta, , q, g)_r$$

We postpone defining Δ for mixed triples.

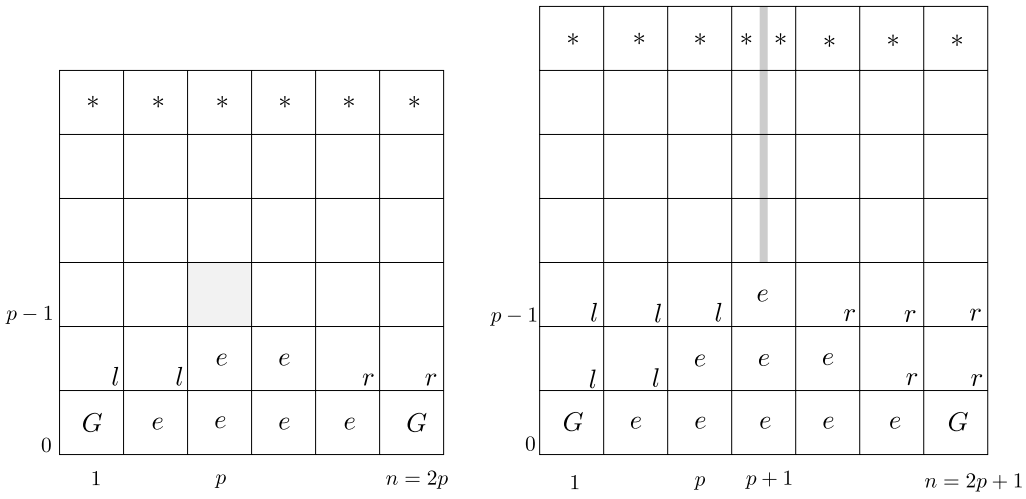


Fig.18

To obtain the s.t.d of (Q, δ) from the left general up to time $\lfloor n/2 \rfloor - 1$, and the symmetrical s.t.d from the right general, we introduce transition rules

$$\Delta(\beta, G, e) = \delta(\beta, G, e)_l \qquad \Delta(e, G, \beta) = \delta(\beta, G, e)_r$$

$$\Delta(G, e, e) = \delta(G, e, e)_l \qquad \Delta(e, e, G) = \delta(G, e, e)_r$$

and

$$\Delta(g_l, q_l, e) = \delta(g, q, e)_l \qquad \Delta(e, q_r, d_r) = \delta(d, q, e)_r$$

$$\Delta(g_l, e, e) = \delta(g, e, e)_l \qquad \Delta(e, e, d_r) = \delta(d, e, e)_r$$

Now what will happen in the middle ?

In case the c.a has odd length $n = 2p + 1$

clearly, at time $p-1$ the r-states can play the part that border states played for line L_{p+1}. Likewise l-states for the half line $[p+1, n]$. This leads us to produce at time p two states

$$\delta(q, e, \beta)_l, \delta(q, e, \beta)_r$$

that we shall join in a double state, so that we have

$$\Delta(q_l, e, q_r) = (\delta(q, e, \beta)_l, \delta(q, e, \beta)_r)$$

and, for the case $p = 1$, $n = 3$

$$\Delta(G, e, G) = (\delta(G, e, \beta)_l, \delta(G, e, \beta)_r).$$

From time p up, the double states on central cell will behave as if they were carried by two distinct cells. The s.t.d looks quite like the s.t.d of a line of length $p + 1$ with its symetrical image in a mirror (Figure 18). The two halves synchronize at time

$$T_{1e}(p + 1) = 2(p + 1) - 2 = 2p = n - 1.$$

In case the c.a has even length $n = 2p$**, with** $p > 1$

let us compare what happens in this line with what happens in the half-line (with only one general G) of length p : in the latter, the right border has already influenced state of site $(p, p-1)$ because

$$< p, p-1 >= \delta(< p-1, p-2 >, e, \beta)$$

whereas state $< p, p-1 >$ of our double line is (Figure 18)

$$\Delta(< p-1, p-2 >_l, e, e) = \delta(< p-1, p-2 >, e, e)_l.$$

Only at time $p-1$ does cell p know it is the last one of the half-line, whereas if there were a real border it would know this from time 0. For this reason we shall delay the starting of (Q, δ) by one unit of time (for instance we decide, as in Figure 19, that G becomes H and H really acts as general), without delaying the l and r marking ! The r/l-states acting as borders for the left/right configurations we then obtain exactly the s.t.d of L_p with the symmetrical s.t.d.

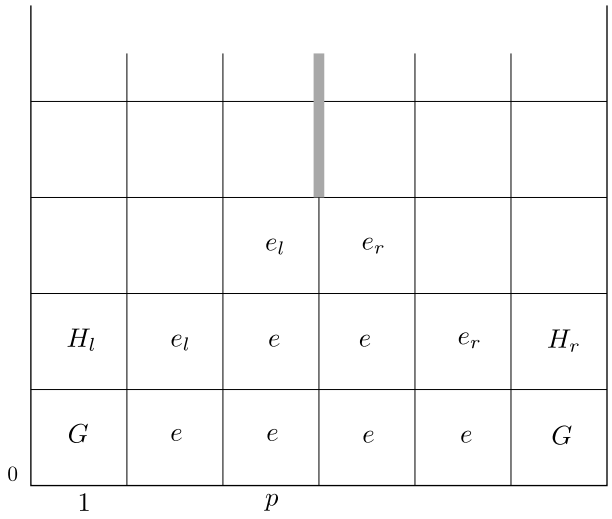


Fig. 19

Synchronizations so appear at time

$$1 + T_{1e}(p) = 1 + 2p - 2 = 2p - 1 = n - 1$$

Now here we are with two solutions, both incomplete, the first one will fail on the first *ggd* or *gdd* triples in case the line has even length and the second one on the first (q_l, e, q_r) triple in case the line has odd length. Of course, the generals cannot choose at time 0 because they do not know the number of the soldiers, except in case $n = 2$, $p = 1$, for which we decide

$$\Delta(\beta, G, G) = \Delta(G, G, \beta) = *.$$

1.6.4 Combining the two solutions

A simple manner of managing the above problem is to run the two solutions simultaneously, so as to be sure to have the good one at hand when the middle cells become active and we can drop the bad one. This is precisely a product of the two c.a's. In Figure 20, sites of the even solution are represented above sites of the odd solution.

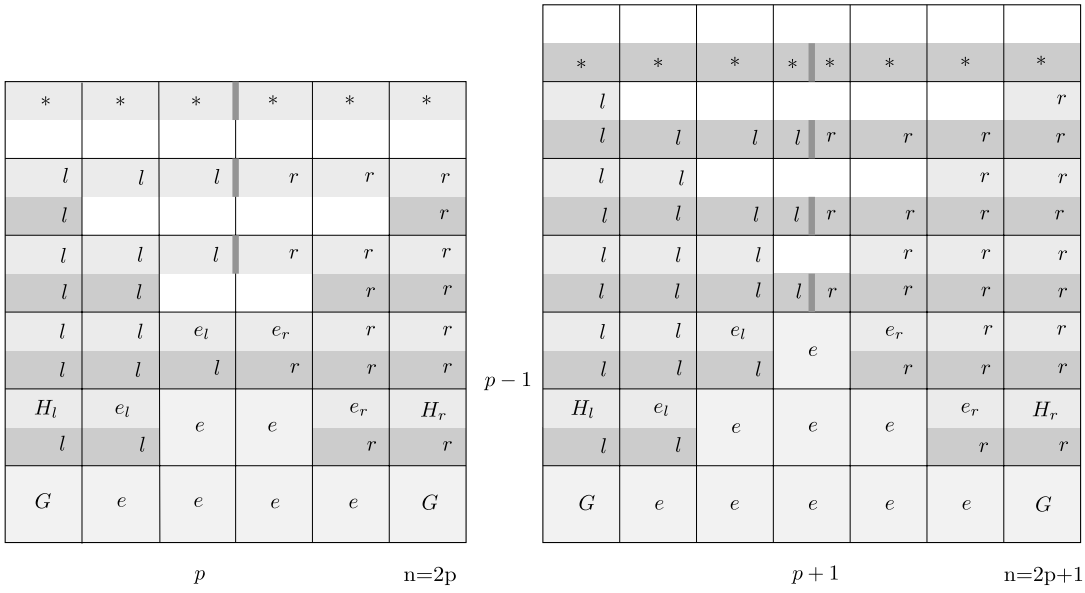


Fig.20

In the present case the product must however be arranged a little and completed :

- first it appears only at time 1, (and not for state e), so as to preserve initial configuration

- then, for certain triples transition is not defined (above in the odd case, below in the even case) : on the figure blanks are to be seen, that we can declare as some sort of breakdown state, which will be a permanent state, and an infectious state also (that is, if a triple of states has a breakdown in it , its image is a breakdown state)

- lastly, we identify all couples in which at least one of the states is the fire state, and this will be the fire state of our final c.a.

Now the synchronizing time of this c.a is $n-1$, the minimal time, for every n, $n=2$ included

$$T_{2e}(n) = n - 1.$$

1.6.5 General solutions by symmetry for the 2e-FSSP

If we apply the same construction to any 1e-synchronizer, in time T_{1e} not necessarily minimal , we obtain a 2e-synchronizer in time

$$\begin{array}{lll} T_{2e}(n=2p+1) = T_{1e}(p+1) & \text{for} & p \geq 1 \\ T_{2e}(n=2p) = T_{1e}(p)+1 & \text{for} & p > 1 \\ T_{2e}(2) = 1 \text{ (or any value we choose).} & & \end{array}$$

This synchronizing time is roughly the 1e-time of half the lines, which of course is not necessarily half the time of the lines.

1.7 The F.S.S.P with general anywhere in the line

Here too we shall make use of any minimal time solution of the FSSP, but this time on either side of the general. As in the preceding section, we begin by differentiating the initial quiescent state e then splitting the states and rules in two, with the same l and r notations.

Initial configuration is (see Figure 21)

$$\overbrace{(\overbrace{e,\dots,e}^{n_1},\underbrace{G,e,\dots\dots,e}_{n_2})}$$

where n_1 and n_2 are the lengths of the left and right half lines with the general included, so

$$n = n_1 + n_2 - 1$$

The idea we have is very simple : we shall delay the synchronization processes of the left and right half lines until respective times $2n_2$ and $2n_1$ so they should end at the same time. How can we organize this ?

State l having two quiescent neighbours sends two signals, L leftwards and R rightwards, which will reflect on the borders and come back on the headquarter cell at times $2n_2 - 2$ and $2n_1 - 2$ respectively. It also becomes at time 1 a triple state that we denote $G\beta G$. The idea with this triple state is that it will behave as would two cells separated by some borders. The l-component states are sleeping generals, the right one will wake up when receiving signal R and become $G_r G_l G_l$, the left one will wake up when receiving signal L and will become G_l. Two independant synchronizing processes take place, in which of course signals L and R do not interfere, they just juxtapose in product states. It is now left for us to identify state $*\beta*$ with the fire state $*$, and the entire line is synchronized, at time

$$2n_1 - 2 + 2n_2 - 2 = 2n - 2.$$

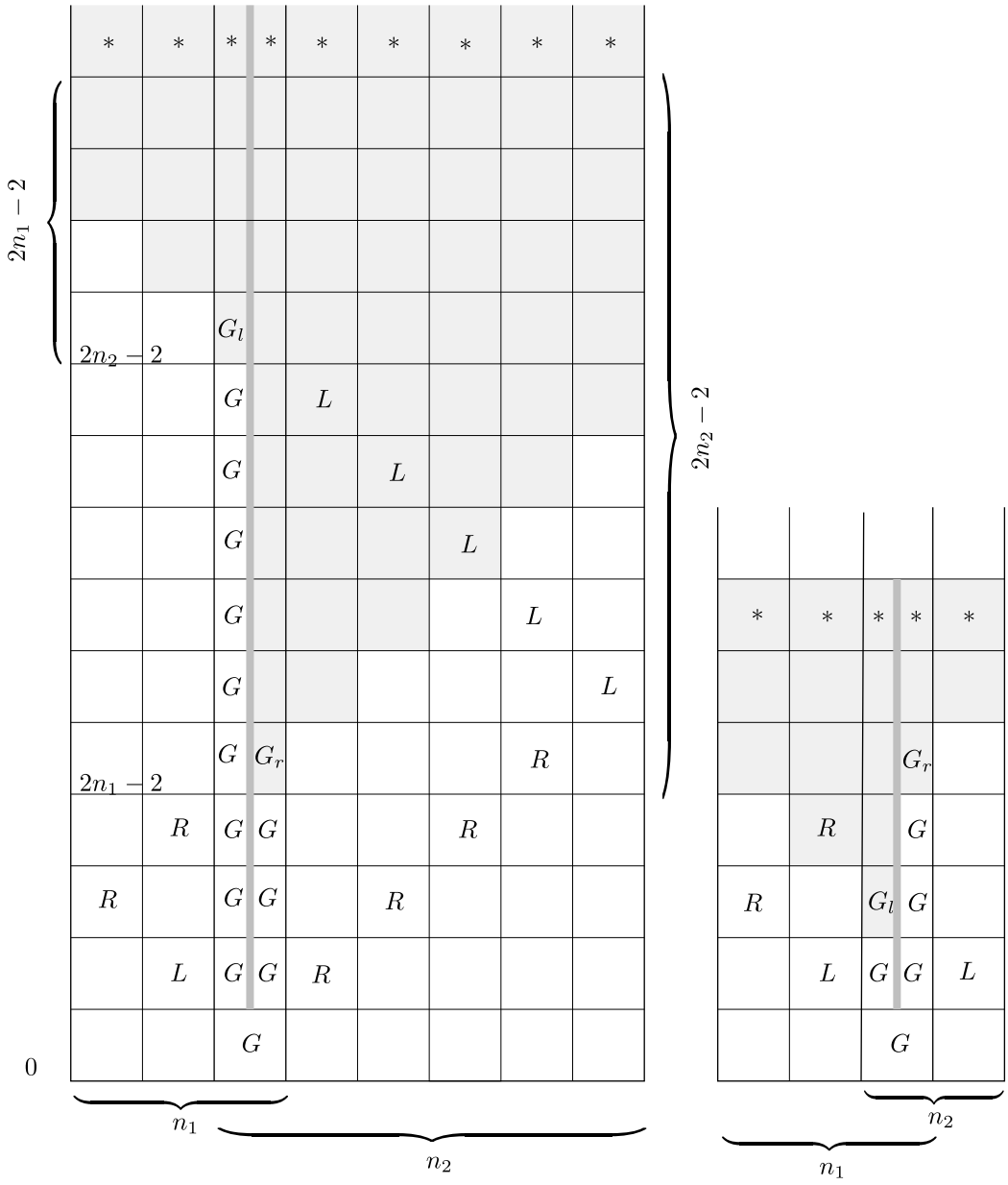


Fig.21

Let us not forget a few particular cases : if l is at the left end, it will directly act as G_r, if it is at the right end it will act as G_l, and for the case when we have just one cell

$$\Delta(\beta, G, \beta) = *.$$

synchronizing time is then

$$T(n) = max(1, 2n - 2).$$

The solution for the synchronization problem for the family of finite lines with general anywhere on the line that we have just given has a synchronizing time which depends only on the length of the line. This time is the best possible for such a solution, as indeed it is the least possible in case the general is at one of the two ends. Clearly a lesser synchronizing time can be obtained in other situations : for instance if the general is at the middle of an odd line of length n we could synchronize in $n - 1$ steps. Should we add as second parameter the position p of the general, existence of a better solution is not excluded.

REFERENCES

Finite automata are extensively studied in a wide range of works, out of which we shall mention :

[14, EILENBERGSamuel]
[20, GINZBURG Abraham]
[52, STERN Jacques]

and more briefly in

[27, HOPCROFT-ULLMAN].

Sequential machines are studied in

[19, GINSBURG Seymour].

The first minimal time solution for the FSSP is to be found in

[21, GOTO Eiichi].

In fact, this technical report seems quite impossible to find, but the solution was rewritten by Umeo in

[56, UMEO Hiroshi].

A systematic rehearsal of all possible four state solutions for the FSSP is presented in

[67, YUNES Jean-Baptiste].

Chapter 2

Mazoyer's minimal time solution for the F.S.S.P

As mentioned in chapter 1, it is, among all known minimal time solutions, the one with the lesser number of states.

Let us remind that the minimal time is $T(n) = max(1, 2n-2)$.

2.1 Its general principle

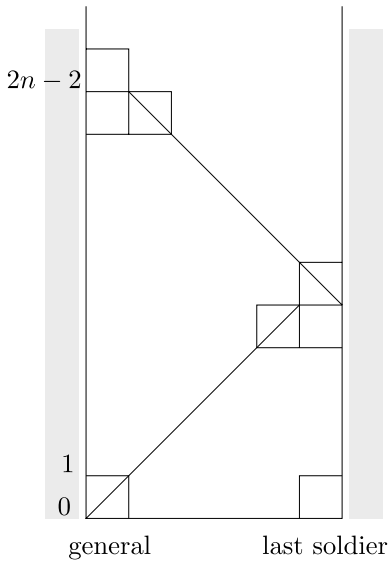


Fig. 1

Time $2n-2$ is exactly the time necessary for the general, who ignores the length n of the line, to call the last soldier and get his response back, call and response being transmitted at the maximal speed of one cell per unit of time (Figure 1).

All the solutions (since Goto [21]) proceed through an induction on the length of the lines, which successively split in smaller and smaller lines. They differ by the way this splitting process is set up.

In Mazoyer's solution the general generates (or so it seems) a bundle of rays which will determine on the response signal a sequence of new generals, each of them starting to command the part of the line at his right (Figure 2). On each sub-line a similar process then takes place iteratively.

The process ends by the firing of all the smallest lines, (and of the small part of the line which remains on the left).

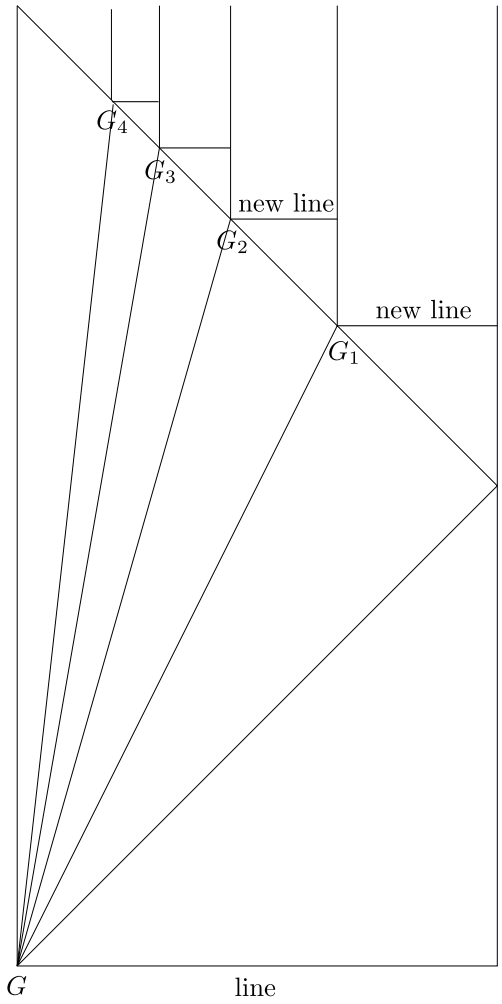


Fig.2

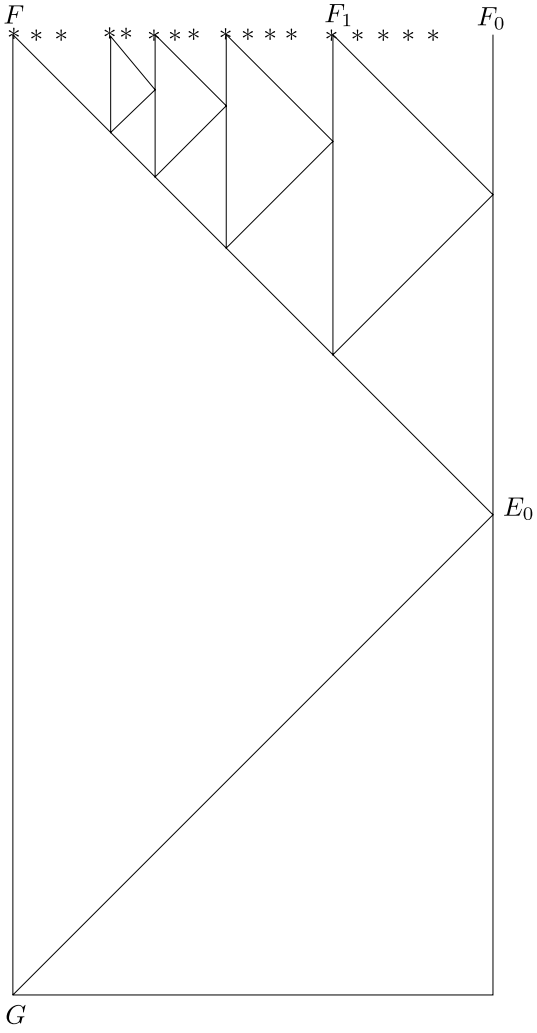


Fig.3

2.2 Position and delay for the new generals

Splitting of the line must occur at the right time for all sub-lines to fire at time $2n-2$ (Figure 3).

Let us forget for a while the rigidity of the discrete world, and solve a little geometric problem illustrated on Figure 4 : how to choose G_1 so that F_1's second coordinate be that of F ? A moment reflexion gives the solution :

$$E_0G_1 = \frac{1}{3}E_0F.$$

Similarly we must then have

$$G_1G_2 = \frac{1}{3}G_1F$$

$$\ldots$$

$$G_iG_{i+1} = \frac{1}{3}G_iF$$

$$\ldots.$$

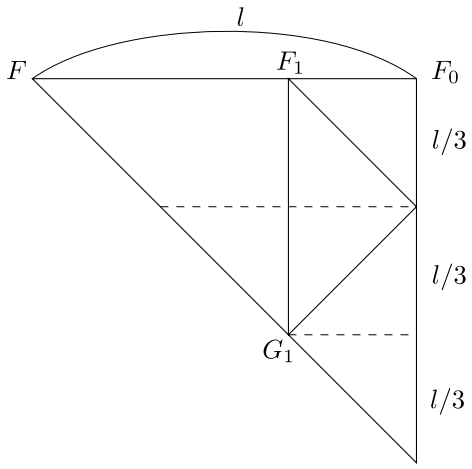


Fig.4

Let us now return to our line of cells. Let n_1 be the first cell which must become a general in receiving the response signal. Coordinates of the sites of interest being indicated on Figure 5, we want to have

$$4n - 3n_1 - 1 = 2n - 2$$

$$3n_1 = 2n + 1.$$

n and n_1 being integers, this equation may have no solution. We thus decide that the new general on n_1 will wait some delay d_1 before commanding, so that our new equation becomes

$$4n - 3n_1 - 1 + d_1 = 2n - 2$$

$$3n_1 = 2n + 1 + d_1.$$

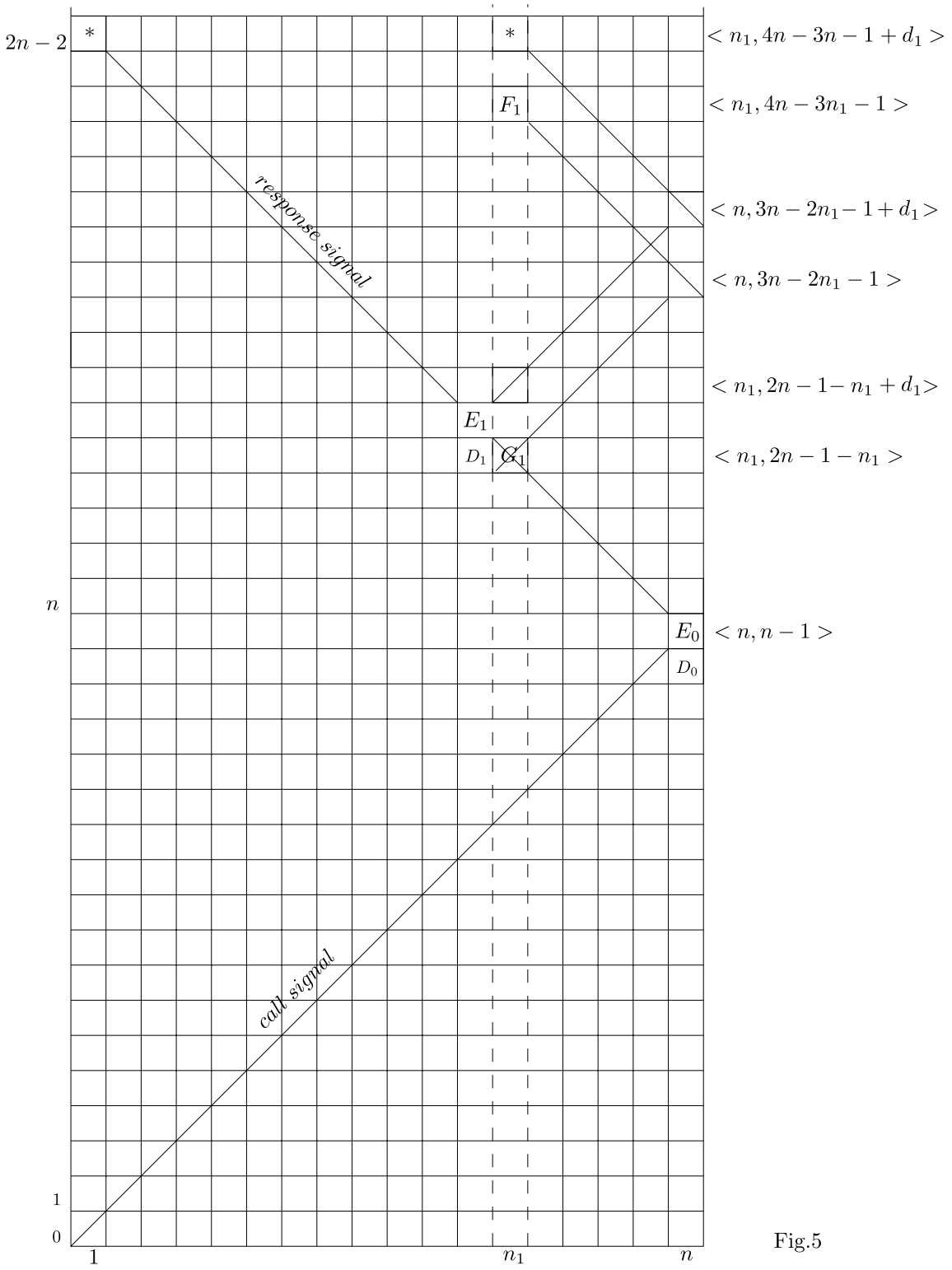

$2n-2$
$*$
$*$
$< n_1, 4n-3n-1+d_1 >$
F_1
$< n_1, 4n-3n_1-1 >$
response signal
$< n, 3n-2n_1-1+d_1 >$
$< n, 3n-2n_1-1 >$
$< n_1, 2n-1-n_1+d_1 >$
E_1
D_1
G_1
$< n_1, 2n-1-n_1 >$
n
E_0
$< n, n-1 >$
D_0
call signal
1
0
1
n_1
n

Fig.5

d_1 will be chosen equal to 0, 1 or 2 according to the value of n, so that the second member should be a multiple of 3. If

$$n = 3p + i \quad \text{with} \quad i = 1, 2 \text{ or } 3$$

we get

$$d_1 = i - 1 \qquad \text{and} \qquad n_1 = 2p + i. \tag{2.1}$$

So the first new general appears on site

$$< n_1 = 2p + i \ , \ t_1 = 2n - 1 - n_1 = 4p + i - 1 = n + p - 1) >$$

and is activated with delay $d_1 = i - 1$ ($d_1 = 0$, 1 or 2). Length of the new line so created is

$$n - n_1 + 1 = p + 1$$

and the length of the portion of the original line remaining on the lefthand side is

$$n_1 - 1 = 2p + i - 1.$$

At this point we can ensure that, if line of length $n - n_1 + 1$ synchronizes in minimal time $2(n - n_1 + 1) - 2 = 2n - 2n_1$, the right part of the initial line synchronizes at time

$$(2n - 1 - n_1 + d) \ + \ (2n - 2n_1) = 4n - 3n_1 + d - 1 = 2n - 2.$$

We shall find the next general, G_2, by the same calculations where E_1 replaces E_0 and $n_1 - 1$ replaces n. So, if

$$n_1 - 1 = 3p_1 + i_1 \quad \text{with} \quad i_1 = 1, 2 \text{ or } 3$$

we have

$$n_2 = 2p_1 + i_1 \qquad d_2 = i_1 - 1$$

where d_2 is the delay for activation of G_2. The next right portion of line begins with this new general and ends just before the first new general. And so on. Until when ?

2.3 End of the splitting process

Until lines are too small to be split, be it new lines or the remaining portion of line on the left.

Figure 6 shows the timing constraints for the very small lines.

If we apply the preceding formulas for $n = 2$, or $n = 3$ (see Figure 6), the right part of the line, which is reduced to the only last cell, will not synchronize at the right time. On the contrary, for $n = 4$ everything works out smoothly, the new general is on cell 3, he acts with no delay, the new line on the righthand side has length 2, and if it synchronizes as a line of length 2 should, this will be at the right time for the original line of length 4.

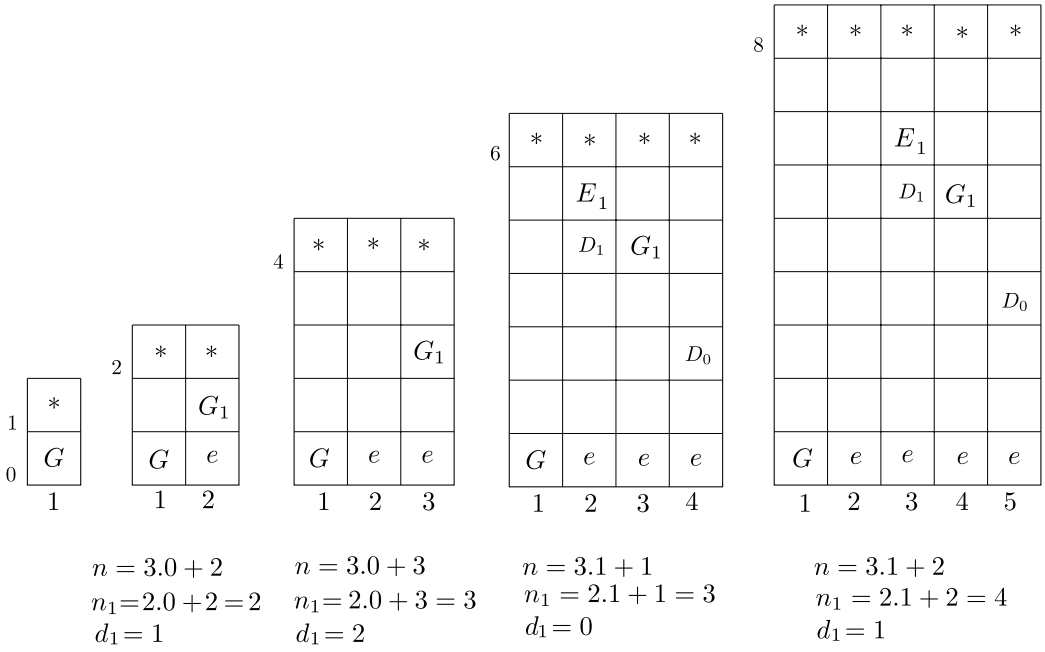


Fig.6

So we decide that we shall split only lines of length

$$n \geq 4.$$

If $n = 3p + i$, with $i = 1, 2$ or 3, then condition $n \geq 4$ amounts to $p \geq 1$. So the lengthes of the new line and the remaining line are respectively

$$p + 1 \geq 2$$

$$2p + i - 1 \geq 2p \geq 2.$$

So all the splittings will result in lines of length 2 or 3.

At this point we can guarantee that, if lines of length 2 and 3 do synchronize in time 2 and 4, then the entire line, except perhaps the last remaining portion (of length 2 or 3) on the left of the line itself and of all created sublines, will indeed synchronize in minimal time.

2.4 First waves

Before we go ahead, let us make a halt to introduce what we shall call waves, by giving an example which will help understand the solution. A more systematic study will be done in chapter 4.

2.4.1 Moving a state by pelting it with signals

Let A be a c.a with, next to the quiescent state, a stable state S, and states A_1, A_2, ... , A_q, which represent left signals of speed 1 (we do not write down the corresponding rules). These signals are generated on the left of the quiescent area by some state A_1 and rules

$$\begin{array}{ccc} & A_2 & \\ A_1 & e & e \end{array} \qquad \begin{array}{ccc} & A_3 & \\ A_2 & e & e \end{array} \qquad \begin{array}{c} \dots \\ \dots \end{array} \qquad \begin{array}{ccc} & A_1 & \\ A_q & e & e \end{array}$$

To make s.t.d's more clear, in figures we just indicate the indexes of the A-signals. In Figure 7 $q = 3$, we have 3 A-signals.

Now, contradicting the perfect stability of state S, we suppose that for certain indexes, for $i \epsilon I \subset \{1, \dots, n\}$, S is pulled to the right by A_i, that is we have rules

$$\begin{array}{ccc} & e & \\ e & S & A_i \end{array} \qquad \begin{array}{ccc} & S & \\ S & A_i & e \end{array} \qquad \text{that is} \qquad \begin{array}{cccc} & e & S & \\ e & S & A_i & e \end{array}$$

In Figure 7, $I = \{1, 2\}$, so A_1 and A_2 pull S to the right, while A_3 leaves it stable.

Let us observe the evolution of this c.a if it starts in a configuration

$$e, \dots, e, S, e, \dots, e, A_1, e, e, \dots .$$

State S appears to submit to the action of the A_i signals, instead of moving by its proper strength, as did the first signals encountered in chapter 1. For this reason, and thinking of how waves in the sea are moved by the wind, we choose to call it a wave.

It is easy to observe that, if I has p elements (i.e p out of q A-signals pull S to the right), then, q right moves of the diagonal generate one sequence $A_1, A_2, \dots, A_q$, which in turn determines p right moves of S and $q - p$ stand stills, so that finally, from the hitting site of A_1 to the hitting site of A_q, S goes p cells further in $2q - p$ time-steps, so the speed of the wave is $p/2q - p$.

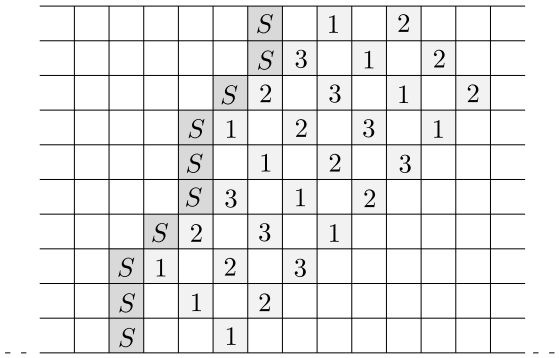


Fig.7

2.4.2 A variation

We shall now slightly change the rules. First we add rules

$$\begin{array}{ccc} & A_i & \\ ? & A_i & e \end{array}$$

where ? is any state. With this rule the area between the quiescent area and the S wave is completely filled with the A-signals.

Now the new rules for the right moves for the i indexes in I will be

$$\begin{array}{ccc} & S & \\ S & A_i & A_{i+1} \end{array}$$

For the other j indexes A_jA_{j+1} leaves S unmoved, as well as all other A_iA_j or A_iA_i couples (this precision for later use in more complex s.t.d's).

Lastly, we make our S wave a bit thicker by deciding that state S is stable except when it has another S at its right. Rules are thus

$$\begin{array}{ccc} & S & \\ e & S & A \end{array} \qquad\qquad \begin{array}{ccc} & e & \\ e & S & S \end{array}$$

Starting from the same configuration, our new s.t.d is the one of Figure 8.

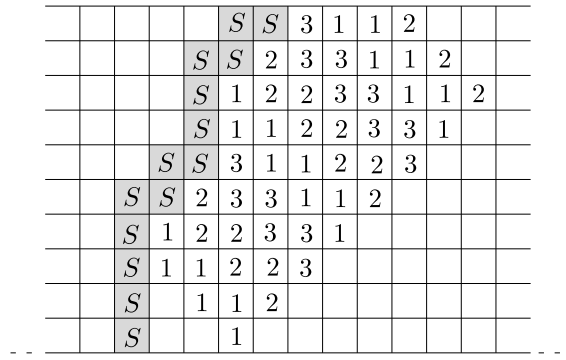


Fig.8

2.4.3 Iterated waves

We shall now suppose that at the left of S in the starting configuration we have state A_1, and that we have the same rules of conservation and regeneration of the A-signals by state S as we had by state e, that is

$$\begin{array}{ccc} & A_i & \\ ? & A_i & S \end{array}$$

and

$$\begin{array}{ccccccccccc} & A_2 & & & A_3 & & & \dots & & & A_1 & \\ A_1 & S & S & A_2 & S & S & & \dots & A_q & S & S \end{array}$$

Then, at the left of the S wave we have a new wind of A-signals. If we have another S state at the left in the starting configuration, then these A signals will act on this state and form a new S wave. This process can be repeated. Figure 9 gives an illustration

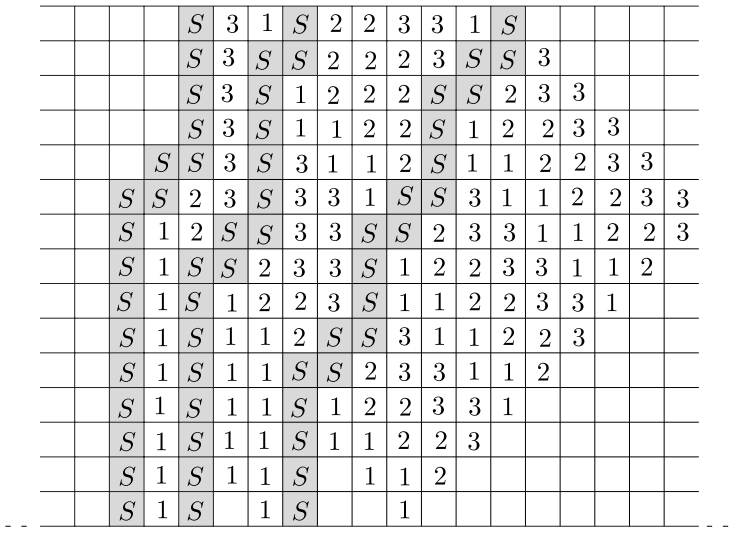


Fig.9

We may observe that the S-waves form a pattern of steps one cell broad and having various heights. Passing from one wave to the next one, $q - p$ steps out of q disappear, the successive waves progressively straighten up.

2.4.4 Recursive generation of waves

We shall now add rules which will enable the A signals to generate new waves when they reach the left border (denoted β), and also A states at the left of these waves :

$$\begin{array}{ccc} & S & \\ \beta & ? & A_1 \end{array} \qquad\qquad \begin{array}{ccc} & A_q & \\ \beta & S & S \end{array}$$

For $q = 3$, starting with initial configuration $A_3, e, e, \ldots$, we obtain the rippled s.t.d of Figure 10.

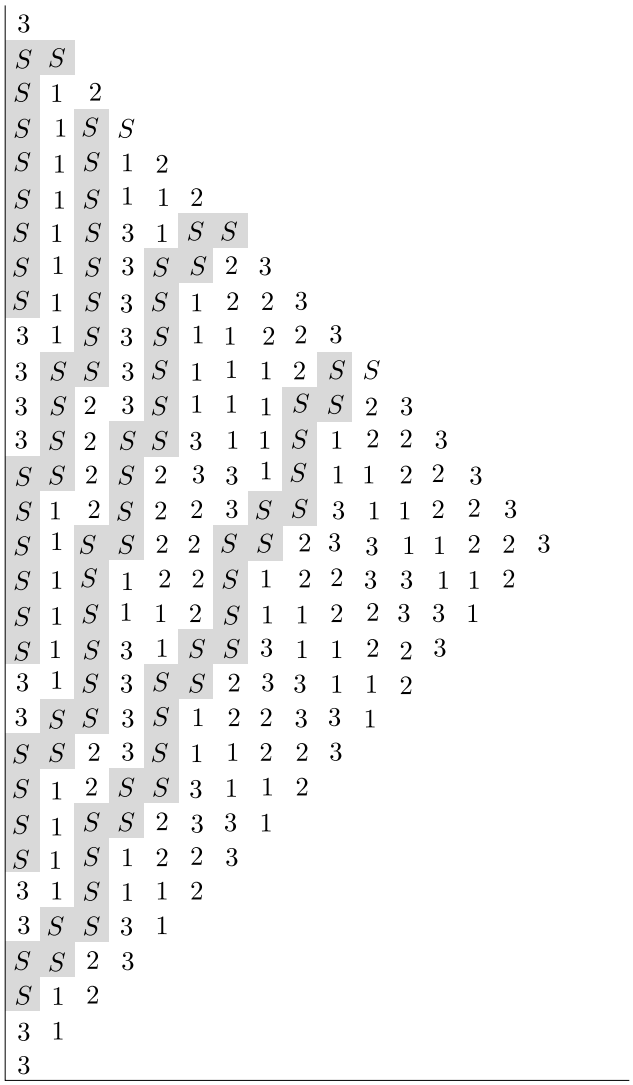


Fig.10

Let us now replace state S by state e, and add a new stable state G on the left border, which will do the generation of the new waves and A_q-states that the border previously did. For $q = 3$ we obtain the s.t.d of Figure 11. In this figure we have coloured the border of the quiescent area like the other waves, thus pointing out that it may now be considered the first wave.

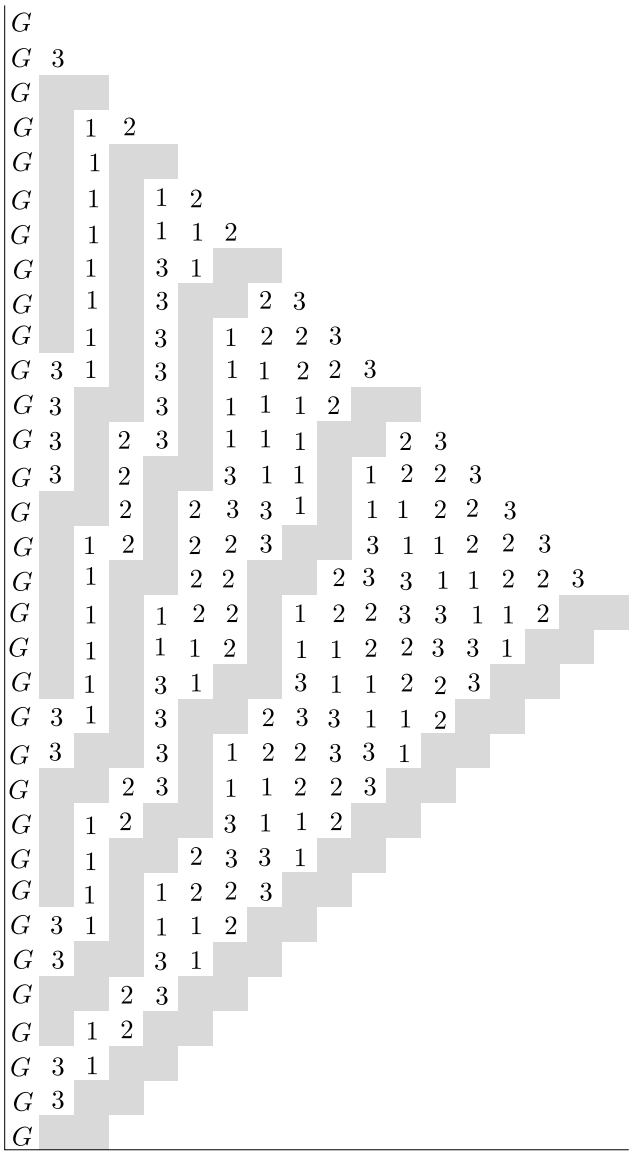


Fig.11

2.5 The bundle of splitting rays

Figure 2 suggests that the sequence of new generals should be determined on the response signal (equation $x + t = 2n - 1$) by rays sent by G. At the present stage of analysis of the s.t.d to be realized, we are in fact proceeding the other way round : having determined where the new generals should be positioned

we shall determine the rays, Σ_1, Σ_2, ... , Σ_k, ... as the locuses of the sites G_1, G_2, ..., G_k, ... when the length of the line varies. In fact to sites G_k we shall prefer sites D_k , their left neighbours. For all lengthes, these are situated on the anti-diagonal (of slope -1) just under the response signal, the one coming out of D_0, the site under E_0 (see Figures 5 and 12).

$G_1 =< n_1, t_1 >$ is obtained from $E_0 =< n = 3p + i, t >$ as follows

$$n_1 = 2p + i = n - p \qquad t_1 = t + p \qquad \text{and delay } d_1 = i - 1$$

So D_1 is obtained from $D_0 =< n = 3p + i, \tau >$ (where $\tau = t - 1$) by

$$D_1 =< n - (p + 1), \tau + (p + 1) > \qquad \text{delay for } G_1 : d = i - 1.$$

And generally D_{k+1} will be obtained from $D_k =< n_k = 3p_k + i_k, \tau_k >$ by

$$D_{k+1} =< n_k - (p_k + 1), \tau_k + (p_k + 1) > \qquad \text{delay for } G_{k+1} : d_{k+1} = i_k - 1.$$

Let Σ_0 be the locus of sites D_0, which is the diagonal starting from site $< 2, 0 >$. The above formula about D_{k+1} allows to successively construct Σ_1, Σ_2, ... , Σ_k, from the initial Σ_0 ray (Figure 13).

From the locus of sites D_0, which is the diagonal starting from site $< 2, 0 >$, we shall by this way be able to successively construct the locuses of D_1, D_2, ..., D_k,... (Figure 13).These locuses will each have one site on each return diagonal from sites D_0, that is one diagonal out of two.

We go from one site D_0 to the next (corresponding to a line having one cell more) by a move of $\vec{d} = (1, 1)$ (Figure 13) :

$$D_0(n + 1) = D_0(n) + (1, 1).$$

According to wether these two sites correspond to the same p or to successive p's, we shall pass from one site D_1 to the next by the same move $\vec{d}$ or by the move $2\vec{v} = (0, 2)$:

$$D_1(n + 1) = \begin{cases} D_1(n) + (1, 1) & \text{if } n \neq 3 \ (mod\ 3) \\ D_1(n) + (0, 2) & \text{if } n = 3 \ (mod\ 3) \end{cases}$$

We immediately see by induction that, for all the Σ_k, we pass from one site to the next by one of these two moves : moves $2\vec{v}$ remain moves $2\vec{v}$ (because the two sites correspond to the same p), and some of the moves $\vec{d}$ become moves $2\vec{v}$, so that as k increases the Σ_k tend to straighten up.

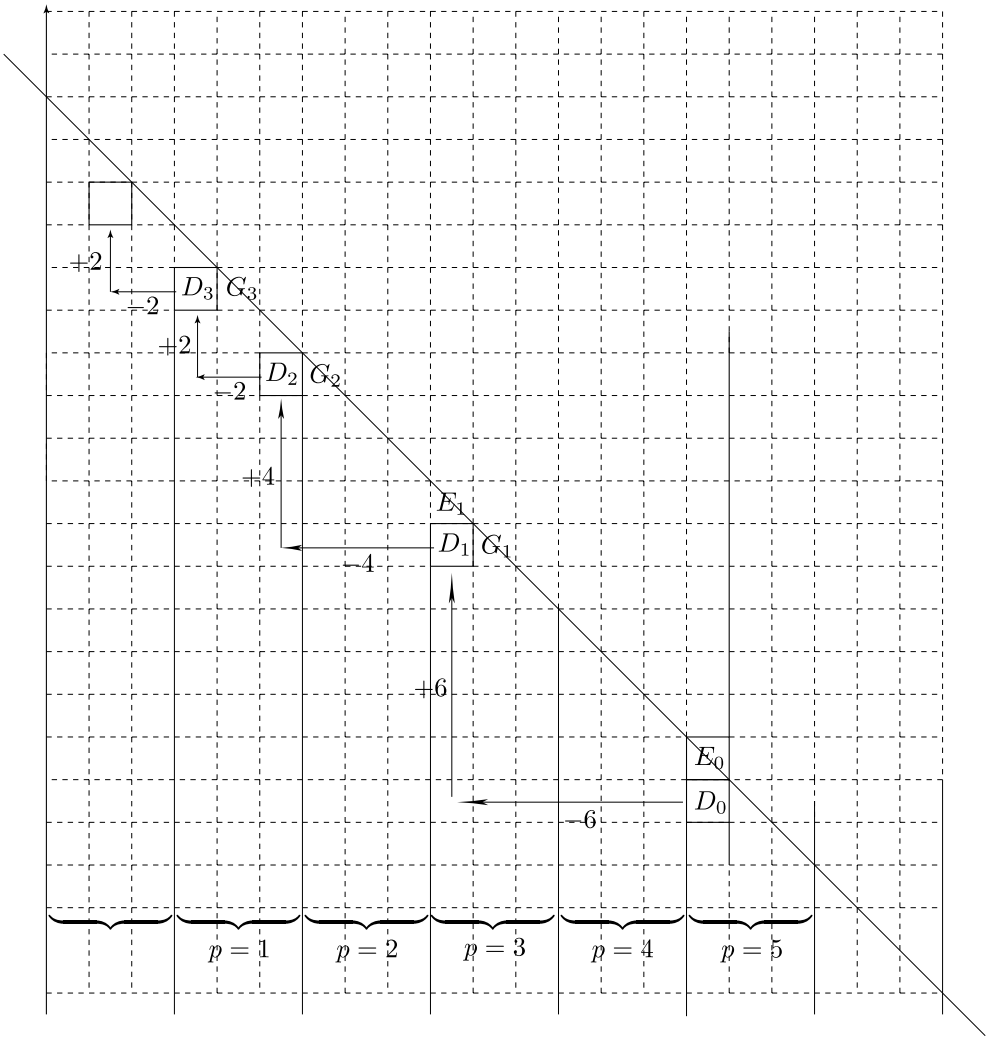
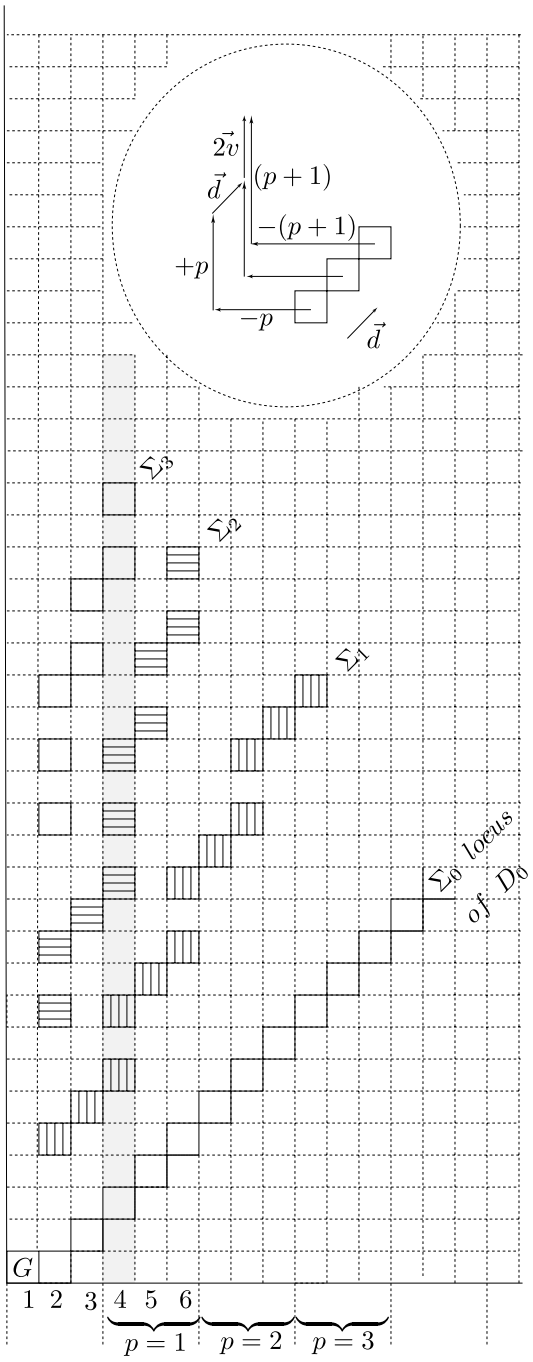
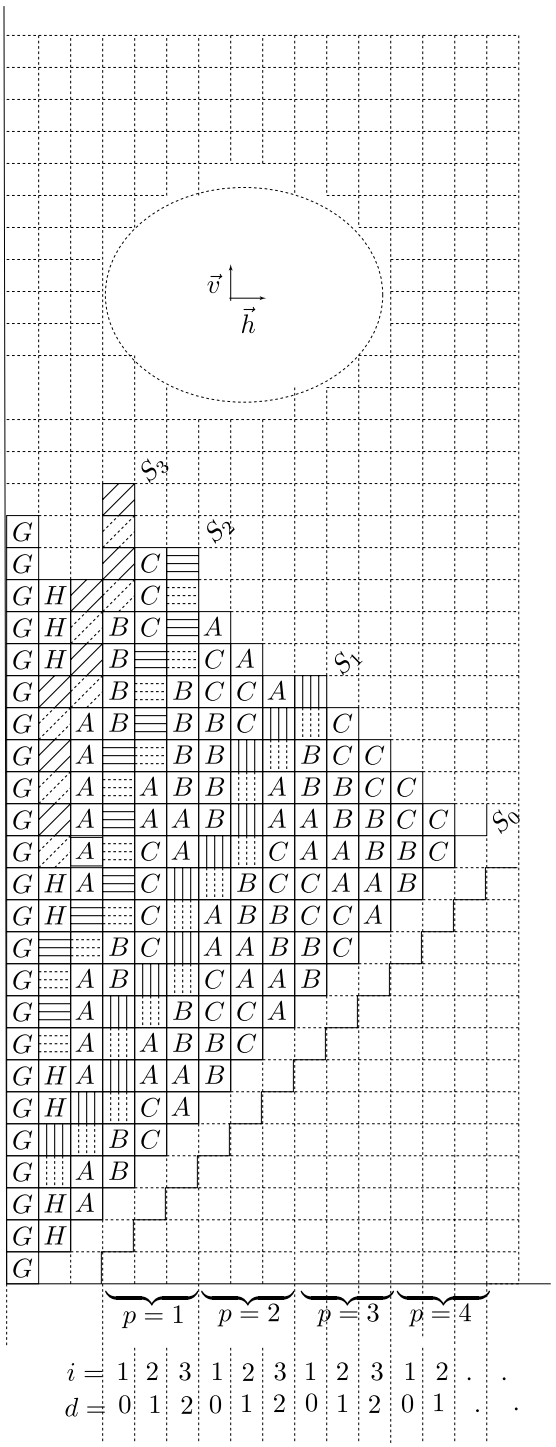


Fig.12



$n = 3p + i \quad i = 1, 2 \text{ or } 3$

Fig.13



$i =$	1	2	3	1	2	3	1	2	3	1	2	.	.
$d =$	0	1	2	0	1	2	0	1	2	0	1	.	.

Fig.14

The first site of each Σ_{k+1} comes from the first site of Σ_k on cell 4 (we don't split lines with less than four cells) : it is therefore on cell 2 (see Figure 6), and its time is even (by induction).

We shall now "fill" the Σ_k's, by adding to each of their sites the site just one time-step before, thus obtaining the S_k's of Figure 14. These S_k's have one site on each return diagonal, we go from one of their sites to the next by one of the moves $\vec{h} = (1, 0)$ or $\vec{v} = (0, 1)$. They look like stairs, the steps of which are one cell broad and have various heights.

For one $\vec{h}$ move out of three, that means one step out of three, p increases by one unit, so the next signal will straighten up.

We know from preceding section how we can implement successive S_k waves, with the help of 3 signals, denoted here A, B and C. These states must be produced on the left of each S_k, and guide the next one. The first wave S_0 is the staircase just under the diagonal, indicated on Figure 14.

We now picture the successive S_k's, created one after the other by the parallel pushing of the A, B, C signals, as the crest of waves by the blowing wind. But instead of doing cheap poetry, we had better have a precise look at how the waves start, and write all the rules down.

2.6 Rules for setting up the S_k's

A little warning first : we shall try to explain the reasons for the rules more than write them in full detail. It may happen that a next rule that we introduce contradicts a preceding one in some cases. But if the object of the rules is clearly understood, these ambiguities will be easy to resolve, and anyway a final precise table with all the rules is given at the end.

We have pointed out that a wave has no proper dynamism, and our first wave, the border of the quiescent area, locus of D_0, is made of quiescent states. As we have done in the last example of preceding section (Figure 11), we shall use the quiescent state for all the waves.

2.6.1 Generation of the pulling signals

Here we have 3 signals, that we shall denote A, B and C. We recall the rules written in preceding section.
A, B, C travel left at maximal speed, as long as they don't bump on e (the wave state)

$$\begin{array}{ccc} & A & \\ ? & \not\in & A \end{array} \qquad \begin{array}{ccc} & B & \\ ? & \not\in & B \end{array} \qquad \begin{array}{ccc} & C & \\ ? & \not\in & C \end{array}$$

where ? is any state to be found on the left (A, B, C or e) and $\not\in$ any state except e (possibly A, B or C).
They stay alongside of e

$$\begin{array}{ccc} & A & \\ ? & A & e \end{array} \qquad \begin{array}{ccc} & B & \\ ? & B & e \end{array} \qquad \begin{array}{ccc} & C & \\ ? & C & e \end{array}$$

where ? is any state to be found on the left (we shall see later that it may be H, A, B, or C).
They produce next state when e steps rightwards

$$\begin{array}{ccc} & B & \\ A & e & e \end{array} \qquad \begin{array}{ccc} & C & \\ B & e & e \end{array} \qquad \begin{array}{ccc} & A & \\ C & e & e \end{array}$$

2.6.2 Rules for the waves

e keeps steady, except in case there is a step on the right

$$\begin{array}{ccc} & e & \\ ? & e & X \end{array} \qquad \begin{array}{ccc} & \not e & \\ ? & e & e \end{array}$$

where X is A, B or C, and $\not e$ cannot be e.

e is pulled to the right by AB and BC, but not by CA :

$$\begin{array}{ccc} & e & \\ e & A & B \end{array} \qquad \begin{array}{ccc} & e & \\ e & B & C \end{array}$$

(in these cases B and C do not travel left).

2.6.3 Starting of the S_k's

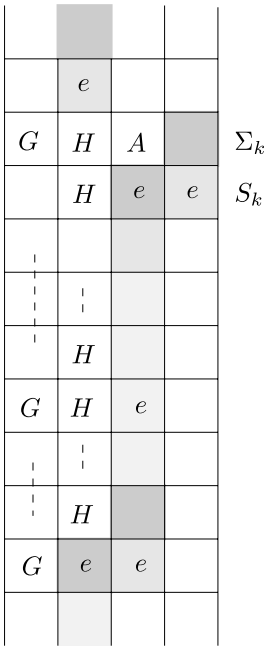


Fig.15

We know that the first sites of all Σ_k's and S_k's are on cell 2. They will be produced by state G, which is permanent (when the neighbouring states are not G's), with the help of an auxiliary state H. This state H will produce the bud for the waves, and start the A, B, C signals.

The first D_{k+1} comes from the first D_k on cell 4. The latter results from a right step of the last D_k on cell 3, and the first D_k on cell 3 results from a right step of the last D_k on cell 2. This is illustrated in Figure 15.

G will produce H when S_k leaves cell 2, H will stay alongside e as long as S_k stays on cell 3, it will produce state A and state e when S_k leaves cell 3. Rules for this are

$$\begin{array}{cccccccccccc} & H & & & H & & & A & & & e & \\ G & e & e & G & H & e & H & e & e & G & H & A \end{array}$$

With these rules, all sites D_k on cell 4 send signal A, then sites D_k on cell 5(/6) will send signal $B(/C)$, and more generally the signal under the response signal of the line carries the information on the length of the line, A if $n = 3p+1$, B if $n = 3p + 2$ or C if $n = 3p + 3$.

It is interesting to observe that the s.t.d we have now obtained is exactly the same as the one in Figure 11, except that we have state H instead of the first A_3 state. As far as the generation of our S_k waves is concerned H could be replaced by C, but other reasons will prevent us from doing this.

2.7 Creating the new lines

We must now organize the outcoming of the new generals and the start of the new lines.

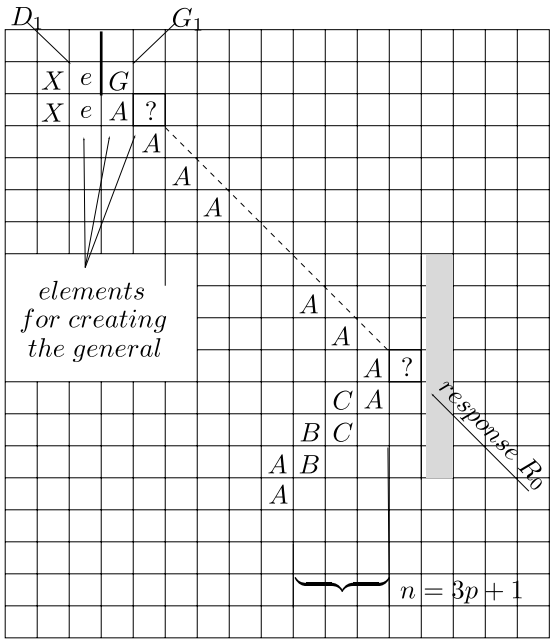


Fig.16a

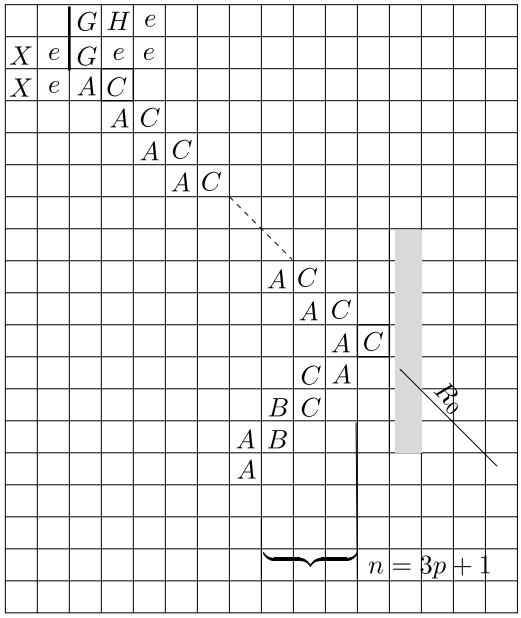
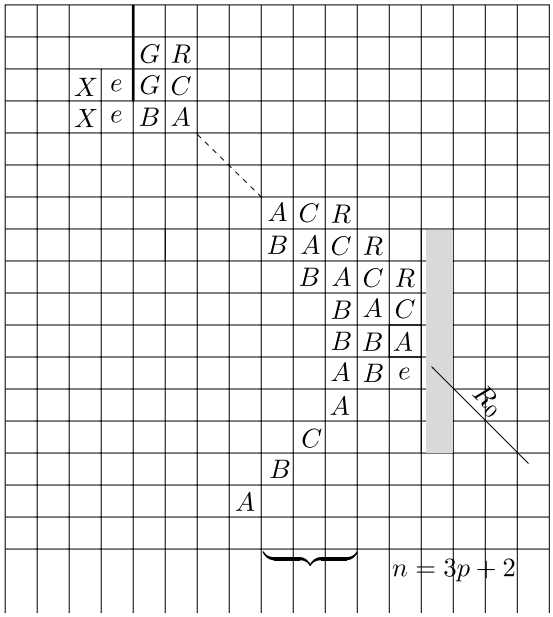


Fig. 16b

Fig.16



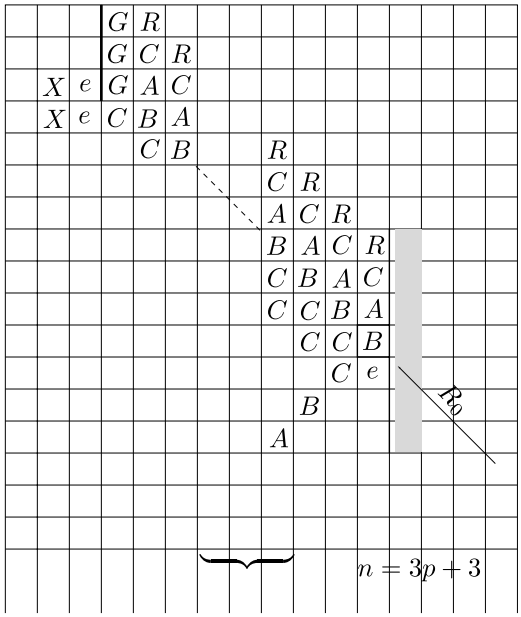


Fig.17

Let us begin with the first general G_1. The main diagonal, from which the signals start, reaches the right border in state B, C or A (Figure 16a). G_1 must appear on the return diagonal, right of site D_1 of S_1, which is quiescent. So at this time we must have all we need for creating the general : and what we shall use is, at the right of state e under D_1, two states in inverse order

$$AC,\ BA \text{ or } CB.$$

It is the approaching of the border that will reverse generation of the states on the return diagonal, with rules (Figure 16b)

$$\begin{array}{cccc} & C & & \\ A & e & \beta & \end{array} \qquad \begin{array}{cccc} & A & & \\ B & e & \beta & \end{array} \qquad \begin{array}{cccc} & B & & \\ C & e & \beta & \end{array}$$

The general will then be created by rules

$$\begin{array}{cccc} & G & & \\ e & A & C & \end{array} \qquad \begin{array}{cccc} & G & & \\ e & B & A & \end{array} \qquad \begin{array}{cccc} & G & & \\ e & C & B & \end{array}$$

(here again we have a new rule contradicting the drifting left of states A, B and C).

Now, remains the question of the delay for activation, 0,1 or 2 according to the value of $i = 1,2$ or 3 in the length $n = 3p + i$ of the line, or to the state A, B or C under the response signal, or now to the state on the response signal, C, A or B. We decide that the general will be activated by a new signal R produced after signal C, C being in turn produced after A, and A after B. Rules for this are (Figure 17)

$$\begin{array}{cccc} & R & & \\ A & C & \beta & \end{array} \text{ preceeded if necessary by } \begin{array}{cccc} & C & & \\ B & A & \beta & \end{array} \text{ and } \begin{array}{cccc} & A & & \\ C & B & \beta & \end{array}$$

The general will start acting as soon as he has R on his right by rule

$$\begin{array}{ccc} & H & \\ G & R & e \end{array}$$

Quietness comes back after R :

$$\begin{array}{cccc} & e & & \\ C & R & \beta & \end{array} \qquad \begin{array}{cccc} & e & & \\ C & R & e & \end{array} \qquad \begin{array}{cccc} & e & & \\ R & e & e. & \end{array}$$

(This R state cannot be simply replaced by the quiescent state, because of rule

$$\begin{array}{ccc} & B & \\ C & e & \beta \end{array}$$

which would result in the reappearance of signal B, and also because quietness cannot come back after states A, B and C).

The new general will act as right border for the remaining left part of the line.

We now hope that the same rules can make the S_k's to produce the following generals. Let us see if it is the case. First of all we must ascertain that the response signal keeps going straight on, or, to be more accurate, that reflection on the newly created general continues the initial response signal : well, on the left of the new general we have a quiescent state, the state of site D_1 ; the state presently at the left of the S_1 wave (X on Figures 16, 17) will reflect against G at the next time with the same rules, except we have G instead of the border (see below), this occurring on the return diagonal of G_1, which coincides with the initial response signal, as we wish.

$$\begin{array}{ccc} & C & \\ A & e & G \end{array} \qquad \begin{array}{ccc} & A & \\ B & e & G \end{array} \qquad \begin{array}{ccc} & B & \\ C & e & G \end{array}$$

$$\begin{array}{ccc} & A & \\ C & B & G \end{array} \qquad \begin{array}{ccc} & C & \\ B & A & G \end{array} \qquad \begin{array}{ccc} & R & \\ A & C & G \end{array} \qquad \begin{array}{ccc} & e & \\ C & R & G \end{array}$$

For cases at the end of the process when the newly created line has length 2 only, we must add rules

$$\begin{array}{ccc} & A & \\ G & B & \beta \text{ or } G \end{array} \qquad \begin{array}{ccc} & C & \\ G & A & \beta \text{ or } G \end{array} \qquad \begin{array}{ccc} & R & \\ G & C & \beta \text{ or } G \end{array}$$

We don't care any more for the length of the line, the number of new lines that have broken off, rank of the next S_k wave. The only important things are that the response signal is where it should be, that it carries the information requested, and that the next S_k wave is correctly set up, all matters we hold now for sure.

2.8 Very small lines

It remains for us to examine how lines of length 2 and 3 behave, be they original lines, or new lines, or remaining lines. It is in these lines that the fire finally appears.

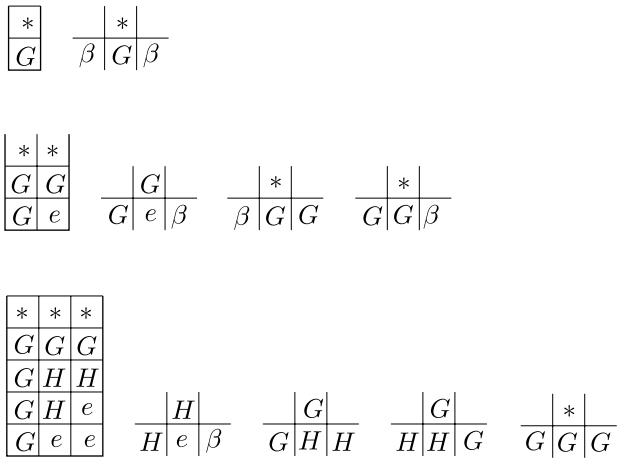


Fig.18

Figure 18 shows the s.t.d's that very small initial lines must obey. Corresponding rules are :

$$\begin{array}{ccc} & * & \\ \beta & G & \beta \end{array}$$

$$\begin{array}{ccc} & G & \\ G & e & \beta \end{array} \qquad \begin{array}{ccc} & * & \\ \beta & G & G \end{array} \qquad \begin{array}{ccc} & * & \\ G & G & \beta \end{array}$$

$$\begin{array}{ccc} & H & \\ H & e & \beta \end{array} \qquad \begin{array}{ccc} & G & \\ G & H & H \end{array} \qquad \begin{array}{ccc} & G & \\ H & H & \beta \end{array} \qquad \begin{array}{ccc} & * & \\ G & G & G \end{array}$$

For the remaining lines, the right border must be replaced by G, so we add

$$\begin{array}{ccc} & G & \\ G & e & G \end{array} \qquad \begin{array}{ccc} & H & \\ H & e & G \end{array} \qquad \begin{array}{ccc} & G & \\ H & H & G \end{array}$$

at last, for a new line of length 2 we must add

$$\begin{array}{cc} & G \\ G & R \end{array} \quad \beta \text{ or } G$$

It is time to mention here that state H, which characterizes the very small lines, obeys rules different from that of C. This is the reason why H cannot be replaced by C.

Rules having been established so as to launch the desired induction process and achieve this properly, and lines in which successively break the initial line all synchronizing in the desired time, we are now guaranteed that any line synchronizes in time $2n - 2$.

An example of a complete s.t.d is given in Figure 19.

*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
G		G	**R**	G	**R**	G		G	**R**	G	H	H	G	**R**	G	H	H	G	**R**	G	H	H
G		A	C	G	C	G		A	C	G	H		G	C	G	H		G	C	G	H	
G		A		G	A	G	H	A		G			B	A	G			B	A	G	**R**	
G		A		C	B	G	H			G		A	B		G		A	B		G	C	**R**
G		A		C		G	**R**			G	H	A			G		A			B	A	C
G		A		C		A	C	**R**		G	H				G		A		A	B	B	A
G	H	A		C		A	A	C	**R**	G	**R**				G	H	A		A	A	B	
G	H			C		C	A	A	C	G	C	**R**			G	H			C	A		
G	H		B	C		C	C	A		G	A	C	**R**		G			B	C			
G	H		B			C	C			C	B	A	C	**R**	G		A	B				
G	H		B		B	C	C		B	C	C	B	A	C	G	H	A					
G	H		B		B	B	C		B	B	C	C	B	A	G	H						
G			B		B	B			B	B	B	C	C	B	G	**R**						
G		A	B		B	B		A	B	B	B	B	C		G	C	**R**					
G		A			B	B		A	A	B	B	B			B	A	C	**R**				
G		A		A	B	B		C	A	A	B	B		A	B	B	A	C	**R**			
G		A		A	A	B		C	C	A	A	B		A	A	B	B	A	C	**R**		
G		A		C	A			C	C	C	A			C	A	A	B	B	A	C	**R**	
G		A		C			B	C	C	C			B	C	C	A	A	B	B	A	C	**R**
G		A		C		A	B	B	C	C		A	B	B	C	C	A	A	B	B	A	C
G	H	A		C		A	A	B	B	C		A	A	B	B	C	C	A	A	B	B	A
G	H			C		A	A	A	B			C	A	A	B	B	C	C	A	A	B	
G	H		B	C		A	A	A			B	C	C	A	A	B	B	C	C	A		
G	H		B			C	A	A		A	B	B	C	C	A	A	B	B	C			
G			B		B	C	C	A		A	A	B	B	C	C	A	A	B				
G		A	B		B	B	C			C	A	C	B	B	C	C	A					
G		A			B	B			B	C	C	C	A	B	B	C						
G		A		A	B	B		A	B	B	C	A	A	A	B							
G		A		A	A	B		A	A	B	B	A	C	A								
G		A		C	A			C	A	A	B	B	C									
G	H	A		C			B	C	C	A	A	B										
G	H			C		A	B	B	C	C	A											
G			B	C		A	A	B	B	C												
G		A	B			C	A	A	B													
G		A			B	C	C	A														
G		A		A	B	B																
G		A		A	A	B																
G	H	A		C	A																	
G	H			C	A																	
G			B	C																		
G		A	B																			
G	H	A																				
G	H																					
G																						

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

Fig.19

2.9 Collecting the rules for an 8-state solution

Each of the following tables concerns one central state, in the left column the left state is to be found, and on the upper row the right state.

Some obvious rules, mainly stability of certain states, have not been explained. But they naturally appear in the tables, which rehearse all cases.

If certain triples of states do not appear, or the place of the resulting state remains empty, the reason is that they never occur in the s.t.d's. But in this first rehearsal we may have listed rules that are in fact never used.

e	e	β	G		A	B	C
e	e	e	e		e	e	e
β	e				e	e	e
G	H	G	G		e	e	e
H	A	H	H		e	e	e
A	B	C	C		e	e	e
B	C	A	A		e	e	e
C	A	B	B		e	e	e
R	e						

G	β	G	?
β	$*$	$*$	G
G	$*$	$*$	G
?	G	G	G

H	e	β	G	H	A
G	H			G	e
H		G	G		

R	e	β	G
G	H	G	G
C	e	e	e

A	e	β	G	A	B	C
e	A			A	e	G
β	A			A	B	C
G	A	C	C	A	B	C
H	A			A	B	C
A	A			A	B	C
B	A	C	C	A	B	C
C	A			A	B	C
R	A			A	B	C

B	e	β	G	A	B	C
e	B			G	B	e
β	B			A	B	C
G	B	A	A	A	B	C
H	B			A	B	C
A	B			A	B	C
B	B			A	B	C
C	B	A	A	A	B	C
R	B			A	B	C

$\boldsymbol{C}$		e	β	G	A	B	C	R
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
e	$\vert$	C			A	G	C	
β	$\vert$	C			A	B	C	
G	$\vert$	C	R	R	A	B	C	R
H	$\vert$	C			A	B	C	
A	$\vert$	C	R	R	A	B	C	R
B	$\vert$	C			A	B	C	
C	$\vert$	C			A	B	C	
R	$\vert$	C			A	B	C	

It is recommended to work out an example for oneself !

We shall now examine tables closely so as to eliminate all rules that are never used, the best way for doing this being to write a program making the s.t.d's for all wished lengthes :

$\boldsymbol{e}$		e	β	G	H	A	B	C	R
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
e	$\vert$	e	e	e			e	e	
β	$\vert$								
G	$\vert$	H	G	G		e			
H	$\vert$	A	H	H			e		
A	$\vert$	B	C	C		e		e	
B	$\vert$	C	A	A		e	e	e	
C	$\vert$	A	B	B		e	e	e	
R	$\vert$	e	e	e					

$\boldsymbol{G}$		e	β	G	H	A	B	C	R
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
e	$\vert$				G	G		G	G
β	$\vert$	G	$*$	$*$	G				
G	$\vert$		$*$	$*$					
H	$\vert$	G			G				G
A	$\vert$	G			G			G	G
B	$\vert$	G			G			G	G
C	$\vert$	G			G			G	G
R	$\vert$	G			G				G

$\boldsymbol{H}$		e	β	G	H	A
$-$	$+$	$-$	$-$	$-$	$-$	$-$
G	$\vert$	H			G	e
H	$\vert$		G	G		

$\boldsymbol{R}$		e	β	G
$-$	$+$	$-$	$-$	$-$
G	$\vert$	H	G	G
C	$\vert$	e	e	e

$\boldsymbol{A}$		e	β	G	H	A	B	C	R
G		A	B	C					
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
e	$\vert$	A				A	e	G	
β	$\vert$								
G	$\vert$		C	C					
H	$\vert$	A							
A	$\vert$	A				A	B	C	
B	$\vert$		C	C				C	
C	$\vert$	A				A			
R	$\vert$								

$\boldsymbol{B}$		e	β	G	H	A	B	C	R
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
e	$\vert$	B				G	B	e	
β	$\vert$								
G	$\vert$								
H	$\vert$								
A	$\vert$	B					B		
B	$\vert$	B				A	B	C	
C	$\vert$		A	A		A			
R	$\vert$								

C	e	β	G	H	A	B	C	R
e	C				A	G	C	
β								
G	C	R	R					R
H								
A	C	R	R					R
B	C						C	
C	C				A	B	C	
R								

2.10 Reducing the number of states

Two among the 8 states e, G, F, A, B, C, H and R, the two last ones, appear very seldom and in special circumstances. Mazoyer has contrived and succeeded in replacing them by certain of the other states.

How ? First, the starting state of the B-signals is changed for G, so Aee produces G and Gee produces C. With AAG producing B state B is restored, and with AGe producing B the second B-signal starts with B.

Then state R is replaced by state B. After B quietness can come back if Bee produces e, which is now possible since signal C is now generated by Gee.

When such changes are made, a careful and detailed review of all rules where the states involved appear must then take place, to make sure that no contradiction has been introduced in the rules, and there is no alteration of the synchronizing process.

At last state H has been replaced by states C and B depending on the step parity, and this leads to a complete altering of the process on cells 2 and 3. But all other cells work as before.

The resulting c.a has 6 states only (β is not a state), and as a consequence its transition tables are very much reduced :

e	e	β	G	A	B	C
e	e	e	e		e	e
β						
G	C	A	A	e	e	e
A	G	C	C	e		e
B	e	e	e	e	e	e
C	A	G	G	e	e	e

G	e	β	G	A	B	C
e				G	G	G
β	A		$*$		G	G
G	B	$*$	$*$		G	G
A	B		e		G	G
B	B	G	G		G	G
C	A	A	A		G	G

$\boldsymbol{A}$	|	e	β	G	A	B	C
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$
e	|				A	e	G
β	|						G
G	|			C			C
A	|	A		B	A	B	C
B	|	G	C	C		G	C
C	|	A			A		

$\boldsymbol{B}$	|	e	β	G	A	B	C
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$
e	|			B	G	B	e
β	|						
G	|	C	G	G	C		B
A	|	G			B	B	e
B	|	G		B	A	B	C
C	|	e	e	e	A		

$\boldsymbol{C}$	|	e	β	G	A	B	C
$-$	$+$	$-$	$-$	$-$	$-$	$-$	$-$
e	|	C		G	A	G	C
β	|						
G	|	B		B		B	
A	|	B	B	B		B	
B	|	C		G			C
C	|	C		B	A	B	C

All this tinkering has not changed the fire line, so the new automaton with 6 states synchronizes in minimal time. Figure 20 shows synchronisation by this automaton of the same line, of length 23, which was synchronized in Figure 19.

As we have not explained the last changes that have been made in the states we can not consider that we have given a proof that this automaton is a solution. But the proof is given in [35], and has moreover been checked by a mechanical means, the Coq proof assistant, by Jean Duprat [13].

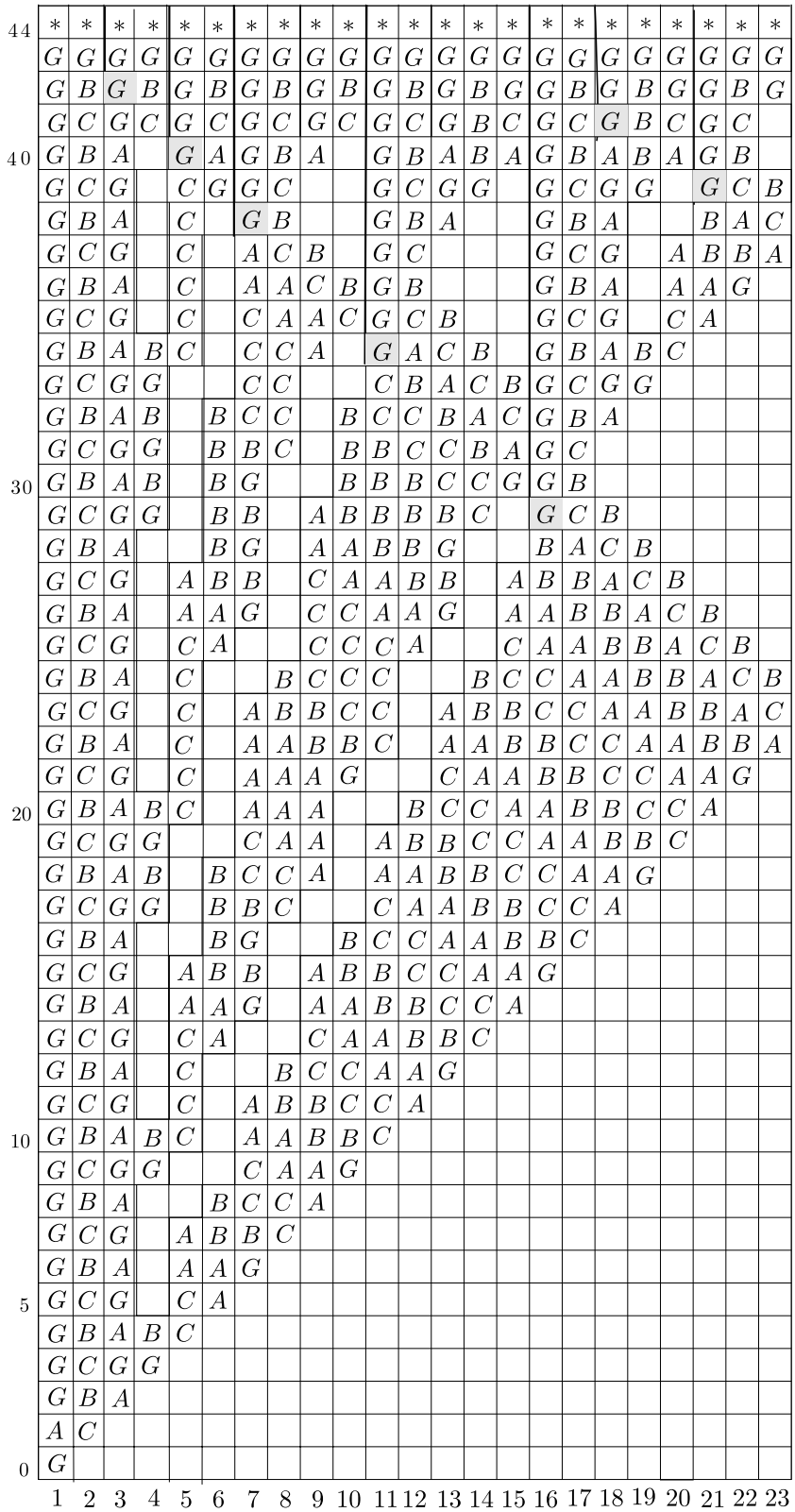


Fig.20

Chapter 3

Half-lines : generalities

Half-line is an abbreviation for : semi-infinite linear c.a with scope 1.

3.1 A basic definition

3.1.1 General structure

The title makes it clear that we shall deal here with a line of cells with set of indexes $\mathbb{N}$. The state of a particular cell depends on its proper state and the states of its two left and right neighbours at the preceding time-step. The set of states (which is finite, and contains a quiescent state e) will still be denoted Q, and the transition function δ (with $\delta(e, e, e) = e$).

But this basic structure needs to be completed in many ways so as to suit various purposes and uses. Each author writing on the subject has adopted the definition best suitable for his results. The main variations between different authors concern inputs and outputs. We shall try here to give a definition general enough, natural enough, and simple enough to serve as a reference.

The following simple observations will give some support to our choices.

Our c.a's are abstract machines meant to be models for real machines or organisms. The advantage of machines is that they work by themselves, and as for organisms, they live by themselves. This does not prevent outside interference, but it should be limited. As a matter of fact, should some outside actor interfere at all places and all times, we could no more speak of autonomy. Such an actor is then left to interfere at some determined time, and that will be at starting time, or at some determined place, where it can introduce data that will contribute to the running of the machine while being processed by it. Result, or results, will samely be collected at some determined place, progressively as work proceeds, or at the end, when the machine stops. It will not be forbidden to open the machine, to put it in a determined state, to let it work and open it again to see what has happened, but this seems more how a repairer would act than a user. We rather imagine the user introducing information at some precise input place, and collecting results at some precise output place. Finally,

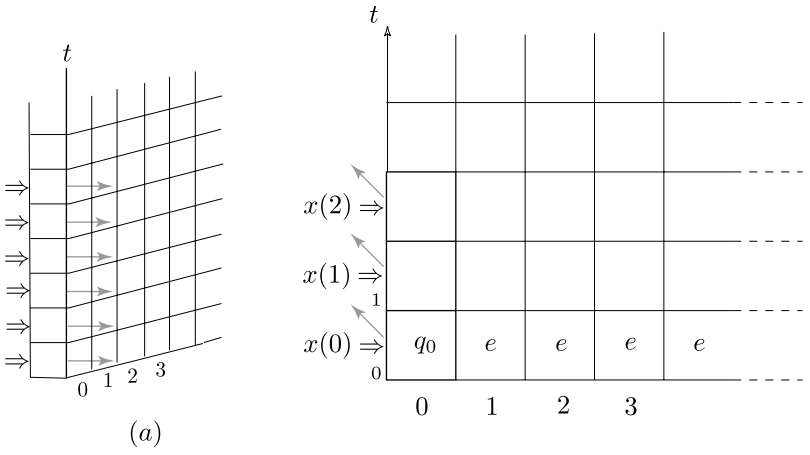
let us observe that the starting state of the machine should always be the same (otherwise the result would be unpredictable and undetermined), while the outside actor introduces data which vary (otherwise we could make them, once and for all, part of the machine). We shall keep these remarks in mind in the sequel.

3.1.2 Initial state - Input and output

Initial state is the configuration of states of the cells at time 0, the starting time. It seems that it could be any infinite sequence of states in Q, but in fact we shall restrict ourselves to "impulse" starting states, of the form

$$(q_0, e, e, \ldots\ldots\ldots)$$

where only cell 0 is non-quiescent (Figure 1).



(a) : inputs and outputs as a process "orthogonal"(though not independent) to the evolution of the half-line

Fig.1

There is only one input cell, cell 0, which is therefore different from the others, which are all alike. So we must consider two transition functions, $\delta : Q^3 \longmapsto Q$ for ordinary cells,

$$< c, t+1 >= \delta(< c-1, t >, < c, t >, < c+1, t >)$$

and $\delta_0 : X \times Q^2 \longmapsto Q$ for cell 0, where X is the input alphabet

$$< 0, t+1 >= \delta_0(x(t), < 0, t >, < 1, t >).$$

Input goes on without interruption while the c.a works, but nothing forbids introducing in X a void symbol, which will allow all interruptions wished for,

as well as the ending of inputs. It may occur that an end marker $*$ is also added to X. We naturally suppose that

$$\delta_0(-, e, e) = e, \qquad \text{where } - \text{ is the void symbol.}$$

Cell 0 is also the only one with an output. Output will depend only on state of cell 0 (Moore's choice). The output alphabet will be denoted Y, it is finite, and the output function will be $\sigma : Q \longmapsto Y$. As for inputs, Y may contain a void symbol and an end marker.

We have chosen here the simplest possible output function. Let us give just one example of some other possibility : an output function taking the input into account

$$\sigma : X \times Q \longmapsto Y$$

(Mealy's choice) would allow a very simple machine (with very few states) to handle numerous inputs with as numerous outputs produced.

Input and output as we have just described them are called *sequential*, especially if other ways for introducing and retrieving data are considered. In our opinion they are the only genuine input and output, so unless otherwise mentioned, input and output will always be sequential. The so-called parallel input is dealt with in section 3.3.

3.1.3 Infinite linear c.a's

Definition is the same except that set of indexes for the cells is $\mathbb{Z}$, inputs enter and outputs exit through central cell 0, which is also the one where impulse initial states are set.

We shall not study such infinite c.a's, the reason being that they can be reduced to semi-infinite c.a's if we "fold" them in two by the middle : it is very easy to understand what this means, so explanations are not really needed, let us only mention that states on cell 0 will still be states of Q, while cells i, for positive i, will have double states, i.e states of cells i and $-i$ together, with matching transition function. The set of states of the semi-infinite c.a is $Q \cup Q \times Q$.

Let us mention another way of folding the infinite c.a's : by bringing cells -1,-2, ... , $-i$, ... over cells -0,-1, ... , $i-1$, ... Then the set of states of the semi-infinite c.a is only $Q \times Q$.

3.1.4 Halting the c.a

Two ways for halting can be thought of :

- halting when input ceases : δ_0 is not defined if input is the end marker or inputs have come to an end. If n is the length of the word of inputs, halting time is then necessarily n (Figure 2).

- halting with a halting state : among the states of Q some are halting states. They constitute subset Q_a, on which δ_0 is not defined. Halting time has then no relation with the length of the input word (Figure 2).

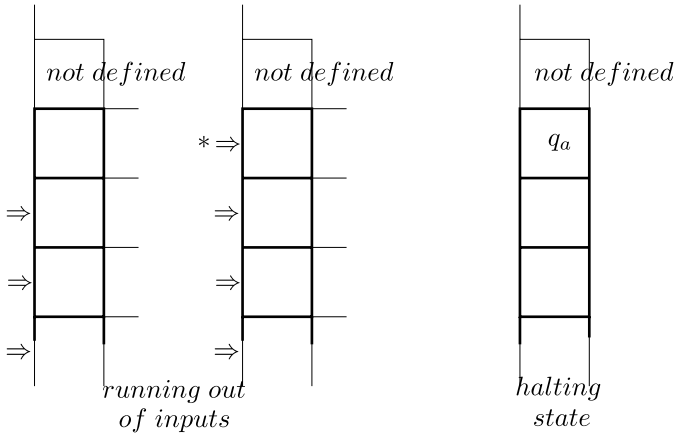


Fig.2

Halting is thus determined on cell 0. We may observe that the next cells go on working for a while, stopping one after the other, cell c at time $t_h + c$ if t_h is the halting time of cell 0.

3.1.5 Recognizing or computing

As all models of abstract machines, c.a's can be used in two ways, as acceptors or as computers.

If a c.a works as a computer, the result for an input word of X^* is the word of Y^* given by the output function σ of the sequence of states of cell 0, up to the halting time.

If a c.a works as acceptor, it accepts words of X^* which as input words lead to a halting considered accepting :

- if the c.a halts because input ends, it will accept or reject depending on value of the output function (so on the state of cell 0 when it halts) ; most of the time this function has the two values 1 (accepting) and 0 (rejecting).

- if the c.a halts with a halting state, subset Q_a divides in two parts, accepting states and rejecting states. It can occur that some input words, which therefore will not be accepted, never lead to a halting state.

3.1.6 Product of cellular automatas

As in the case of finite lines, it is the c.a obtained by running simultaneously two (or more) c.a's having of course the same structure, here that of a semi-infinite line.

If the two c.a's have state sets and transition functions :

$$(Q, q_0, \delta_0, \delta) \quad \text{and} \quad (P, p_0, \gamma_0, \gamma)$$

the product has state set $Q \times P$, (whose elements we shall denote $q \times p$ rather than (q, p)), impulse initial state $q_0 \times p_0$, and transition function

$$\Delta(q_l \times p_l, q \times p, q_r \times p_r) = \delta(q_l, q, q_r) \times \gamma(p_l, p, p_r)$$

for ordinary cells. We don't give transition function Δ_0 for cell 0, as different ways for combining inputs may be thought of. Halting, and output, are defined depending on the task to be achieved.

Product, which is theoretically a most simple operation, will be our most important means for constructing, with already existing c.a's, new ones, having, inevitably, a great number of states : $|Q| \times |P|$. A product of c.a's was already implemented in chapter 1.

3.1.7 Site dependence

Figure 3 shows the s.t.d of a semi-infinite c.a, where we have indicated, for any site whatever, sites from which its state results and sites that its state influences, which we can call its dependance triangle and its influence triangle.

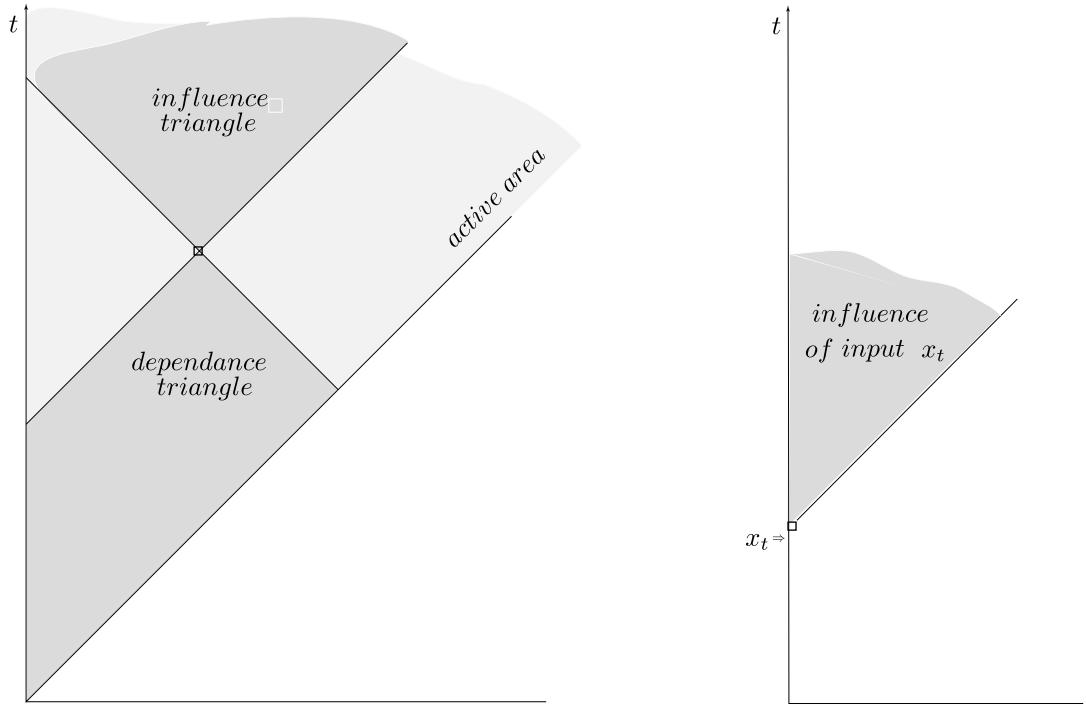


Fig.3

3.1.8 Inputs and states

Input x_t at time t contributes to determine the state of cell 0 at time $t+1$ and as a consequence influences the later run of the c.a (Figure 3). It has been digested by the c.a, and has disappeared as such.

In fact, more complicated treatments may be set up : it will be possible for inputs to travel inside the c.a, so as to be used at some other place than cell 0, or at some later time. Indeed the c.a states may be complex elements, formed of several components, one or more of these being stored inputs, which the c.a may then pass from one cell to the next. This storing and transporting of the inputs is then part of the c.a's work. Numerous examples of this will be met in the sequel.

3.2 Time for recognition

In this section we are concerned only with c.a's used as recognizers and with the languages that they recognize.

3.2.1 Recognizing time

The time $T(w)$ needed by the c.a (or any other machine) for recognizing some word w depends on this particular word. Let us first observe that if some word w is accepted before it is entirely read, then all extensions of a prefix of this word are also accepted ; the shortest prefix u of w such that all its extensions are accepted appears then to be more interesting a word than$w$ itself, and this word cannot be accepted before it is completely read.

To be very precise, let us observe that any language (set of words) accepted by a c.a or another machine may be partitioned as

$$L = L_1 X^* \cup L_2$$

where L_2 is the subset of words of L which have at least one extension not in L and L_1 is the set of words of L that are minimal among those having all their extensions in L. (Proper) extensions of words of L_1 thus do not belong to L_1, and every word in L_2 has at least one (proper) extension that does not belong to L_2 : so, in order to be accepted, words of L_1 and words of L_2 must be entirely read. So from now on we shall consider only such languages, whose words all have an extension not in the language. As for language L, a word belongs to it either if it is in L_2 or if one of its prefixes is in L_1.

Now then, to read to its end a word $u = x_1 \dots x_n$ of length n, n units of time are necessary. Time for recognition $T(u)$, which is the number of transition steps from initial state to accepting, will therefore be no less than n, the length of u

$$T(u) \geq |u|.$$

If we should acknowledge that the input word is ended one more unit of time would be needed.

Length of a word is only one aspect of its particular value, but an essential one : it is customary, justified, and anyway we cannot do otherwise in a general study, to admit that recognition time depends only on the length of input words. (It probably is the most frequent case and when it isn't results should indeed be precisely reexamined). Recognition time will consequently be expressed as $T(n)$ and so we admit that

$$T(n) \geq n.$$

3.2.2 Real time recognition - Computing time

Acceptance of a word takes place as c.a halts, and we have distinguished two halting modes.

Let us examine first the case when halting occurs as input ends, and acceptance is by an accepting value of the output function :

> Accepting time is $n = |u|$ (Figure 4). At time n, value of the output function tells if, in case input comes to an end, word $x_0 \dots x_{n-1}$ will

be accepted, without our knowing yet if it is continued by $x_n \ldots$. So all along we know if the part of word already read belongs to the language or not : that is *interactive* recognition, and *strict real time*

$$T(n) = n$$

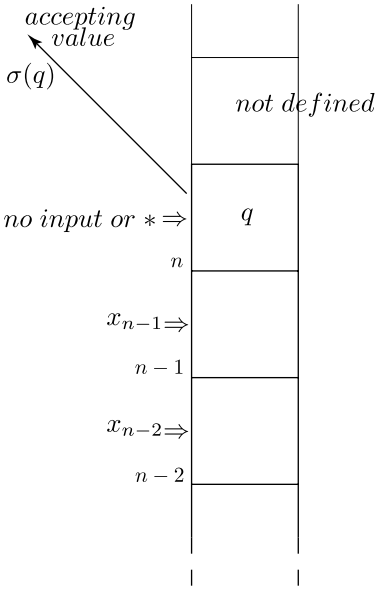


Fig.4

Let us now examine the case when accepting is by an accepting halting state :

can recognition time be n ? no, because halting state would occur before we know what input comes after the n-th letter, x_{n-1} and we have excluded this situation (Figure 5).

The quickest imaginable recognition will be if entering of the end marker, or first empty input, systematically brings a halting state, and then

$$T(n) = n + 1$$

in this case we shall speak of *large real time*.

Of course recognition can also occur later.

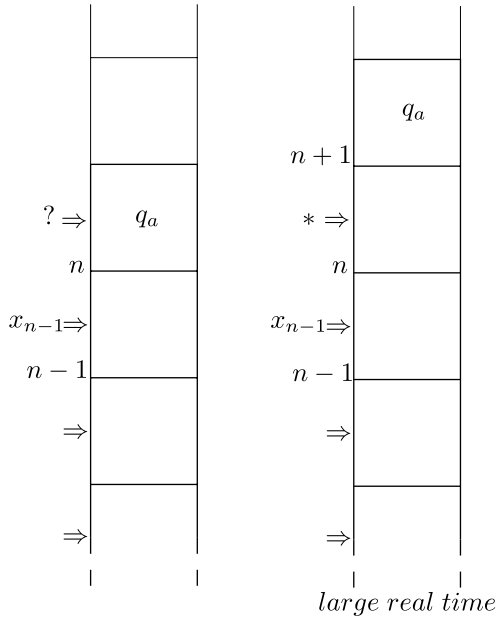


Fig.5

The first definition is strictly more demanding.

For computers, we shall likewise consider only computations involving the whole input word, so computing time cannot be less then the length of the input word. Generally, the result has not necessarily the same length as the input, so halting will be by halting state, and minimum time will then be $n+1$.

3.2.3 A property of languages accepted in strict real time

Let L be some language on alphabet X. Syntactic equivalence mod L for words of X^*, which we shall denote $\equiv^L$, is defined by

$$w \equiv^L w' \quad \text{if and only if} \quad \forall u \in X^* \quad wu \in L \Leftrightarrow w'u \in L.$$

In the same way we define, for any integer k, syntactic equivalence $\equiv_k^L$ by

$$w \equiv_k^L w' \quad \text{if and only if} \quad \forall u \in X^* \text{ such that } |u| \leq k \quad wu \in L \Leftrightarrow w'u \in L.$$

So two words are separated by $\equiv_k^L$ if and only if there exists a word u of length not more than k which extends the one in a word of L and the other in a word not in L.

Let now L be a language accepted in strict real time by some c.a with h states ($h \geq 2$). If two words w and w' are in two different classes of $\equiv_k^L$, indeed their exists some word u, of length not more than k, such that wu is accepted and

$w'u$ is not (or vice versa) : state immediately following input of wu is accepting, while state immediately following input of $w'u$ is rejecting. But these two states depend only on word u and states of cells $0, 1, \ldots, |u|$ after entering of words w and w' respectively (Figure 6) ; length of u being no greater than k, these cells are part of the $k+1$ cells 0, 1, ... , k. c.a having h states, the number of possible configurations of states for these $k+1$ cells is at most h^{k+1}, so index of $\equiv_k^L$ cannot exceed h^{k+1}.

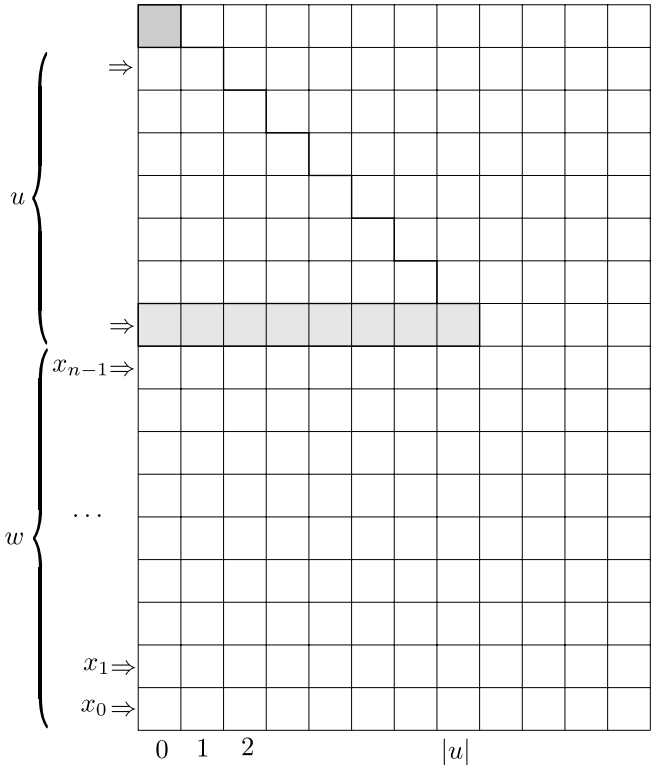


Fig.6

Criterion 3.2.1 (Cole) *if a language L is accepted by some c.a having h states in strict real time, then for all $k \geq 0$, index of equivalence $\equiv_k^L$ is no greater than h^{k+1}.*

If for some language L we find a number k such that $\equiv_k^L$ has more than h^{k+1} classes, this language can not be accepted by any c.a with h states in strictly real time.

We obtain a similar result for large real time

Criterion 3.2.2 *if a language L is accepted by some c.a having h states in real time (large), then for all $k \geq 0$, index of equivalence $\equiv_k^L$ is no greater than h^{k+2}.*

Proof is the same.

The criterion as presented here is the particular case, in dimension 1, of Cole's criterion in dimension n, to be demonstrated in chapter 10. In section 10.5.3 we also prove that the converse of this criterion is false.

3.3 Parallel input

Even if our opinion is that such an input as we shall describe next is not a model for a real input, it may be interesting to consider as a theoretical input, or as a possible stage, all the more because many works take as a starting point a parallel situation.

3.3.1 Conceiving a parallel input

See Figure 7 : initial state is quiescent (it is out of question to mix everything up, initial states and inputs) and input word is $u = x_0x_1 \dots x_{n-1}$. State of a cell at time 1 depends on

- input for this cell at time 0, (but not on input of neighbouring cells)

- states of cell itself and left and right neighbours at time 0, which all are the quiescent state (or the border for cell 0)

Transition function of time 0 is $\delta^0 : Q^3 \times X \longmapsto Q$ (it is on purpose that here 0 is an exponent and the inputs are written on the right, to mark the distinction from the sequential case)

$$< 0, 1 >= \delta^0(\beta, e, e, x_0)$$

$$< i, 1 >= \delta^0(e, e, e, x_i).$$

As for the sequential case, X must contain a void symbol which serves as "no input".

Let us name e_i the states $\delta^0(\beta \text{ or } e, e, e, x_i)$. Naturally $\delta^0(\beta \text{ or } e, e, e, -) = e$. The subset

$$E = \{e_i = \delta^0(e, e, e, x) \mid x \in X\}$$

of Q is in one-to-one correspondance with the input alphabet ; at time 1 we have a pseudo-initial state which is an image of the input word. Authors which use parallel input just declare without making fuss that input alphabet is a subset of Q and place the input word as initial state, which comes to starting from our time 1 situation.

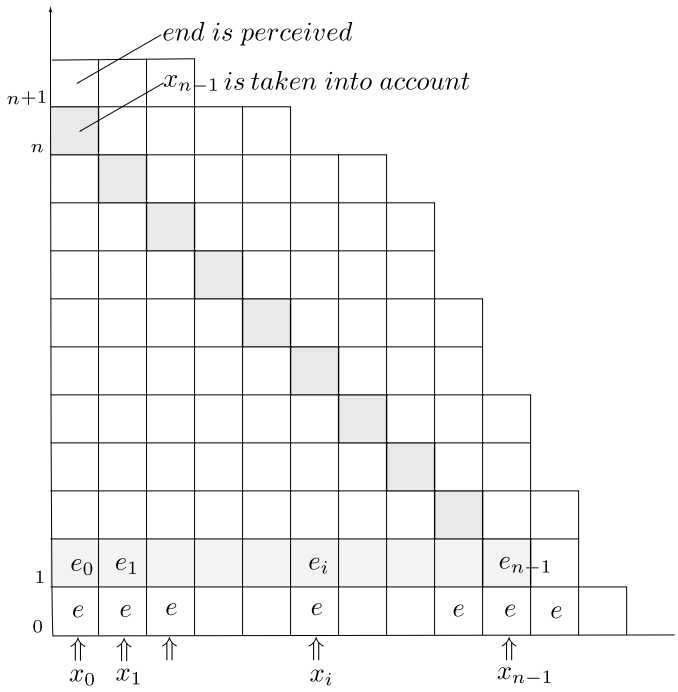


Fig.7

This parallel input is sometimes used by some authors to set up a particular configuration, and very often used for recognition purposes. As the input word is entirely known from time 1, the accepting times have good chance of being better. Nevertheless, accepting is by cell 0, and cell 0 cannot know of the last letter x_{n-1} before time n and know that the input word is ended before time $n+1$, and this knowledge is essential if all the extensions of word $x_1 \dots x_n$ should not be accepted. So accepting time is $T(n) \geq n+1$, at best it is large real time.

In what precedes we have claimed that initial state should be quiescent. But we shall be a little less rigorous and admit that it can be what we have called an impulse initial state, mixing up of initial state and input being in this particular case extremely limited.

The question we now put is : can we use a parallel recognition to build a real sequential recognition ? The reverse question seems to have no very great interest, we deal with it first.

3.3.2 Constructing a parallel c.a from a sequential one

Let A be the sequential c.a we have at first, Q, q_0, δ and δ_0 its set of states, initial state and transition functions.

The parallel c.a A', after having absorbed the input word, will simply send its letters left, (at the right of the cells in Figure 8, that is as right components

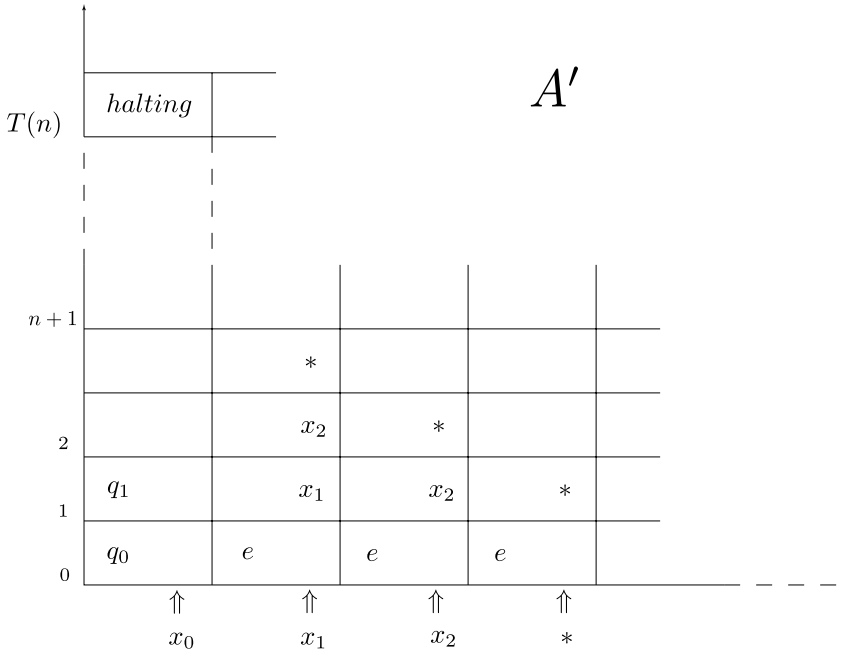


Fig.8

of the new states), so they will arrive against cell 0 one after the other ; then they can be used as were the inputs of A.

States and principal rules of A' will be :

$$Q' = Q \ \cup \ Q \times X, \qquad \text{initial state} \quad q_0.$$

For the new states formed of a state of A and an input letter we choose here notation : $q \times x$ rather than (q, x).

At time 0

$$\delta'^0(\beta, q_0, e, x) = \delta_0(x, q_0, e)$$

$$\delta'^0(q_0, e, e, x \text{ or } *) = \delta(q_0, e, e) \times x \text{ or } *$$

$$\delta'(e, e, e, x \text{ or } *) = e \times x \text{ or } *$$

then inputs travel left to cell 2

$$\delta'(l \times z, q \times y, r \times x) = \delta(l, q, r) \times x$$

$$\delta'(l, q \times y, r \times x) = \delta(l, q, r) \times x$$

and then at last

$$\delta'(\beta, q, r \times x) = \delta_0(x, q, r)$$

It is easy to check that A' works correctly, even if input word has only one letter : x_0*. From time $n+1$ on, its s.t.d is the same as s.t.d of A. If A accepts by accepting halting state, the recognition time will be the same. But if A accepts in strictly real time, A' will not do quite as well : if we decide that any state for which output function is accepting produces an accepting state when it detects the end marker (or no more input), we shall obtain what we have called the large real time.

3.3.3 Constructing a sequential c.a from a parallel one

Clearly the new automaton A' will have the parallel c.a A as essential feature, but the latter cannot start working until it is presented the input word neatly settled on cells 0, 1, ... Furthermore, the only way to start this working on the row of cells will be by the fire state of a synchronization .

To prepare all this we shall need new states with several components, states or inputs, which we shall evocate with images rather than define with scientific precision, hoping to make clear how they work without a tedious listing of numerous rules.

The new states will have a low part (Figure 9) reserved for the penetrating of the inputs : an input finding this space empty settles in the right corner ; if the right corner is already occupied, it settles in the left corner, then travels from this left corner to the left corner of the next cell on the right until it finds a free right corner. At time $2n-1(/2n+1)$ all the inputs (as well as end marker $*$) are in the right corners of cells 1 to n, available for the parallel automaton.

Now we need the synchronous starter that our new c.a must set of his own, internally. The way to do it will be a F.S.S.P synchronization process (see chapter 1) started by the entering of the end marker in cell 0. This synchronization must extend on a number of cells equaling length of the input, and then bump on a border, or rather some state acting as a border : any state having in its right corner nothing yet or the end marker will do. synchronization will develop independently from the travelling of the inputs, in the upper part of the sites, in minimal time (that is number of transition steps)

$$S(n) = max(1, 2n-2).$$

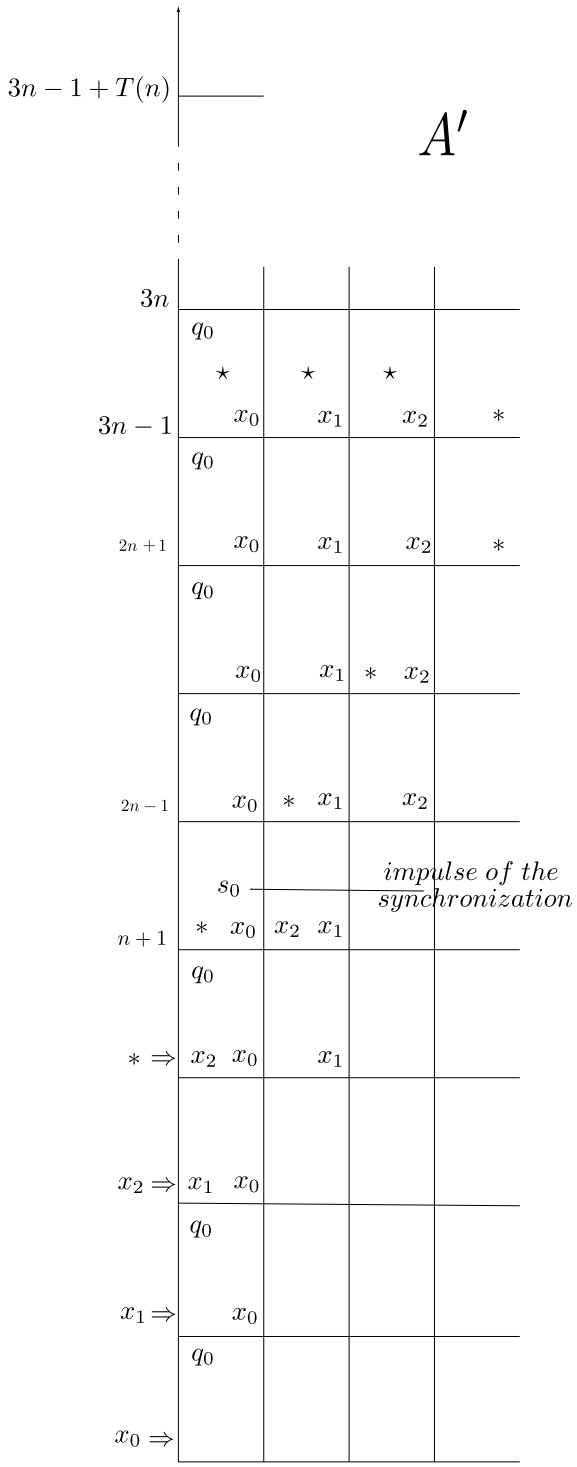

$3n-1+T(n)$
A'
$3n$
$3n-1$
$2n+1$
$2n-1$
$n+1$
$*\Rightarrow$
$x_2\Rightarrow$
$x_1\Rightarrow$
$x_0\Rightarrow$
q_0
$\star$
x_0 x_1 x_2 $*$
s_0
$*$ x_0 x_2 x_1
x_2 x_0 x_1
x_1 x_0
impulse of the synchronization

Fig.9

In this upper part we shall keep a little place where the q_0 impulse of A' will stay waiting. Fire state appears at time $n + 1 + 2n - 2 = 3n - 1$, at time 3 if $n = 1$. Now we decide that this state (for the end marker its proximity will suffice) wakes all the sleeping elements up, q_0 and the inputs, so that from time $3n$ up we have exactly the s.t.d of A from time 1. The recognizing time of A' is therefore

$$T'(n) = T(n) + 3n - 1$$

$$T'(1) = T(1) + 3 \text{ if } n = 1.$$

The number of states added to Q is in proportion with the square of the number of input letters, and with the number of states for synchronization, that makes a lot !

For a conclusion, let us point out that a parallel c.a, which disposes of the whole information since time 0, has a serious advantage over a sequential one.

3.4 Three exemples

The three are examples of recognition in strict real time, second and third examples are from Cole [6].

3.4.1 A c.a for the language $\{a^n b^n | n \in \mathbb{N}^*\}$

Cells will behave as little files containing at most three letters a or b. States will thus be triples, possibly partly empty (on the right). When a cell is full, its right element is pushed out towards the neighbouring cell, and when it isn't, it attracts an element from the right cell. Inputs stuff in cell 0 before they are pushed on the right, except if input is a b and cell 0 contains a's : then this b disappears along with an a. Besides, (in order to exclude words containing ba), as soon as a b is entered it is memorized in a corner of cell 0 and entering an a makes accepting definitetively impossible : for example it can change this b in a permanent x. Output function has value 1 if cell 0 is empty, so with no memorized x (Figure 10). In the meanwhile states have become quadruples.

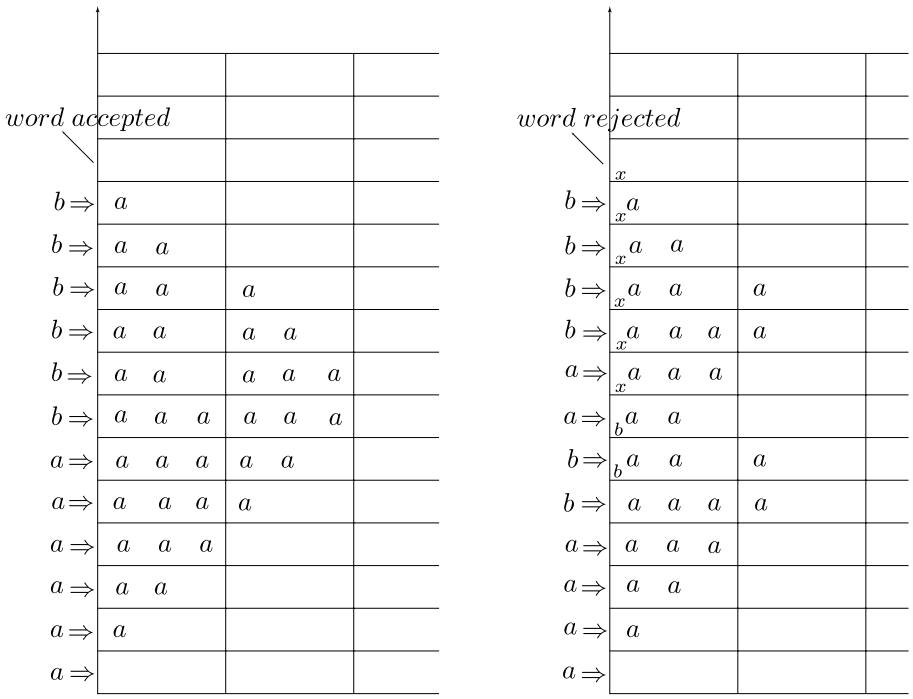


Fig.10

3.4.2 A c.a for the language of square words

Alphabet X is of course supposed to have at least 2 letters.

We find the geometric idea in Figure 11 : from two points of axis Oy having ordinates y et $y' = y + p$ two lines are issued, one is broken by reflections and has slopes 1,-1 et 3, the other has slope 1, they intersect on the line of slope -1 issued from point of ordinate $2p$. These two lines will be the trajectories for the inputs, which will bring them to meet at the point where we shall compare them.

Comparisons of inputs x_i and x_{i+p} for $i = 0, \ldots p-1$ will be collected on the third line arriving at point of ordinate $2p$ on Oy (Figure 12).

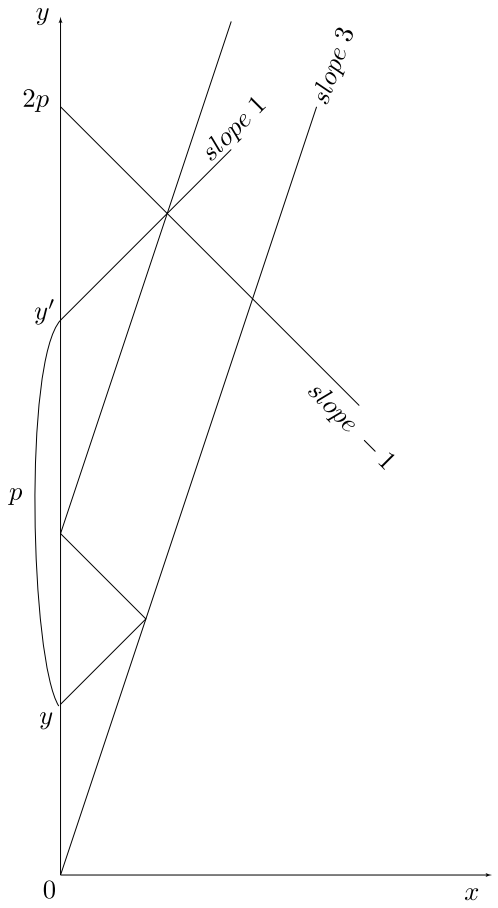


Fig.11

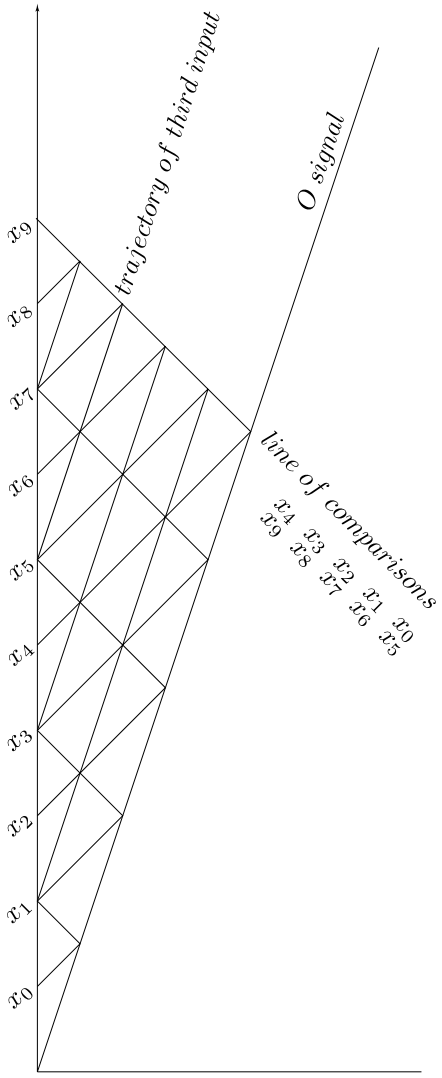

trajectory of third input
O signal
line of comparisons
x_9
x_8
x_7
x_6
x_5
x_4
x_3
x_2
x_1
x_0
x_4 x_3 x_2 x_1 x_0
x_9 x_8 x_7 x_6 x_5

Fig.12

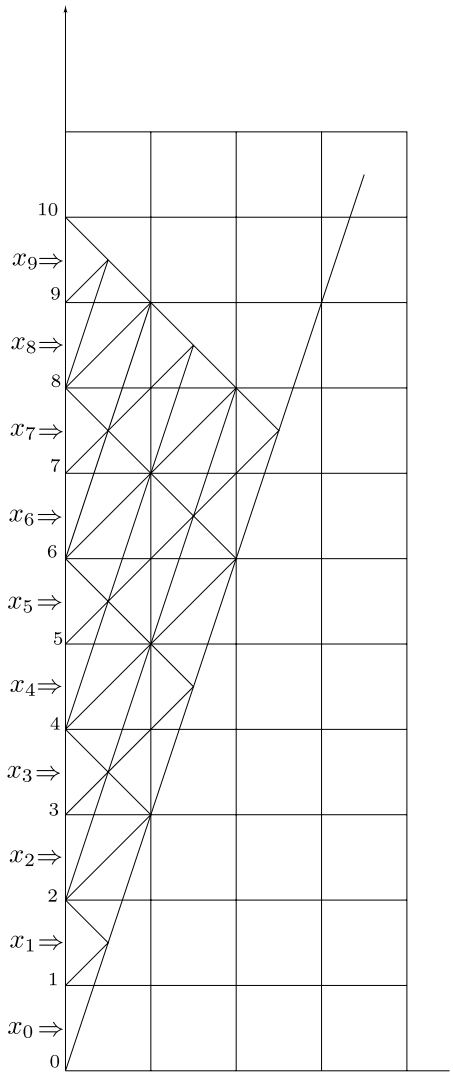

10
$x_9\Rightarrow$
9
$x_8\Rightarrow$
8
$x_7\Rightarrow$
7
$x_6\Rightarrow$
6
$x_5\Rightarrow$
5
$x_4\Rightarrow$
4
$x_3\Rightarrow$
3
$x_2\Rightarrow$
2
$x_1\Rightarrow$
1
$x_0\Rightarrow$
0

Fig.13

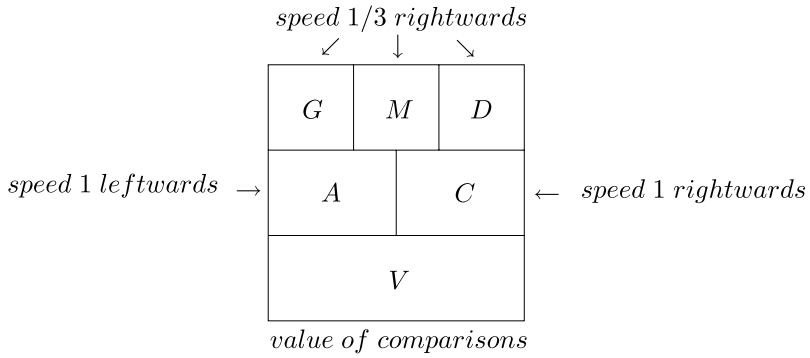


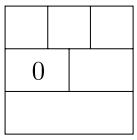
Fig.14

On next Figure 13 we have added the sites of the s.t.d, to help understand how the states are built : these will approximately reproduce the sites, the way inputs travel through being indicated by the places they occupy. These places, indicated in Figure 14, are 5, named G, M, D, A, C, they may be occupied by input letters or a null symbol 0. In addition we shall have a place, named V, for the result, 0 or 1, of cumulated comparisons. Each of these 6 places may be empty (symbol $\emptyset$ in next formula, no symbol in Figure 16), which is not the same as containing symbol 0 or value 0. So states will be sextuples belonging to

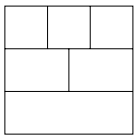
$$[X \cup \{0\} \cup \{\emptyset\}]^5 \times \{0, 1, \emptyset\}.$$

We don't intend to give the rules : we only give, in Figure 15, some essential elements of the rules, from which all the rules could be deduced, and with which we can build the s.t.d, represented in Figure 16. It must be observed that all elements of any site in the s.t.d depend exclusively on information contained in the three sites of the time preceeding.

initial state :



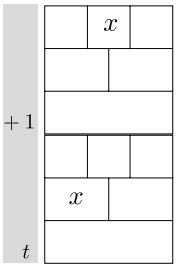
quiescent state :



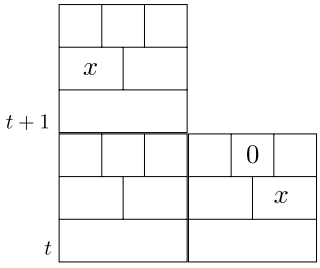
input on initial state : goes in place A
on any other state : goes in place C

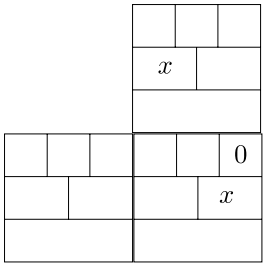
reflection

on left border



on signal 0





generation of $1-$*values on signal* 0 :

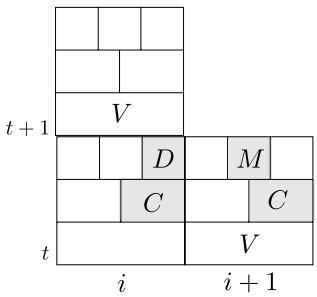
place V of cell i contains 1 *at time* $t+1$
if and only if at time t :

$M_i = 0$ *or* $D_{i-1} = 0$

comparisons :

place V of cell i contains 1 *at time* $t+1$
if and only if at time t :

$C_i = D_i$
$C_{i+1} = M_{i+1}$
$V_{i+1} = 1$



output :

1 *if* $C = M$ *and* $V = 1$

0 *otherwise*

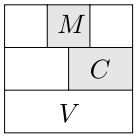
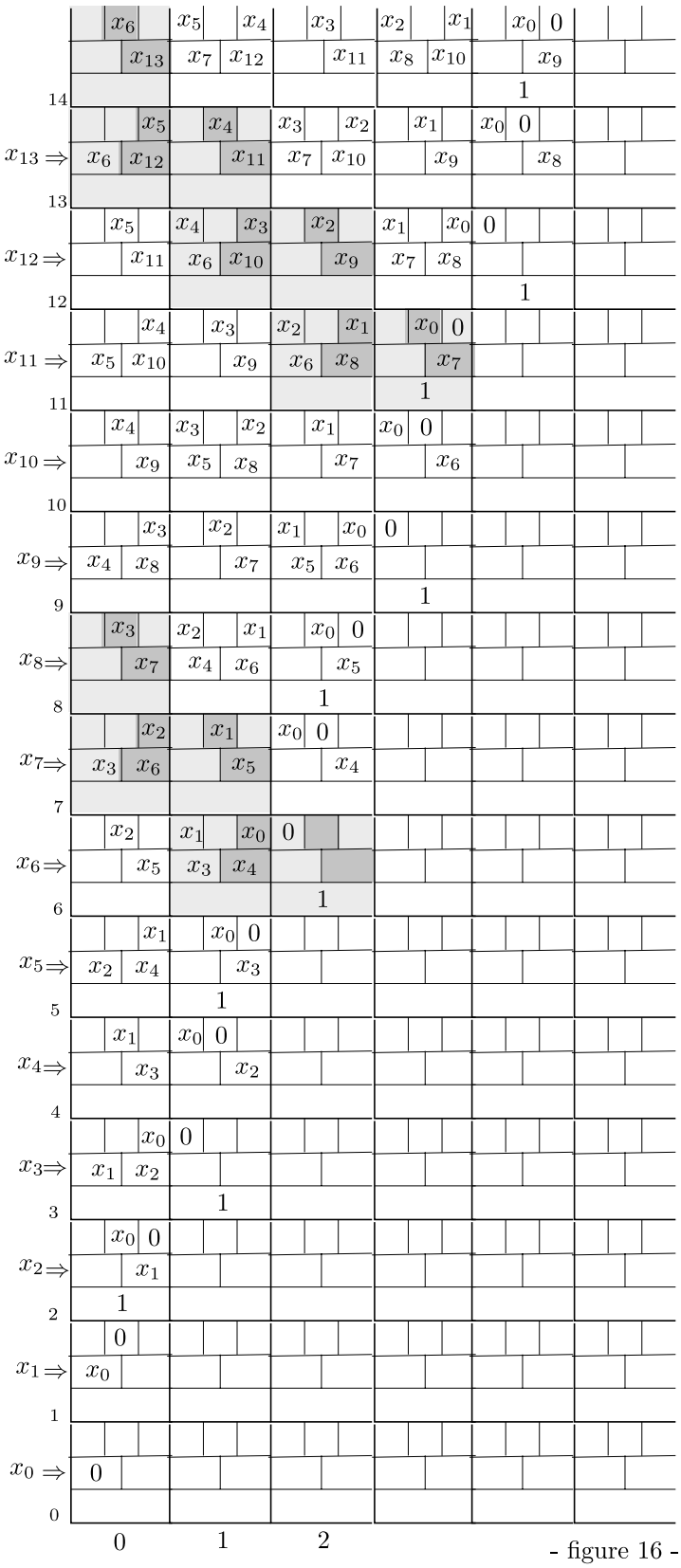


Fig.15


x_6 x_5 x_4 x_3 x_2 x_1 x_0 0
x_{13} x_7 x_{12} x_{11} x_8 x_{10} x_9
1
14
x_5 x_4 x_3 x_2 x_1 x_0 0
$x_{13} \Rightarrow$ x_6 x_{12} x_{11} x_7 x_{10} x_9 x_8
13
x_5 x_4 x_3 x_2 x_1 x_0 0
$x_{12} \Rightarrow$ x_{11} x_6 x_{10} x_9 x_7 x_8
1
12
x_4 x_3 x_2 x_1 x_0 0
$x_{11} \Rightarrow$ x_5 x_{10} x_9 x_6 x_8 x_7
1
11
x_4 x_3 x_2 x_1 x_0 0
$x_{10} \Rightarrow$ x_9 x_5 x_8 x_7 x_6
10
x_3 x_2 x_1 x_0 0
$x_9 \Rightarrow$ x_4 x_8 x_7 x_5 x_6
1
9
x_3 x_2 x_1 x_0 0
$x_8 \Rightarrow$ x_7 x_4 x_6 x_5
1
8
x_2 x_1 x_0 0
$x_7 \Rightarrow$ x_3 x_6 x_5 x_4
7
x_2 x_1 x_0 0
$x_6 \Rightarrow$ x_5 x_3 x_4
1
6
x_1 x_0 0
$x_5 \Rightarrow$ x_2 x_4 x_3
1
5
x_1 x_0 0
$x_4 \Rightarrow$ x_3 x_2
4
x_0 0
$x_3 \Rightarrow$ x_1 x_2
1
3
x_0 0
$x_2 \Rightarrow$ x_1
1
2
0
$x_1 \Rightarrow$ x_0
1
$x_0 \Rightarrow$ 0
0
0 1 2

- figure 16 -

Input word being $x_0 \cdots x_{l-1}$, with $l = 2p$ or $2p+1, x_i \epsilon X$, it remains for us to prove that output at time l, when x_{l-1} has entered, has value 1 if and only if

$$l = 2p \text{ and } x_0 = x_p \cdots x_n = x_{p+n} \cdots x_{p-1} = x_{2p-1}.$$

Better than do tedious though elementary calculations of the equations of the different signals involved, we shall simply observe Figure 16 : the 1-values are generated on signal 0. The only thing they can do is travel left at speed one (see Figure 15) if they don't extinguish. We first observe that they arrive in cell 1 at odd times, so output can be 1 only at even times, that is if l is even. Next, they will not extinguish and finally output will be 1 at time $l = 2p$ if and only if in the sites where they find themselves and in the site at the left, the coloured places contain the same input letters, and this resumes precisely in the condition we desired.

We now give the calculations for a reader which should not be satisfied :

- equation for signal 0 :

$$t = 3i + \begin{cases} 0 & \text{(position G or A)} \\ 1 & \text{(position M)} \\ 2 & \text{(position D)} \end{cases}$$

- first trajectory of x_n : $t = i + n + 1$
- meeting with signal 0

$$\text{signal 0 in position M, n is even: } < \frac{n}{2}, \frac{3n+2}{2} >$$

$$\text{signal 0 in position D, n is odd : } < \frac{n-1}{2}, \frac{3n+1}{2} >$$

- second trajectory of x_n : $t = -i + 2n + 1$
- arrival on cell 0 at time : $t = 2n + 1$
- third trajectory of x_n, at speed 1/3 :

$$t = 2n + 1 + 3i + \begin{cases} 0 & \text{(G or A)} \\ 1 & \text{(M)} \\ 2 & \text{(D)} \end{cases}$$

- here we note that output cannot be 1 if place M is empty while place C is occupied ; now then, place M can be occupied only by the trajectories of the x_n at times $2n + 1 + 1 = 2n + 2$, which are even. So outputs can have value 1 only at even times, after some word $x_0 \cdots x_{l-1}$ of even length $l = 2p$ has entered. To test this word we must compare

$$x_0 \quad x_1 \quad \cdots \quad x_n \quad \cdots \quad x_{p-1}$$

respectively with

$$x_p \;\; x_{p+1} \;\cdots\; x_{p+n} \;\cdots\; x_{2p-1}.$$

Comparisons $x_n \;\; x_{p+n}$ take place when first trajectory of x_{p+n} and third trajectory of x_n meet:

$$\text{for } n \text{ and } p \text{ of same evenness : } < \frac{p-n-2}{2}, \frac{3p+n}{2} >$$

$$\text{for } n \text{ and } p \text{ of different evennesses : } < \frac{p-n-1}{2}, \frac{3p+n+1}{2} > .$$

Let us point out here that the language of square words is not a context-free language, except indeed if alphabet has only one single letter.

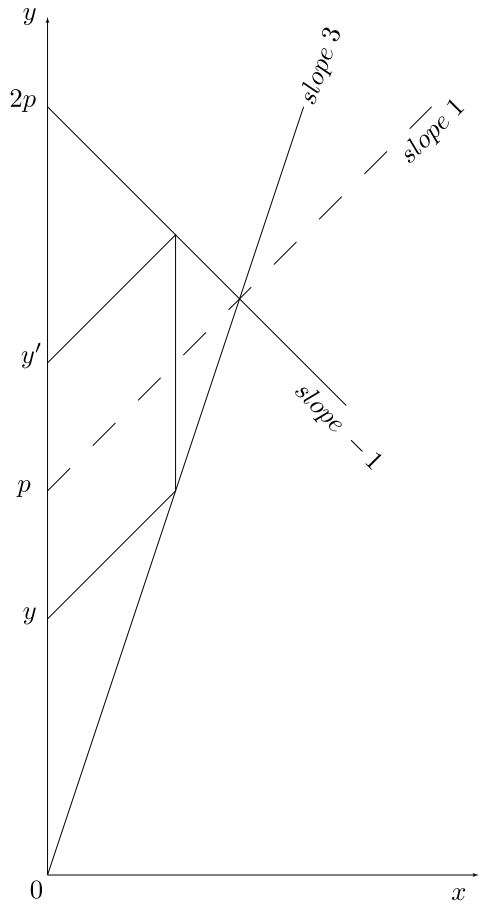


Fig.17

3.4.3 A c.a for the language of palindromes

Alphabet X is of course supposed to have at least 2 letters.

Figure which gives the idea is geometrical Figure 17 : from two points of the Oy axis having ordinates y and y' such that $(y + y')/2 = p$ are issued two lines of slope 1, the first one being broken by a reflection on the line from O of slope 3. They meet on the line of slope -1 from point of ordinate $2p$. As in the preceeding example, these two lines will be the pathes that the inputs will follow, the third line being the locus for the comparisons between them. The automaton is sketched in Figure 18.

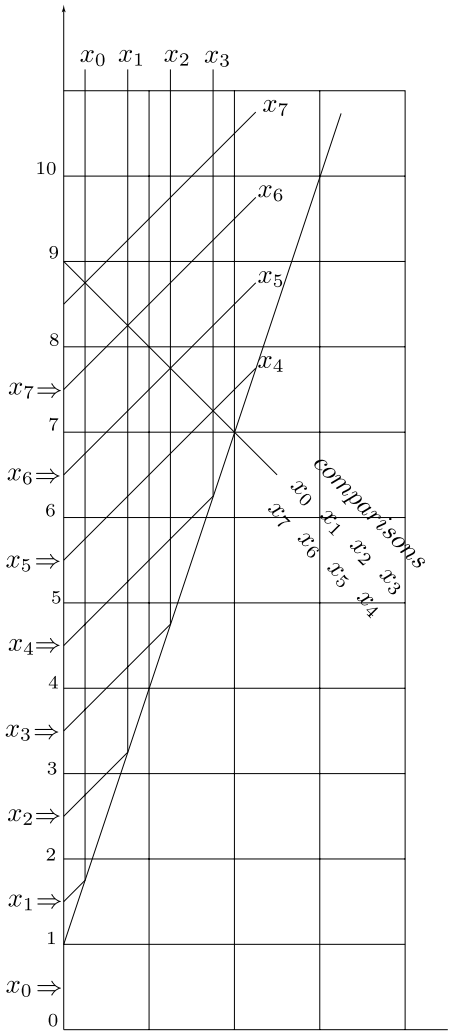
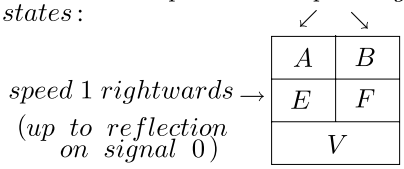


Fig.18

structure of states : *permanent parking places (except for 0)*

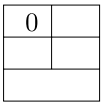


speed 1 rightwards → *(up to reflection on signal 0)*

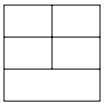
← *speed 1 rightwards (with extinction on quiescent or initial state)*

value of comparisons

initial state :

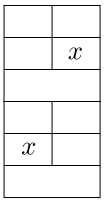


quiescent state :

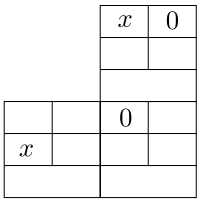


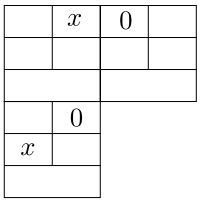
input on initial state : goes in place A , 0 pushed in B
on any other state : goes in place E

influence of left border on place E : pushes content in F



parking of inputs and progress of signal 0 :

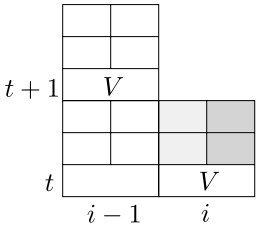




generation of $1-$values : on signal 0

comparisons :
place V of cell $i-1$ contains 1 at time $t+1$
if and only if at time t :

$A_i = E_i$ ← (*if a couple of input letters is there*)
$B_i = F_i$ ←
$V_i = 1$



output :
1 *if* $A = E$ ← (*if a couple of input letters is there*)
$B = F$ ←
$V = 1$
0 *otherwise*

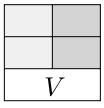


Fig.19

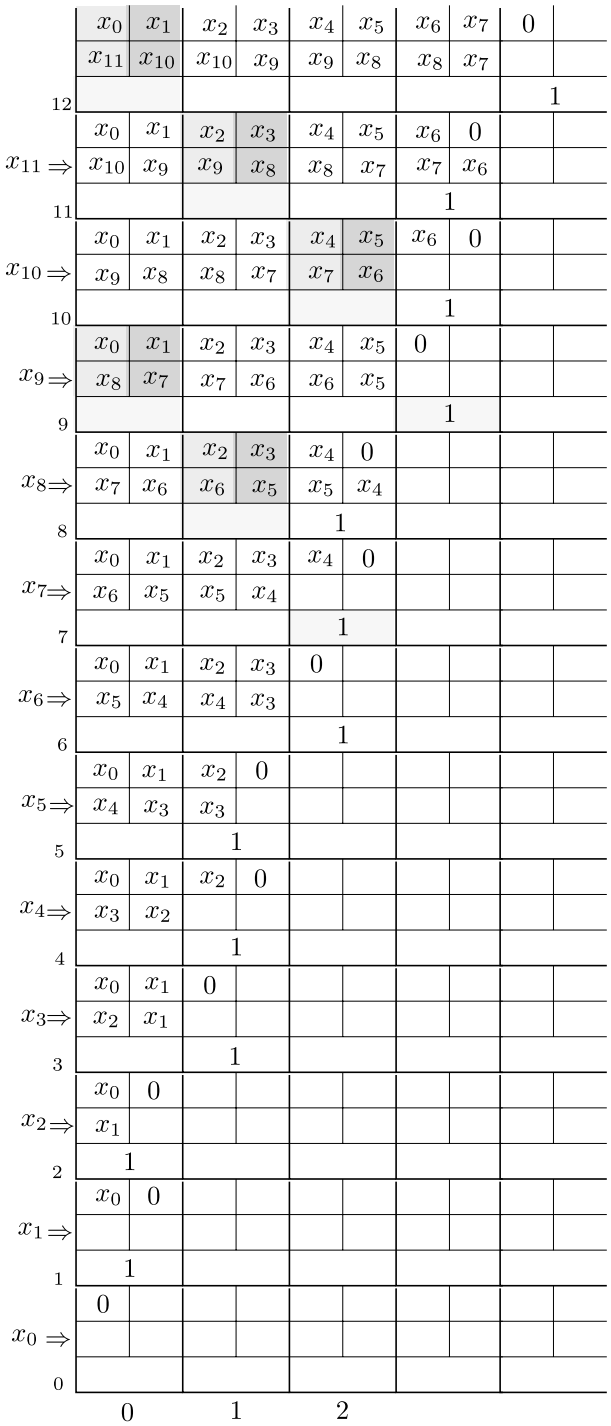

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 0
x_{11} x_{10} x_{10} x_9 x_9 x_8 x_8 x_7
12
1
x_0 x_1 x_2 x_3 x_4 x_5 x_6 0
$x_{11}\Rightarrow$ x_{10} x_9 x_9 x_8 x_8 x_7 x_7 x_6
11
1
x_0 x_1 x_2 x_3 x_4 x_5 x_6 0
$x_{10}\Rightarrow$ x_9 x_8 x_8 x_7 x_7 x_6
10
1
x_0 x_1 x_2 x_3 x_4 x_5 0
$x_9\Rightarrow$ x_8 x_7 x_7 x_6 x_6 x_5
9
1
x_0 x_1 x_2 x_3 x_4 0
$x_8\Rightarrow$ x_7 x_6 x_6 x_5 x_5 x_4
8
1
x_0 x_1 x_2 x_3 x_4 0
$x_7\Rightarrow$ x_6 x_5 x_5 x_4
7
1
x_0 x_1 x_2 x_3 0
$x_6\Rightarrow$ x_5 x_4 x_4 x_3
6
1
x_0 x_1 x_2 0
$x_5\Rightarrow$ x_4 x_3 x_3
5
1
x_0 x_1 x_2 0
$x_4\Rightarrow$ x_3 x_2
4
1
x_0 x_1 0
$x_3\Rightarrow$ x_2 x_1
3
1
x_0 0
$x_2\Rightarrow$ x_1
2
1
x_0 0
$x_1\Rightarrow$
1
1
0
$x_0\Rightarrow$
0
0 1 2

Fig.20

Here states belong to $[X \cup \{0\} \cup \{\emptyset\}]^4 \times \{0, 1, \emptyset\}$, they are a bit simpler than Cole's ; all information about the rules (which are not explicitly written as such) is given in Figure 19, so we could write them, and we can build the s.t.d (see Figure 20).

Computation of the trajectories and their meeting points are simpler here than in preceeding example, they are of no particular interest.

We shall now give several variants of this c.a.

First variant for accepting only even length palindromes we shall simply generate 1-values on signal 0 only when this signal is in place A.

If on the contrary we want only the odd length palindromes, we shall generate the 1-values on signal 0 only when it is in place B.

Second variant we now want to recognize P_2X^*, the set of words beginning with a palindrome, of length 2 at least (otherwise we should have all non empty words !)

We need only suppress the first 1-value, this to eliminate palindromes x_0, and decide that in cell 0 any state having output 1 produces a permanent 2-value one unit of time later, output being 1 if we have a 2-value.

Third variant if we want to recognize P_3X^*, the set of words beginning with a palindrome of length at least 3, we do the same after having suppressed the first two 1-values.

3.4.4 A c.a for language a^*P_3

In the next section we shall prove that language X^*P_3 is not recognizable by any c.a in strict real time. Therefore it is not uninteresting to show that a^*P_3, where a is any letter of X, is recognizable in strict real time.

To begin with, note that

$$\begin{array}{lcl} a^*P & = & a^* \cup \{a^m u a^n \mid u \epsilon P, m \geq n\} \\ a^*P_2 & = & a^* \cup \{a^m x a^n \mid x \neq a, m \geq n \geq 1\} \cup \{a^m u a^n \mid u \epsilon P_2, m \geq n\} \\ a^*P_3 & = & a^* \cup \{a^m x a^n, a^m x x a^n \mid x \neq a, m \geq n \geq 1\} \cup \{a^m u a^n \mid u \epsilon P_3, m \geq n\} \end{array}$$

where u is a word starting and ending with a letter different from a.

We shall now build a c.a recognizing

$$\{a^m u a^n \mid u \epsilon P, m \geq n\},$$

where u starts and ends with a letter different from a.

In this construction, P may be replaced by P_2 or P_3. We may then conclude that the 3 preceding languages are recognizable in strict real time by using a general and easy result given in section 10.5.5, which states in particular, that the family $\mathcal{L}_1$ of languages recognized in strict real time by one-dimensional c.a's is closed under union.

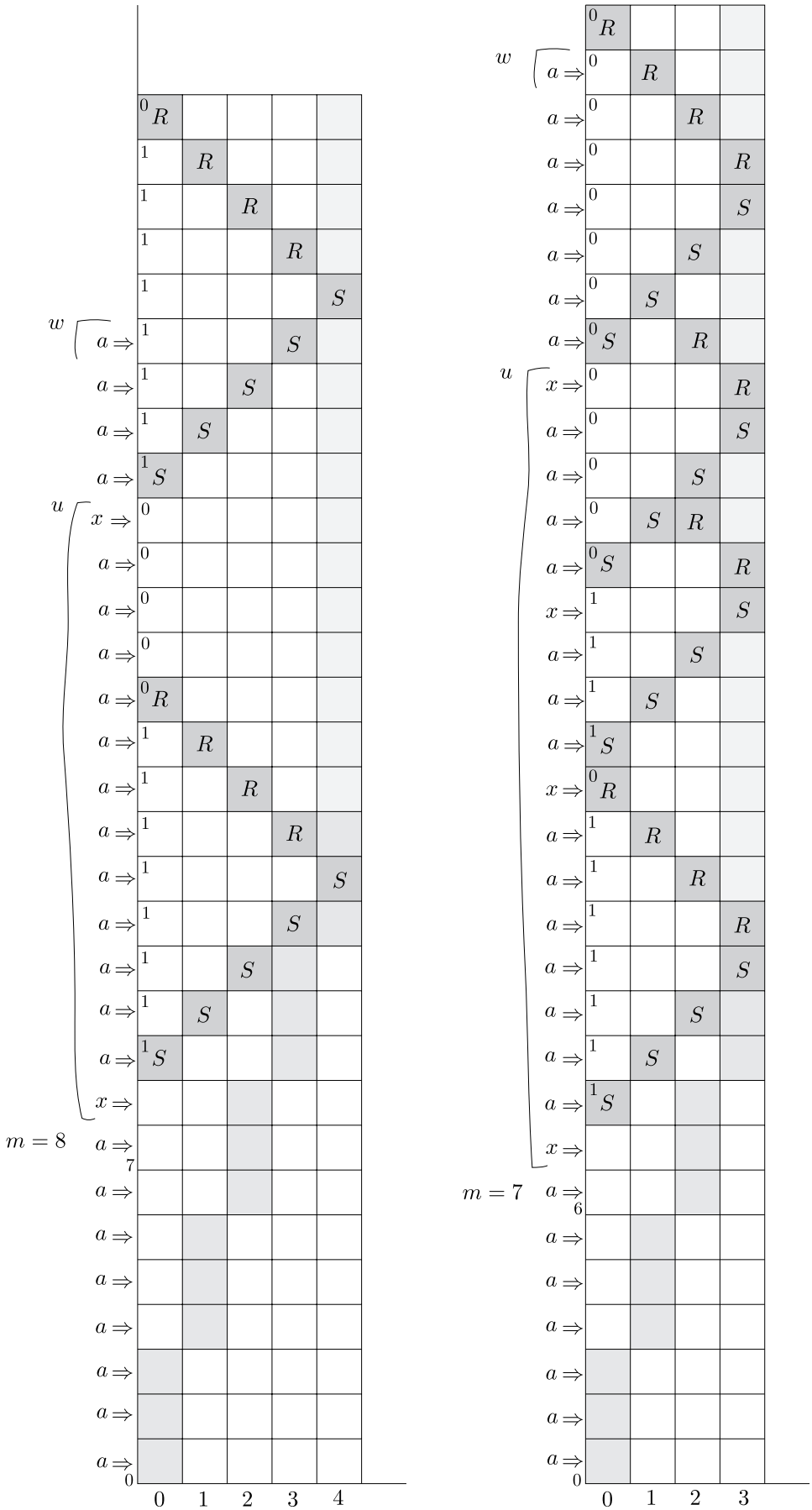


Fig.21

The idea for this c.a is that input of the first letter not an a starts the c.a recognizing P. At each time-step, this c.a tells if the beginning of word u is in P or not, and after the last letter not an a has entered, if u itself is in P. To the states of this c.a we add a little memory, which will be updated after every input of a letter other than a, to value 1 if state is accepting, to value 0 otherwise (see Figure 21).

But we must further check that the n last a letters are less than the m first a letters. For this, at time 0 a signal of slope 3 is emitted, ($t = 3c + 0, 1, 2$) and input of the first letter not an a starts a signal of slope 1 (t=m+1+c). These signals meet on cells

$$\begin{cases} c = & \frac{m}{2} & \text{if m is even} \\ \\ c = & \lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{2} \rfloor + 1 & \text{if m is odd} \end{cases}$$

Cell $\lfloor m/2 \rfloor$, where signals first meet, is marked, (differently if m is odd or even). Now, a speed 1-signal starting from cell 0 at time t and reflected on cell c comes back on cell 0 at time $t+2c$ (fast reflection) or $t+2c+1$ (slow reflection). If the last letter of word u sends such a signal S, at the same time as the first of the n last a letters is input , its reflection R, fast in case m is even, slow in case m is odd, comes back on cell 0 m time-steps later. For word w to be accepted, last letter a must be input before R has come back. So we decide that the return of R on cell 0 resets the memory to value 0.

In fact all letters not a will send S signals, because it is not known if such a letter is not the last one. But if new signals S stop signals R, then at the end only the last signal R comes back on cell 0.

Finally, w is accepted if memory has value 1 when its last letter is input . In Figure 21, the first w word is accepted, and the second one is rejected because u is not a palindrome.

3.4.5 No c.a recognizes X^*P_3 in strictly real time

Proof is Cole's [6].

We shall simply show that certain $\equiv_k^{X^*P_3}$ equivalences have too many classes. Having this aim in view we shall build, with some elaborate work we must say, an interesting lot of words of X^*P_3, numerous and not equivalent.

Alphabet X has at least two letters, otherwise talking of palindromes would be nonsense. Let these two letters be denoted 0 and 1, the other letters if there be some we shall not use.

Let m be a positive integer which we intend to choose later on, and V be the set of words on {0,1} of the form

$$v = 1x_1 1x_2 \ldots\ldots 1x_m 100 \quad \text{where} \quad \forall i \;\; x_i = 0 \text{ or } 1$$

There are 2^m of these words, and they all have the same length $k = 2m + 3$. The number of subsets of V is 2^{2^m}, exponential in m : now then, for each of

these subsets

$$P = \{v_1, v_2, \ldots\ldots, v_p\}$$

we shall build a word w_P belonging to X^*P_3. First of all we order the v_i's, arbitrarily, (for example in lexicographical order), so as to have a sequence

$$(v_1, v_2, \ldots\ldots, v_p).$$

Then we progressively define the new sequence of words

$$w_0 = \text{the empty word} \quad (\text{if } P \text{ is empty we stop here})$$

$$w_1 = \tilde{v}_1$$

$$w_2 = \tilde{v}_2\tilde{w}_1 w_1 = \tilde{v}_2 v_1 \tilde{v}_1$$

$$\ldots\ldots\ldots$$

$$w_{i+1} = \tilde{v}_{i+1}\tilde{w}_i w_i$$

$$\ldots\ldots\ldots$$

$$w_p = \tilde{v}_p\tilde{w}_{p-1}w_{p-1}.$$

Each word w_i is a suffix of the next ; and each w_i is formed, as is easy to see by induction, of an odd number of length k-words which are alternatively $\tilde{v}_i$'s and v_j's of P. The w_P corresponding to subset P will be the last word we obtain, w_p. Now we shall observe two things concerning the words w_P :

- observation 1 : if $v_i \in P$ then $w_P v_i \in X^*P_3$.

Indeed, $w_P v_i$ has the suffix $w_i vi$ which is a palindrome of length at least 3.

- observation 2 : if $v \in V - P$ then $w_P v \notin X^*P_3$.

If w_P is the empty word, $w_P v = v$. This word ends with palindrome 00, of length 2, but, ending with 100, by no palindrome of length 3, and by no longer palindrome because the only 00 it contains is the one at the end.

If w_P is not empty,

$$w_P v = \tilde{v}_{i_1} v_{i_2} \tilde{v}_{i_3} v_{i_4} \ldots \tilde{v}_{i_r} v.$$

Because of the arrangement of the 00's, if $w_P v$ ends with a palindrome, $\tilde{v}$ must be one of the $\tilde{v}_i$'s, and this precisely is impossible as we have taken v in $V - P$.

From the two preceeding observations results that two different subsets P and P' produce words that belong to different $\equiv_k^{X^*P_3}$ classes : indeed, $P \neq P'$ implies the existence of some v of P not belonging to P', (or vice versa), and then $w_P v \in X^*P_3$ and $w_{P'} v \notin X^*P_3$. So equivalence $\equiv_k^{X^*P_3}$ has at least 2^{2^m} classes. But, for any $h \geq 2$, if we choose m such that

$$2^m > (2m+4) log_2 h,$$

we have

$$2^{2^m} > h^{2m+4} = h^{k+1}$$

which, according to Cole's criterion, excludes that X^*P_3 could ever be recognized by some c.a in strictly real time.

We could prove in exactly the same way that language X^*P_2 cannot be recognized by any c.a in strictly real time, but we must have at least three letters, 0, 1 and 2 in alphabet X. For V we take words of the form

$$x_1 x_2 \ldots\ldots x_m 2 \quad \text{where} \quad x_i \in \{0,1\}$$

and we end by choosing m such that

$$2^{2^m} > h^{k+1} = h^{m+2}.$$

3.4.6 languages P, P_3X^* and X^*P_3 are accepted by parallel c.a's in large real time

Idea of the c.a is due to Smith [51], and we suggest it in Figure 22 : each input letter is sent at maximal speed 1 in the two directions, inputs are compared when they meet, if they differ a permanent # is cast on the cell. If a # sign is found when signals 0 and $*$ meet, the input word is not a palindrome.

There is no managing without signal $*$ to determine the middle of the word where a possible # must be detected, so strictly real time is impossible with this method : this confirms what we had announced in general for parallel recognizing.

Figure 23a gives indications on states and rules, and an example is given in Figure 23b.

With minor changes we can construct c.a's recognizing X^*P_2 and P_2X^*. For X^*P_2, at least one of the comparison lines of the right half of the input word must arrive without a # on the $*$ signal, thus indicating that some suffix of the word is a palindrome : this event will transform $*$ in a protected $\circledast$ which later crosses signal 0 without damage. We must not forget to discard length 1-palindromes by deciding that two $*$ in places A and B produce a # on places M and D of cell at the left one unit of time later. Figure 24 illustrates these little changes, for the sake of clearness the sites are not completely filled.

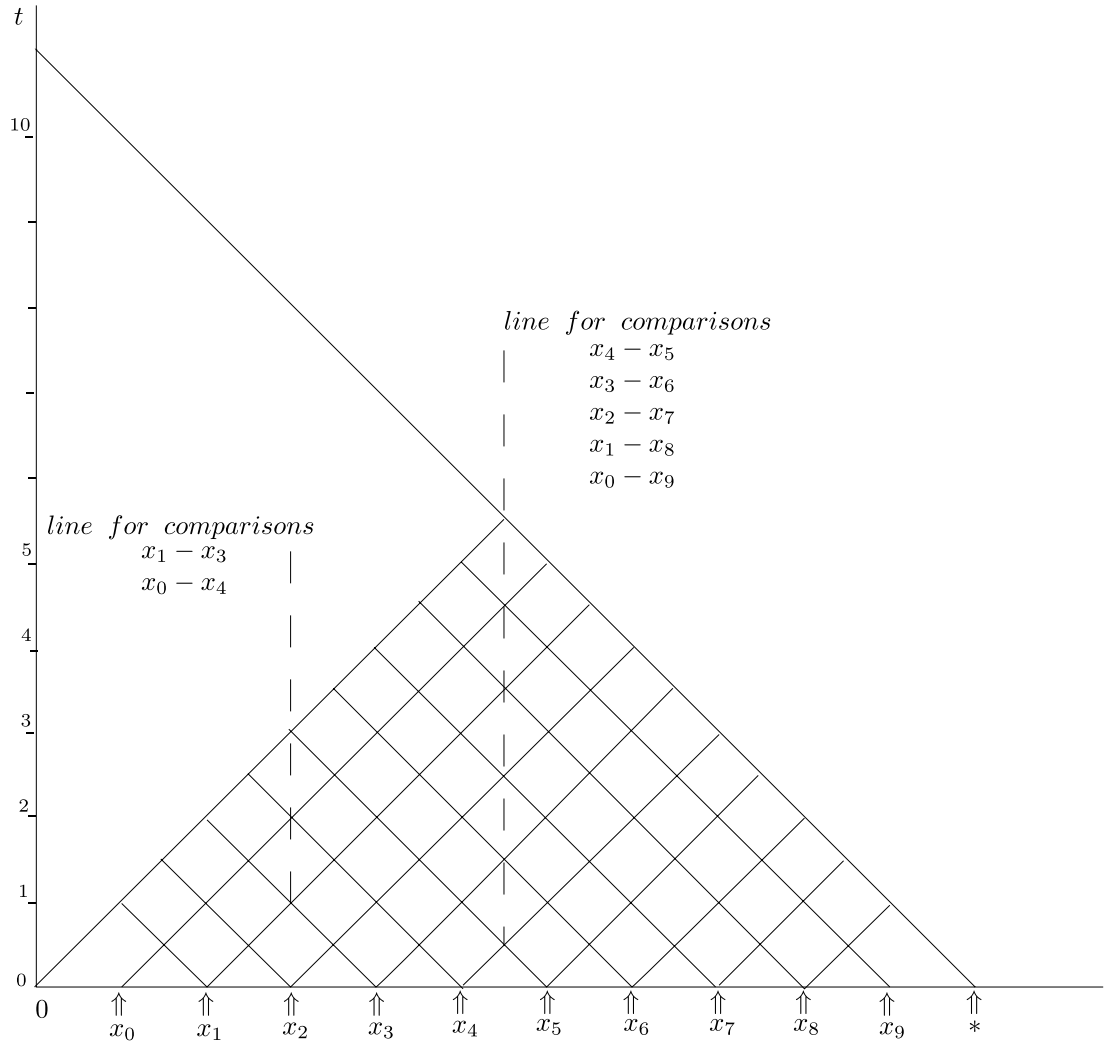
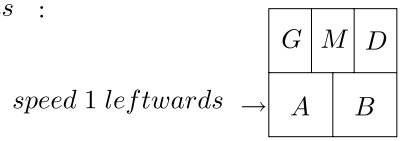

t
10
5
4
3
2
1
0
line for comparisons
$x_4 - x_5$
$x_3 - x_6$
$x_2 - x_7$
$x_1 - x_8$
$x_0 - x_9$
line for comparisons
$x_1 - x_3$
$x_0 - x_4$
0
x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
$*$

Fig.22

structure of states :

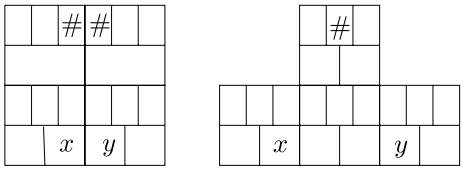


initial state : 0 *in* G , (*to pass in* M *then in* B)

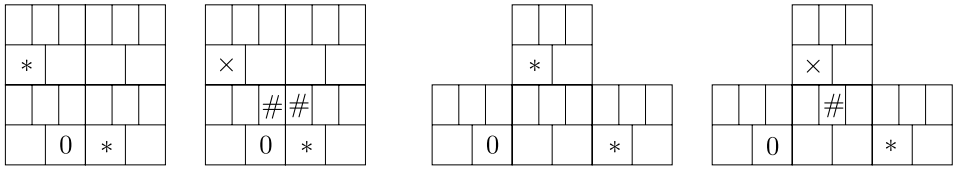
inputs : *go in* A *and* B

generation of permanent # *by bad comparisons* :

if $x \neq y$



$*$ *crosses* 0 *only if no* # *on the way* *otherwise becomes* $\times$:



output : 1 *if* A *contains* $*$
0 *if* A *contains* $\times$

Fig.23a

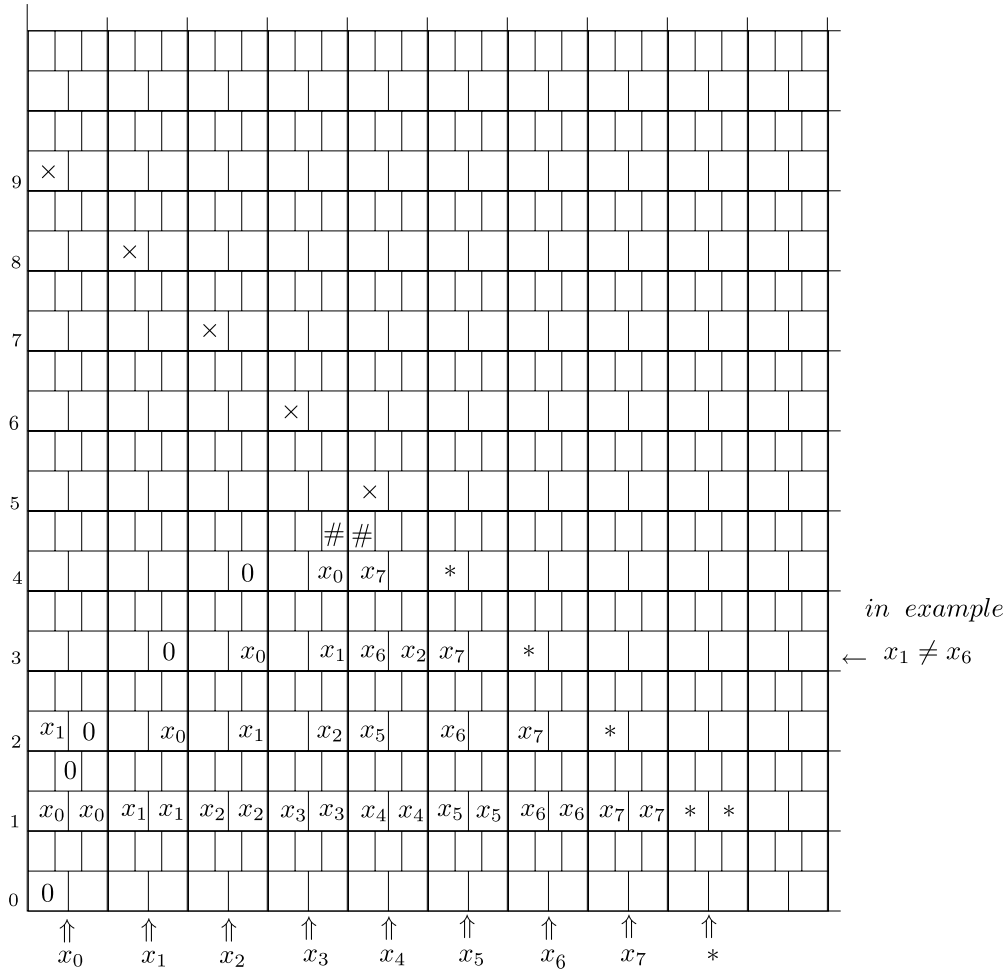


Fig.23b

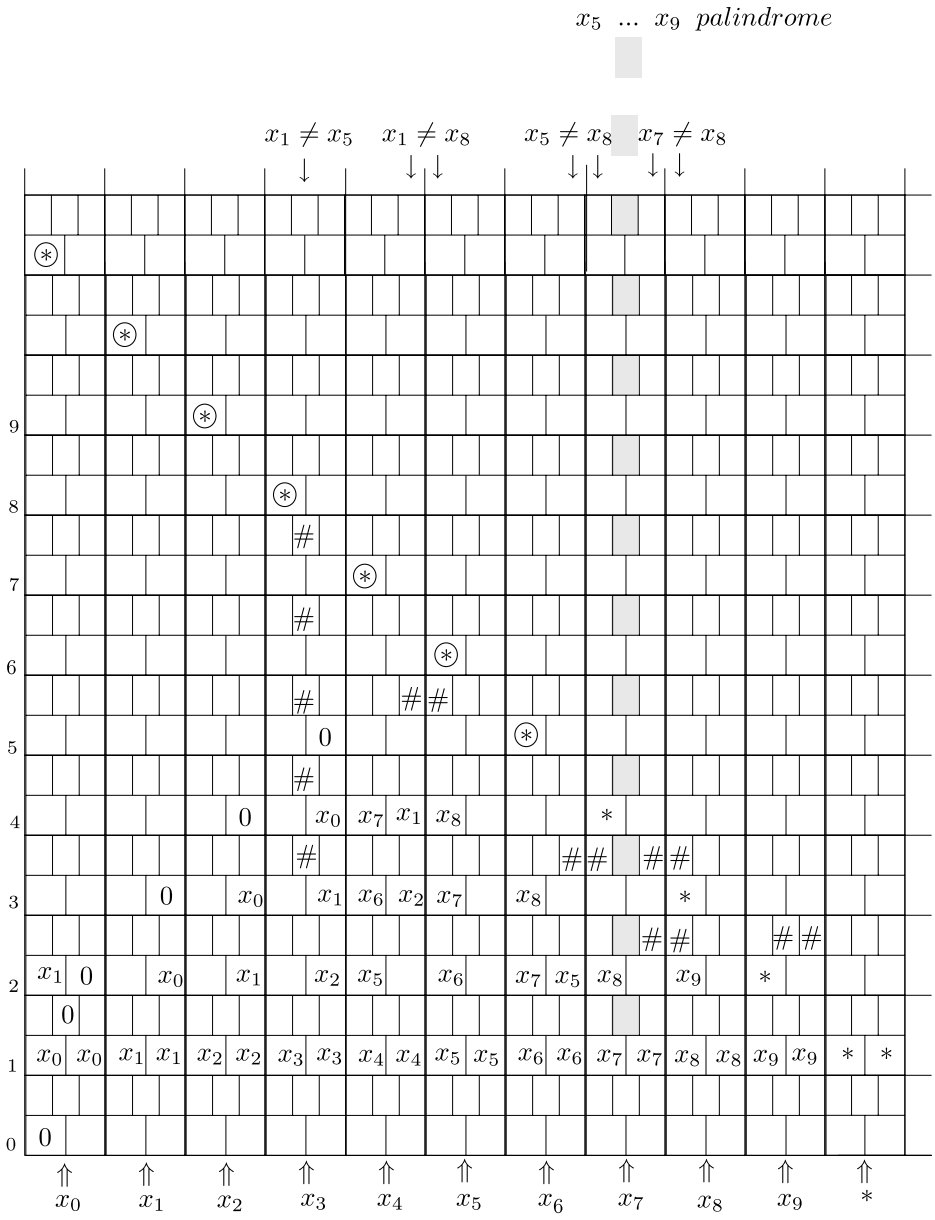


Fig.24

For P_2X^*, at least one of the comparison lines of the left half of the input word must be free of #, thus indicating that some prefix of the word is a palindrome : it suffices to decide that crossing such a line restores $\times$ in $*$. To

discard length 1-palindromes, we decide that initial state produces 2 units of time later two # symbols on places G and M of cell 0.

If we want c.a's for X^*P_3 and P_3X^* : we must discard not only length 1, but also length 2-palindromes. for X^*P_3 we can decide that the two # symbols of time 1 produce 4 such symbols in the cell left, and one will be pushed in place D in next left cell one unit of time later ! For P_3X^* we decide that initial state produces 2 units of time later three # symbols in cell 0 and one in place G of cell 1. Figure 25 gives an example for P_3X^*, here again sites are not completely filled.

To finish with, let us point out that languages X^*P_2 and X^*P_3, as they can be recognized by parallel c.a's in large real time, can as a consequence be recognized by sequential c.a's in time

$$n + 1 \ + \ 3n - 1 \ = \ 4n.$$

Let us resume results in a small table :

language	recognizing time T(n)	parallel recognizing time
P	n	$n+1$
P_3X^*	n	$n+1$
X^*P_3	$4n$	$n+1$

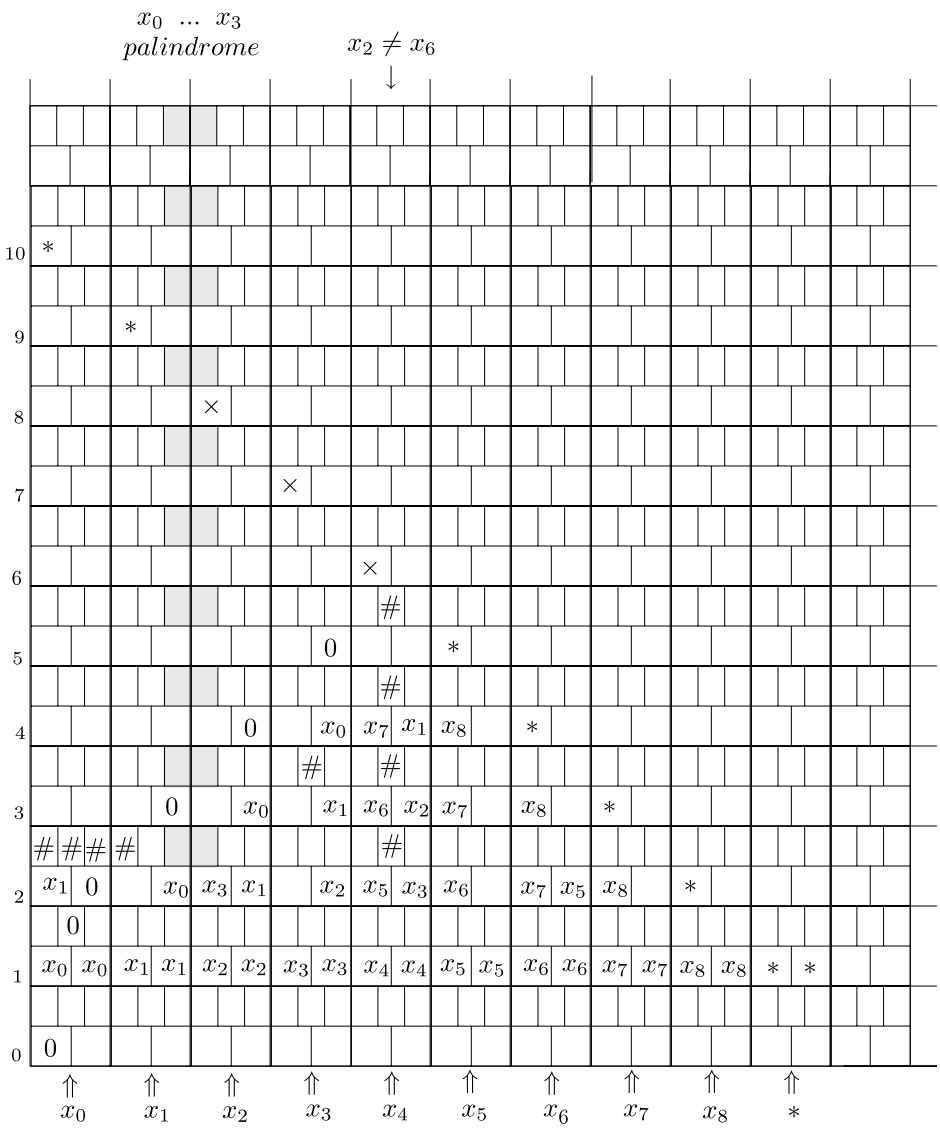


Fig.25

3.5 A particular case : treillis automata

We find it more convenient in this section to consider a bi-infinite c.a, that is to say a line.

Two ways for presenting treillis automata seem possible. The first one gives them a different structure, reducing the neighbourhood of a cell to its two left

and right neighbours, the cell itself being excluded.

The second, which is in fact exactly equivalent, presents them simply as a particular case, where the transition functions do not depend on the central state :

$$\delta_0(x,l,q,r) = \gamma_0(l,x,r)$$

$$\delta(l,q,r) = \gamma(l,r).$$

The dependance and influence triangles, represented in Figure 26, are full of holes, like fretwork or lace. Two completely independant s.t.d's appear, the one of even $(c+t)$ sites, (working with the odd time inputs),

$$(0,0) \quad (2i,2t) \quad (2i+1,2t+1) \quad i \in \mathbb{Z}, t \in \mathbb{N}$$

and the one of odd sites (working with the even time inputs)

$$(1,0) \quad (2i-1,2t) \quad (2i,2t+1) \quad i \in \mathbb{Z}, t \in \mathbb{N}.$$

If, for the second diagram, we take time 1 for starting time, we have the same notations as with the first diagram, and they are more pleasant because more symmetrical.

Let us examine such an s.t.d (Figure 27) : inputs are $x_1, x_3, \ldots$. If we concentrate on the even times, we shall write

$$< c, t+2 >= \gamma(< c-1, t+1 >, < c+1, t+1 >) =$$

$$= \gamma\left(\gamma(< c-2, t >, < c, t >), \gamma(< c, t >, < c+2, t >)\right)$$

$$< 0, t+2 >= \gamma_0(< c-1, t+1 >, x_{t+1}, < c+1, t+1 >) =$$

$$= \gamma\left(\gamma(< c-2, t >, < c, t >), x_{t+1}, \gamma(< c, t >, < c+2, t >)\right)$$

which leads us back to an ordinary c.a having transitions

$$\Delta(l,q,r) = \gamma(\gamma(l,q), \gamma(q,r))$$

$$\Delta_0(x,l,q,r) = \gamma_0(\gamma(l,q), x, \gamma(q,r)).$$

So a treillis automaton has this curious property that its working splits in two processes, altogether completely independant and playing alternately on the same cells. These processes in turn are c.a processes with an intermediate step. This way of working seems more intriguing than useful. We do not see why we should study them as such. They seem to us simply c.a's with diminished possibilities.

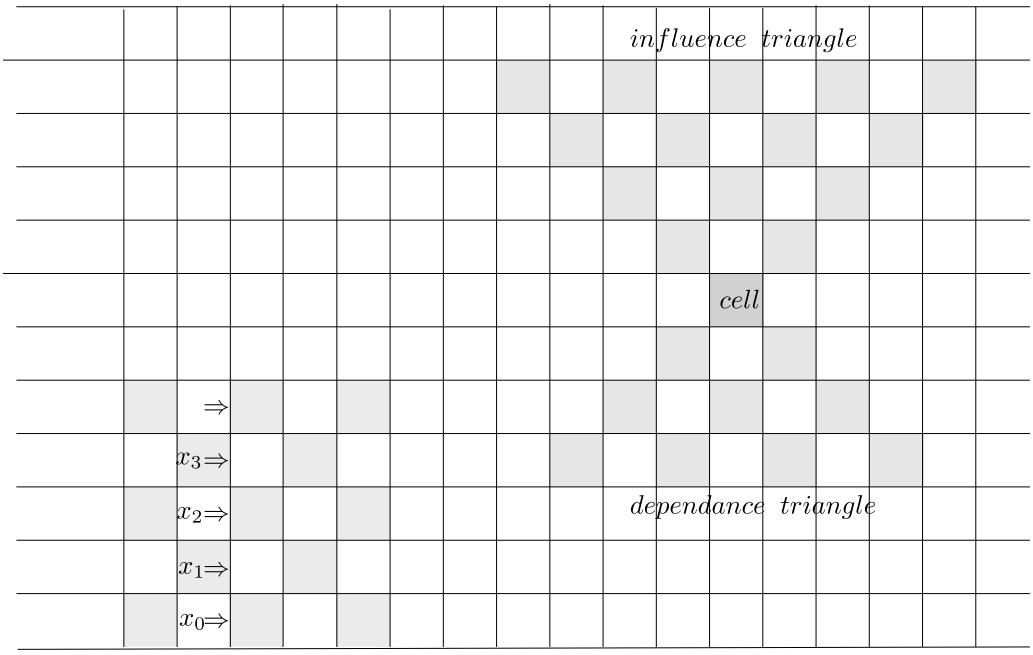

influence triangle
cell
⇒
x_3⇒
x_2⇒
dependance triangle
x_1⇒
x_0⇒
0

Fig.26

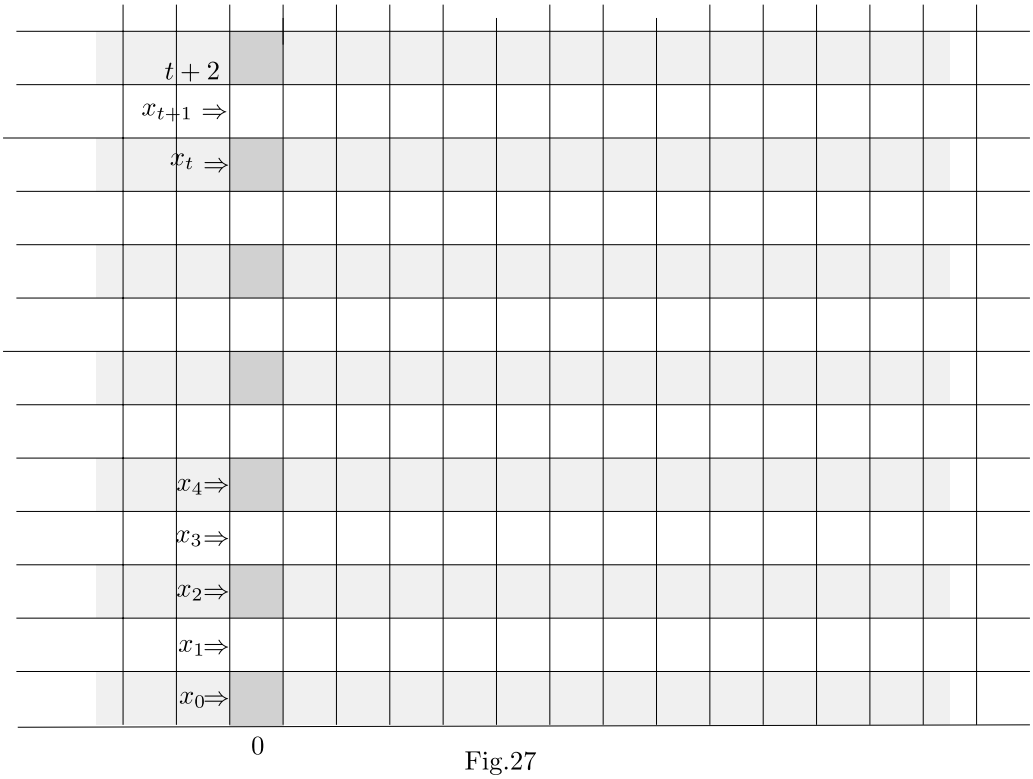

$t+2$
x_{t+1} ⇒
x_t ⇒
x_4⇒
x_3⇒
x_2⇒
x_1⇒
x_0⇒
0

Fig.27

Chapter 4

Comparison with Turing machines

We shall use abbreviation Tm for Turing machine.

Conclusions of this chapter are perfectly well known : c.a's do all that Tm's do, and nothing more, (so they also do the same as computers), but faster. With a little common sense and intuition we would readily admit it, and each of us can get convinced by sketching proofs for oneself. If we attempt here to write them out, it is only for the sake of completeness. Results are fundamental, but never shall we transform a Tm to obtain a c.a, nor a c.a to obtain a Tm ! We shall moreover notice that the most natural constructions give inadequate and ridiculous machines, this because the parallel functioning of c.a's is profoundly different from the linear one of Tm's.

We have in mind machines which recognize and machines which compute. We shall give proofs for recognizers and point out how to complete them for computers.

4.1 Simulation of a Tm by a c.a

First thing, we must choose our model of c.a and our model of Tm. For the c.a we do not hesitate, it will be the one described in chapter 3, our reference. As for the Tm, we know that all models are equivalent and we can transform the one in the other, so we shall choose the machine nearest possible to our c.a, with :

- a semi-infinite tape, with squares 0, 1, 2, ...

- halting by halting state on square 0, that is at the beginning of the tape

- a blank character b, which is not the same as the empty symbol (denoted by $-$, or simply nothing in figures) initially covering the

tape, which the head can not write, and so is to be found where the head never yet reached

- for computers, an end marker *.

The idea

is very natural : at each instant of time the c.a will reproduce the tape of the Tm, with the state of the head added on the cell corresponding to the square where the head is (Figure 1).

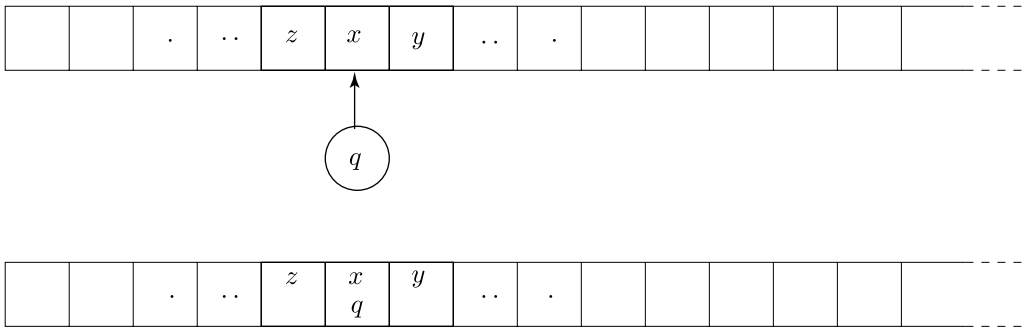


Fig.1

Its set of states will thus be

$$X \cup Q \times X$$

where X is the tape alphabet, including the empty symbol, and Q is the set of states of the Tm.

At each time step the c.a will reproduce one move of the Tm. This is possible because for the Tm, when we know what concerns three consecutive squares, we can deduce all that concerns the middle one at next step. Only the cells neighbouring the only cell having a state of the form (q, x) may change state. We shall not write the transitions which are obvious translations of the Tm rules.

The problem of the inputs :

The c.a must recognize (or not), a word $(x_0, \ldots, x_{n-1})$ deposited at time 0 on the tape of the Tm (Figure 2). This word must be given as input for the c.a, and it appears clearly that a parallel input suits us better for the present purpose. So we shall construct a parallel input c.a, that we know from chapter 3 how to transform afterwards in a sequential c.a.

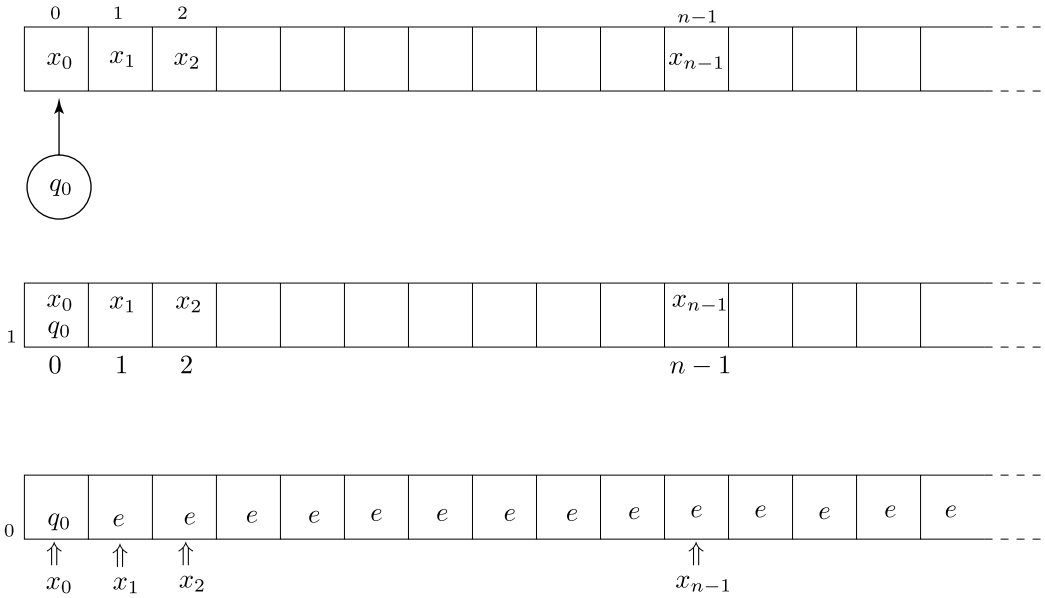


Fig.2

Initial state of the c.a will be (Figure 2)

$$(q_0, -).$$

We must add transition rules :

$$\delta^0(\beta, (q_0, -), e, \text{input } x) = (q_0, x)$$

$$\delta^0((q_0, -), e, e, x) = x$$

$$\delta^0(e, e, e, x) = x.$$

Halting and halting time

The halting states (resp. accepting/rejecting halting states) of the c.a will be all the products of a halting state (accepting state, rejecting state) of the Tm with a letter of alphabet X (Figure 3).

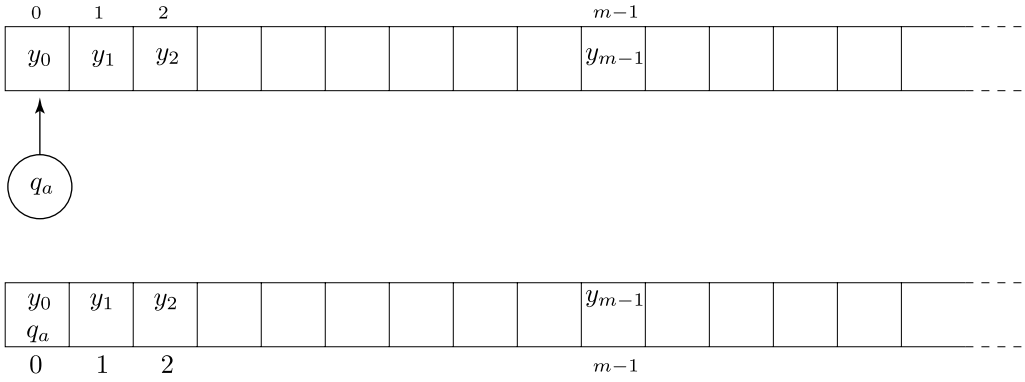


Fig.3

As our Tm halts with head on square 0, if the time it needs for recognizing input words of length n is $T(n)$, the recognizing time T_{ap} for the parallel c.a will be

$$T_{ap}(n) = 1 + T(n).$$

We can consider it is the same.

But if we transform the parallel c.a in a real sequential c.a, the recognizing time T_{as} will be (see 3.3.3)

$$T_{as}(n) = T_{ap}(n) + 3n - 1 = T(n) + 3n.$$

Now we have admitted once and for all, and for any machine (see 3.2.2) that

$$T(n) \geq n$$

so

$$T_{as}(n) \leq 4T(n).$$

The recognizing time of the c.a does not exceed a linear function of the recognizing time of the Tm.

For a computing Tm

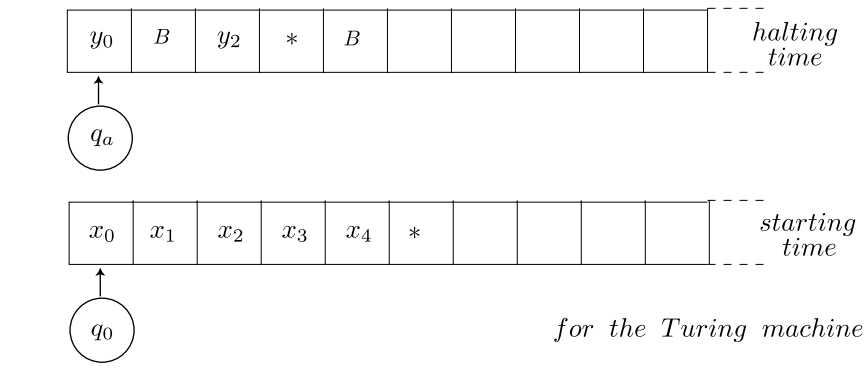
We shall include end marker * in the alphabet, the total length of the input word, end marker included will be n

$$(x_0, x_1, \ldots, x_{n-2}, *)$$

and length of the resulting word will be m

$$(y_0, y_1, \ldots, y_{m-2}, *).$$

We shall complete our c.a by the following rules (see Figure 4)


y_0 B y_2 $*$ B
halting time
q_a
x_0 x_1 x_2 x_3 x_4 $*$
starting time
q_0
for the Turing machine

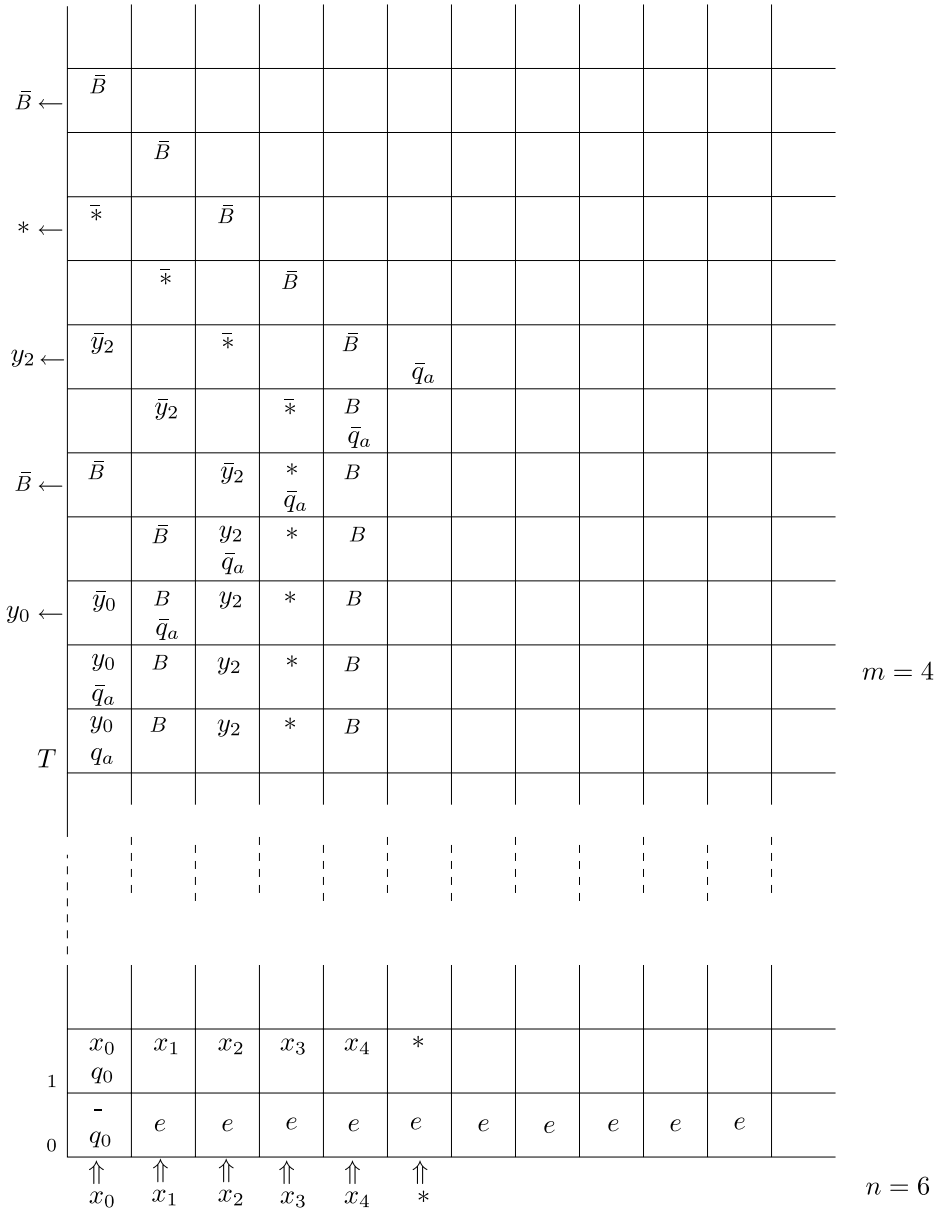

$\bar{B} \leftarrow$
$* \leftarrow$
$y_2 \leftarrow$
$\bar{B} \leftarrow$
$y_0 \leftarrow$
T
$m = 4$
$n = 6$
1
0

Fig.4

- state q_a against the border becomes $\bar{q}_a$

- state $\bar{q}_a$ goes rightwards at maximal speed, changing y_i into $\bar{y}_i$ and $*$ into $\bar{*}$

- state $\bar{q}_a$ extinguishes when it meets the empty symbol (not the blank symbol)

- barred letters travel left at maximal speed and are output without a bar when they arrive in cell 0.

This output episode takes time $2m$. Let us show that

$$2m \leq T(n) + 2.$$

Indeed, to pass over k squares and come back to square 0, the head takes at least time $2(k-1)$. Now, in case $m \geq n$ the head has certainly gone over m squares. And in case $m < n$, as we have agreed in 3.2.2 that input word must be entirely read, the head has passed over more than m squares. We could also require, as a variant, that after the end marker only blanks or empty symbols should be found, which would imply that the head has passed over the input word to erase it.

So in the end we have

$$T_{as}(n) \leq 4T(n) + T(n) + 2 \approx 5T(n)$$

the same sort of result for computing machines as we had for recognizers.

Conclusion

For every Tm, there exists a c.a halting (or not) for the same input words, recognizing the same words or computing the same results, in time not exceeding a linear function of the time taken by the Tm.

Let us note that this c.a uses exactly the same space as the Tm, indeed the number of active cells equals the number of tape squares used.

4.2 Simulation of a c.a by a Tm

Here Q and X will be the set of states and input alphabet of the c.a, and δ its transition function, and we decide that the c.a halts by halting state.

For things to be as clear as possible, we impose that state e of the c.a cannot appear on a cell which is not already in this state, in the same way as the empty state of the Tm could not be written by the head. If necessary a second "new" quiescent state may be introduced : now the quiescent area of the c.a will be perfectly distinct, it will limit the active area on its right, and with time can only shrink (Figure 5).

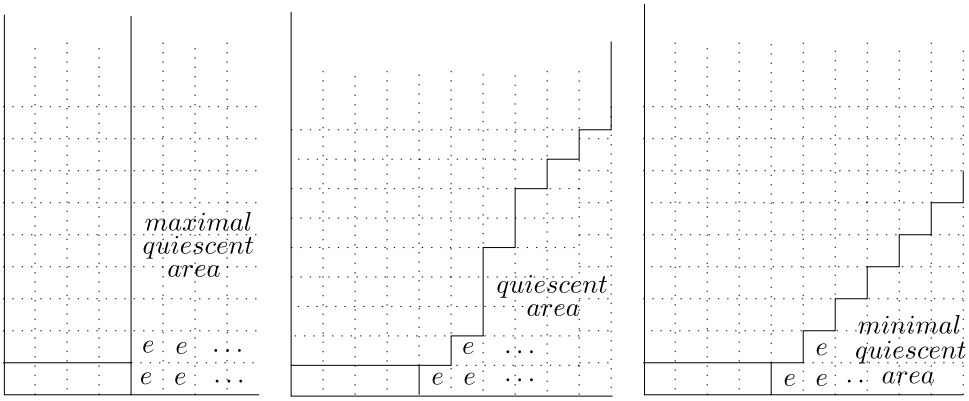


Fig.5

Let us mention now that the machine we shall construct will have rules with no move of the head (0-move), and let us explain why, because it seems quite stupid at first, to change state two times and write a symbol that will imediately be rewritten on. The point is that these symbols written in the 0-moves by our machine will be the end states of the lines of active cells of the c.a, so we do not want the 0 moves to disappear into next move. If we absolutely want a usual Tm, we can replace the 0-moves by two moves, right and back, or left and back, which increases the number of steps by one at each time.

The idea is :

the Tm having on its tape a configuration of the c.a with the head at one end, will compute the next configuration in one passage forward or backward, by memorizing at each step two states and reading the third state. So the head will shuttle between the left border and the quiescent area.

Computations forward will shift the configuration rightwards, while computations backwards will shift it back leftwards, so the configurations of odd times will start in square 1, and those of even times in square 0 (see Figure 7).

Tape symbols will thus be the states of the c.a, and states will be the couples of states of the c.a, with in addition indication of a direction by a letter D or G, as follows :

$$D(l,q) \text{ or } G(q,r) \quad \text{where } l,q,r \in Q.$$

So main rules of the Tm will be, for each transition $\delta(l,q,r) = q'$ of the c.a when it goes rightwards

$$\Delta[D(l,q),r] = [D(q,r),q',+1]$$

when it goes leftwards

$$\Delta[G(q,r),g] = [G(l,q),q',-1]$$

these rules will be what we shall call the ordinary rules.

Inputs

Once started, a Tm receives nothing from the outer. So our Tm must have the entire input word at its disposal from the beginning and therefore we want our c.a to be a parallel c.a. But there is no easier task than turn a c.a to a parallel c.a (3.3.2), which has exactly the same halting time. So this is the first thing we do. In Figure 6 we show the s.t.d of the parallel c.a. Its set of states is $Q' = Q \cup Q \times X$, but the states of its initial configuration are q_0 and e (of Q).

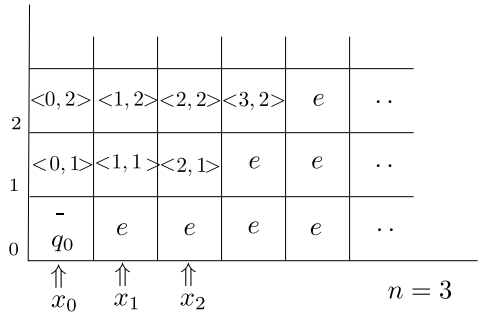


Fig.6

The Tm starts with the input word

$$(x_0, x_1, \ldots, x_{n-1})$$

on the tape (see Figure 7, where $n = 3$). It will have the initial state q_0 of the c.a in the starting state that we denote $E(q_0)$. It is clear that this configuration for the Tm is equivalent to the initial configuration of the parallel c.a (Figure 6). The rules for the Tm to compute the time 1-configuration of the (parallel) c.a will be analogous to the ordinary rightwards rules, except that the letters x_i must play the part not even of mere states , but of state e with input x_i, or state q_0 with input x_0.

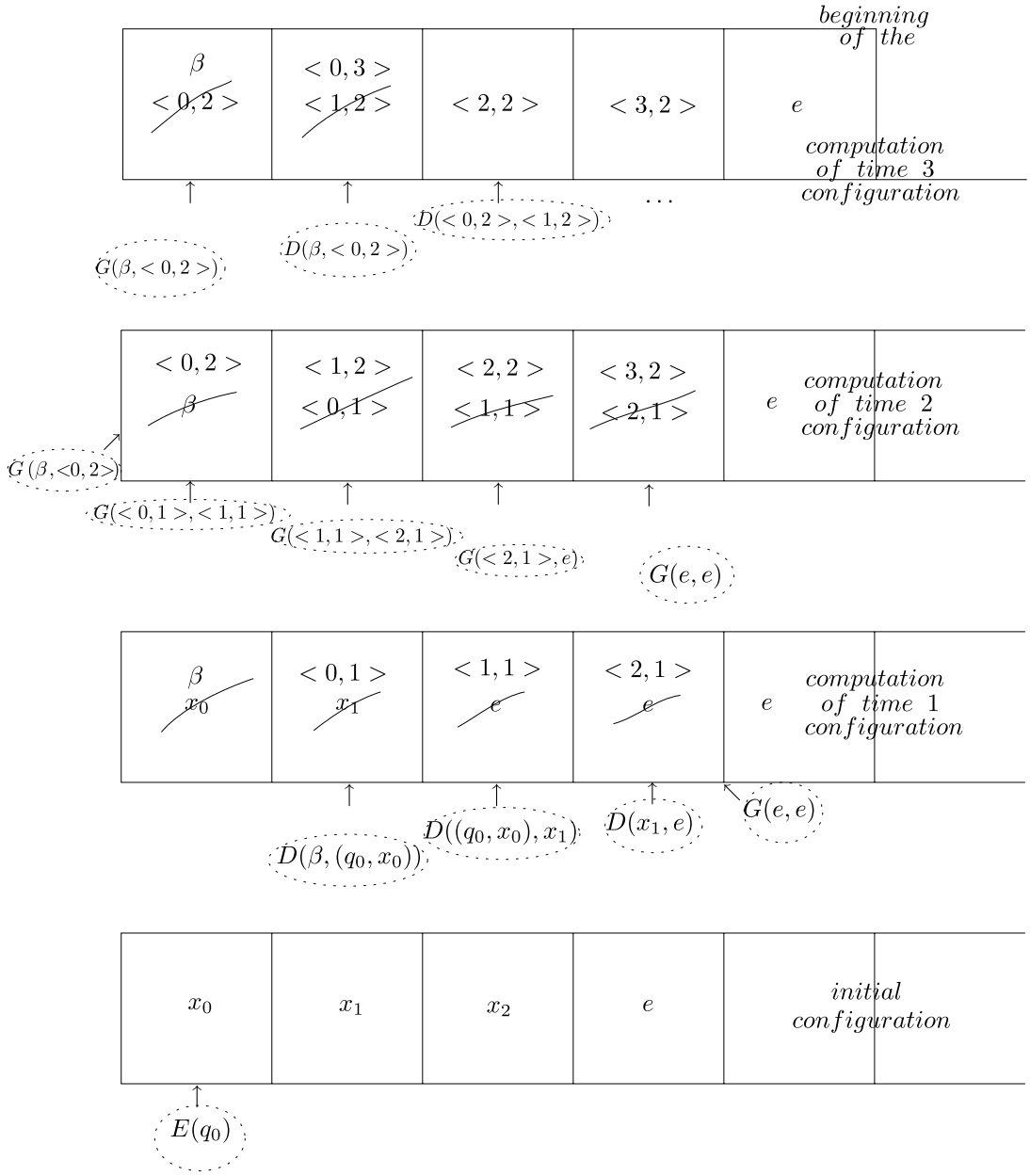

beginning of the computation of time 3 configuration
β
$< 0,2 >$
$< 0,3 >$
$< 1,2 >$
$< 2,2 >$
$< 3,2 >$
e
$\ldots$
$D(< 0,2 >, < 1,2 >)$
$D(\beta, < 0,2 >)$
$G(\beta, < 0,2 >)$
computation of time 2 configuration
$< 0,2 >$
β
$< 1,2 >$
$< 0,1 >$
$< 2,2 >$
$< 1,1 >$
$< 3,2 >$
$< 2,1 >$
e
$G(\beta, <0,2>)$
$G(< 0,1 >, < 1,1 >)$
$G(< 1,1 >, < 2,1 >)$
$G(< 2,1 >, e)$
$G(e, e)$
computation of time 1 configuration
β
x_0
$< 0,1 >$
x_1
$< 1,1 >$
e
$< 2,1 >$
e
e
$G(e, e)$
$D((q_0, x_0), x_1)$
$D(x_1, e)$
$D(\beta, (q_0, x_0))$
initial configuration
x_0
x_1
x_2
e
$E(q_0)$

Fig.7

Rules will be :

$$\Delta[E(q_0), x_0] = [D(\beta, (q_0, x_0))\ ,\ \beta, +1]$$

$$\Delta[D(\beta, (q_0, x_0), x_1] = [D((q_0, x_0), x_1)\ ,\ \delta^0(\beta, q_0, e, x_0), +1]$$

$$= [D((q_0, x_0), x_1)\ ,\ < 0, 1 >, +1]$$

$$\Delta[D((q_0, x_0), x_1), x_2] = [D(x_1, x_2)\ ,\ \delta^0(q_0, e, e, x_1), +1]$$

$$= [D(x_1, x_2)\ ,\ < 1, 1 >, +1]$$

$$\Delta[D(x_1, x_2), x_3] = [D(x_2, x_3)\ ,\ \delta^0(e, e, e, x_2), +1]$$

$$= [D(x_2, x_3)\ ,\ < 2, 1 >, +1]$$

In all these formulas, x_0, x_1, x_2, x_3 are any letters of the input alphabet X of the c.a's (original one and parallel one).

A rehearsal of tape symbols and states is not interesting because this construction will never be used.

To achieve describing the Tm, we lack only the rules to turn, that is the rules to pass to the computation of next c.a configuration. They are the following :

$$\Delta[D(l, e), e] = [G(e, e), \delta(l, e, e), 0]$$

$$\Delta[G(q, r), \beta] = [G(\beta, \delta(\beta, q, r)), \delta(\beta, q, r), 0]$$

$$\Delta[G(\beta, q'), r] = [D(\beta, q'), \beta, +1].$$

Mind that the ordinary rules are not valid in these cases, that is

- for the first one, when $q = r = e$
- for the second one, when $l = \beta$ or $q = \beta$

We can now punctiliously verify that the Tm computes the successive lines of the s.t.d, a line being ended at each 0-move of the head. Computation of one line takes time equal to its length, plus one for the odd time lines (Figure 7).

If we split the 0-moves in two moves we must add one time-step for each line.

Halting and time

The halting states of our Tm will be the G states where some halting state of the c.a has penetrated.

If halting occurs at an even time T, the Tm will stop when computation of line T ends (with head in state $G(\beta, q_a)$ on square 0), and if halting occurs at an odd time, the Tm will stop when computation of line $T+1$ ends,(with head in state $G(\beta, q_a)$ on square 0).

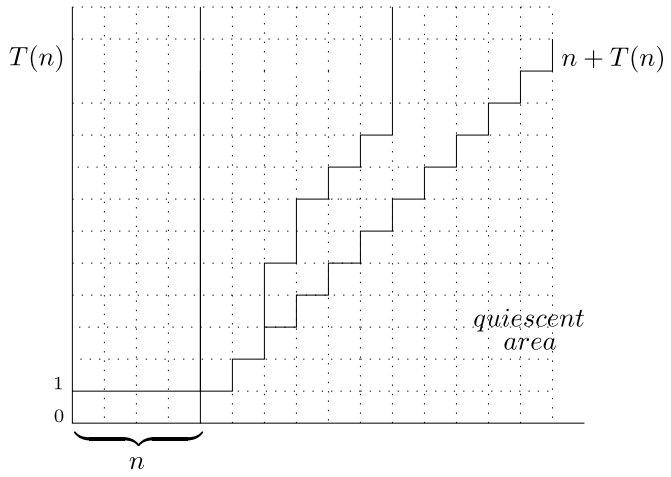


Fig.8

By examining Figure 8, and without being too fussy, we can easily bound the halting time (with 0-moves replaced by 2 moves) :

$$nT(n) + 2T(n) < T'(n) < nT(n) + 2T(n) + \frac{T(n)^2}{2}.$$

The upper bound may be reached. So we can state that recognizing time is at best of order $O(nT(n))$, and at worst of order $O(T^2(n))$.

For a computer c.a

with a little skill we shall build a Tm which computes : outputs will be computed with state of cell 0, that is

- when state G reads β it memorizes the output (if any) of odd time
- at next time-step, when G has β as first component, state D, while keeping preceding output, memorizes output of next (even) time.

State D will then deposit the one and the other successively in a corner of the first square containing no ouput yet. When the end marker has been deposited, it may be necessary to continue towards the unwritten part of the tape so as to erase all the states while coming back, leaving only the outputs on the tape.

Time is then increased by a number between $n + T(n)$ and $2(n + T(n))$, which does not change preceding orders.

Conclusion

is that any c.a may be simulated by a Tm, which halts on the same input words, recognizes the same words, or computes the same results. This Tm takes a time of order greater, but uses the same space. C.a's are machines equivalent to Turing machines and computers.

So c.a's (respectively c.a's which always halt) recognize recursively enumerable (recursive) sets and compute the partial recursive (recursive) functions.We particularly mention that languages recognized by c.a's in real time are recursive languages.

A c.a that mimics a Tm goes about as fast, but a Tm that tries to imitate a c.a is a real snail, taking a time multiplied by input length, or square of the time.

When we say that a c.a mimicking a Tm goes as fast, we could as well say it is as slow, and it is even slower if it is a real sequential c.a. Is this not surprising ? No, as a matter of fact the reason is precisely that it imitates the Tm, with moreover inadequate means if it is sequential. Working in its own way, it certainly will do much better, as we shall see with some examples.

4.3 A cumulator c.a

Here we shall see how some c.a can perform a task that we cannot imagine a Turing machine could achieve. We shall show

how a c.a can be made to cumulate its results

Let us indeed imagine that some c.a produces at the successive time-steps the representations in some basis (2, or 10, or any other d) of integer numbers, with units in cell 0, and the digits for d^i in cell i (as the states or as components of the states in some place P). In Figure 9 we place the cells in reverse order so as to recognize easily the usual elementary way of adding).

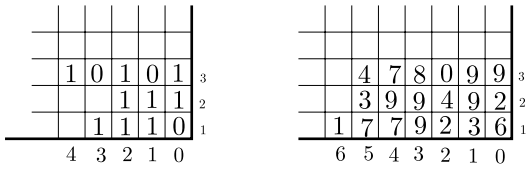


Fig.9

To the states we add two components, in two places denoted S and C (for sum and carry) : there we shall find represented, splitted in two parts, the sum and the carry, the cumulated sum of all the numbers produced and represented at the preceding time-steps. The rules are excessively simple (see Figure 10) :

- place C of cell 0 is of course empty at any time

- in place P, at time t ($1 \leq t \leq N$), over several cells, appear the digits of a new number

- in place S of cell c we have at time $t+1$ the sum mod d of places P, S and C of cell c at time t

- in place C of cell $c+1$ we have at time $t+1$ the sum divided by d of places P, S and C of cell c

Observe that, whatever the basis d, the carry is always 0 or 1, at any time and on any cell. Indeed

- place C is empty in cell 0 at any time t and in all cells at time 0 (and 1)

- by induction, for cell c at time t :

$$P + S + C \leq (d-1) + (d-1) + 1 = 2d - 1$$

whence $(P + S + C) div\ d \leq 1$, so carry in cell $c+1$ at time $t+1$ is at most 1

Figure 10 shows two examples, in basis 2 and 10 respectively, where only 3 numbers are produced and added.

Time for complete computation of the sum

If the last number is produced at time N, we have the sum splitted in two at time $N+1$. But we want the sum completely computed, so when will this be achieved ?

- at time $N+1$ all places P are empty, with no carry in cell 0

- so at time $N+2$ carry in cell 1 is 0, that is to say that the 2 first carries are now 0, and by induction, at time $N+i$ the i first carries are 0. Thus, if the total sum of the numbers produced is strictly less than d^ν, at time $N+\nu$ all the carries being 0, this total sum is exactly the number represented in the S places

So we are assured to have the result computed in the c.a in time at most $N+\nu$, where N is the number of integers produced, and d^ν is an upper bound for their sum.

Time for outputting the sum

Using the sum computed at time $N+n$ will probably need some synchronization at time $N+n$ or later.

Outputting it may also be started by a synchronization at time $N+n$, and so achieved at time $N+2n$. But a much simpler mean is possible : a marker, entered in cell 0 just after the last number is produced, can travel from cell to cell at speed 1 to send one digit of the result after the other rightwards at speed 1, they will be output from cell 0 at times $N+1$, $N+3$, ... , $N+2n-1$, so before time $N+2n$.

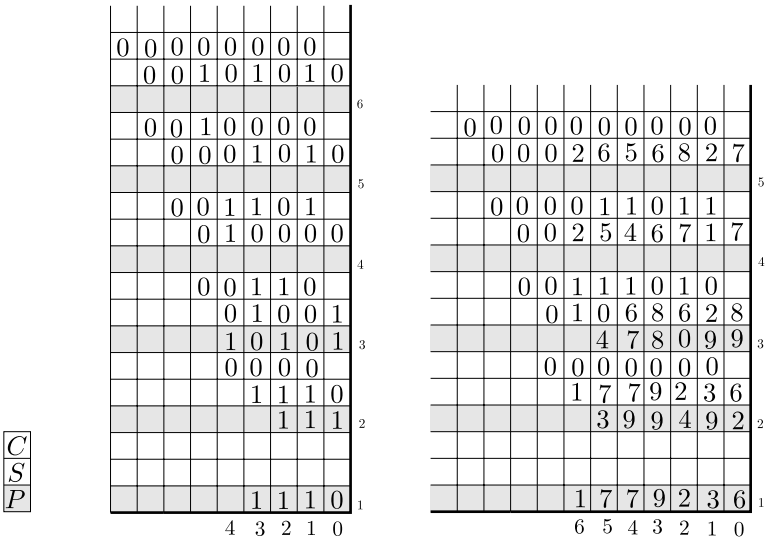


Fig.10

4.4 A multiplier c.a

We should like now to have some example of a c.a producing numbers to be added by such a cumulator process as we have just described.

The multiplication algorithm will give us this opportunity. We shall use binary representations to begin with, this makes work simpler, but we shall generalize afterwards.

Let then

$$a = \sum_{i=0}^{m} a_i 2^i \qquad b = \sum_{j=0}^{n} b_j 2^j$$

be two integers, the product of which we want to compute. We have

$$2^{m+n} \leq p = a.b < 2^{m+n+2}$$

so the last digit of p is p_{m+n} or p_{m+n+1}.

4.4.1 The usual multiplication algorithm

Let us examine it carefully with an example (Figure 11). On the successive lines we find : ab_0, ab_1, ... on line i ab_i which is a if $b_i = 1$ and 0 if $b_i = 0$ (let us underline that this would not be exactly the case in a basis other than 2, because computation of each line may then use carries). More precisely, in square (i, j) of the product area we have the product $a_{j-i}b_i$.

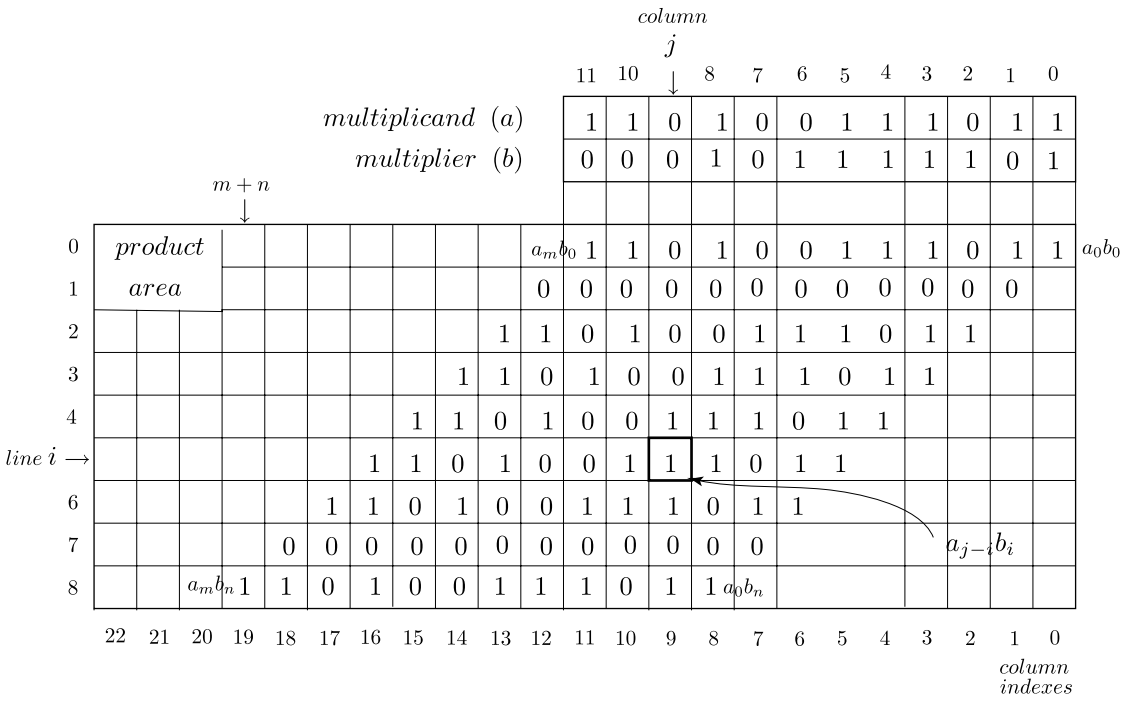


Fig.11

4.4.2 A c.a producing numbers

We shall sketch the working of a c.a producing at each time-step, not exactly the numbers represented at lines 0 to n, but numbers whose sum will be the same, obtained by getting column 1 one step down, column 2 two steps down, and so on, column $m+n$, $m+n$ steps down. Indeed, if the terms in the columns

are the same, the sum is the same, and the numbers appearing at each time-step do not matter much.

The states will have 5 components, in places L_1, L_2, M_1, M_2 and H_1 (see Figure 12).

Input word will be

$$a_m, \dots, a_0, *, b_0, \dots, b_n.$$

Rules are such that

- inputs before $*$ enter in place L_1, they push contents of L_1 (if any) in place L_2

- contents of L_2 enter at next time-step place L_1 of next cell, and push any contents of this place in L_2. This results in inputs a_i progressively setting in place L_1 of cells c_i

- when marker $*$ enters, it definitely closes entering in place L_1. Inputs now enter in M_2, $*$ to begin with

- contents of place M_2 travel right at speed 1

- with $*$ in place M_2, contents of L_1 go in M_2

- contents of M_2 go in H, and contents of H go in place M_2 of next cell. This results in inputs a_i traveling right at speed 1/2

The terms to be added are the products of inputs in M_1 and M_2. In basis 2, these products are 0's or 1's. In a basis $d \geq 3$, these products are not necessarily less than d, as were the digits of the numbers cumulated by ouir c.a of preceding section. But this will make no problem really.

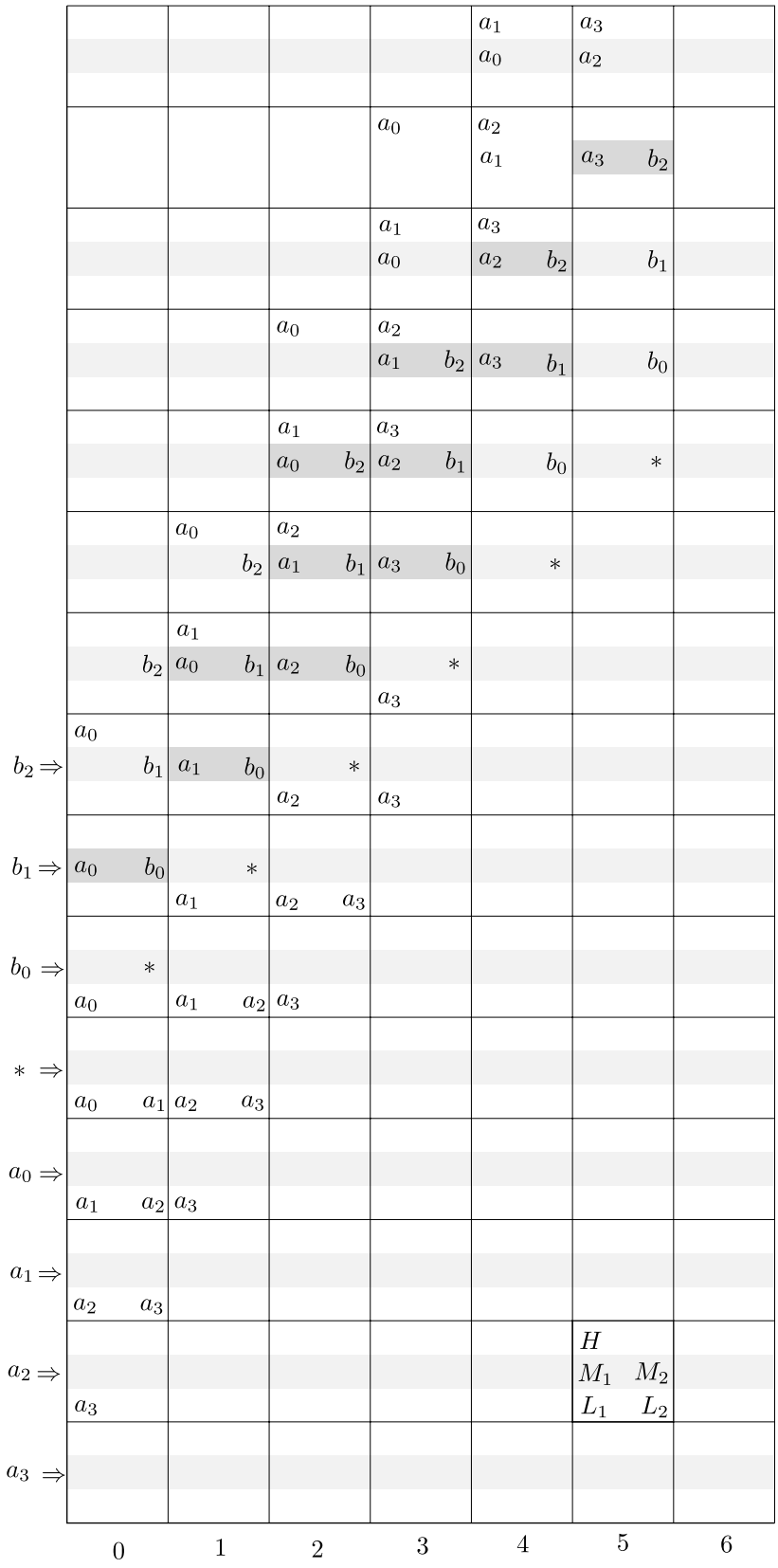

a_1 a_3
a_0 a_2
a_0 a_2
a_1 a_3 b_2
a_1 a_3
a_0 a_2 b_2 b_1
a_0 a_2
a_1 b_2 a_3 b_1 b_0
a_1 a_3
a_0 b_2 a_2 b_1 b_0 $*$
a_0 a_2
b_2 a_1 b_1 a_3 b_0 $*$
a_1
b_2 a_0 b_1 a_2 b_0 $*$
a_3
a_0
$b_2 \Rightarrow$ b_1 a_1 b_0 $*$
a_2 a_3
$b_1 \Rightarrow$ a_0 b_0 $*$
a_1 a_2 a_3
$b_0 \Rightarrow$ $*$
a_0 a_1 a_2 a_3
$* \Rightarrow$
a_0 a_1 a_2 a_3
$a_0 \Rightarrow$
a_1 a_2 a_3
$a_1 \Rightarrow$
a_2 a_3
H
$a_2 \Rightarrow$ M_1 M_2
a_3 L_1 L_2
$a_3 \Rightarrow$
0 1 2 3 4 5 6

Fig.12

4.4.3 A multiplier c.a

If we now tinker preceding c.a by adding places S and C, and make them work on the products of places M_1 and M_2, it will compute the product of our two numbers.

In what time ? The last product to be produced is $a_m \times b_n$. It is produced on the site which is on cell $m+n$, and on the trajectory of input b_n ($< c, m+n+3+c >$), so at time $2m+2n+3$.

As the product of our two numbers is strictly less than 2^{m+n+2}, applying the general result concerning the cumulator, we are sure to have the result computed at time at most $(2m+2n+3)+(m+n+2) = 3m+3n+5$.

But here we can be more precise. Indeed our terms $a_i b_j$ (see Figure 12) appear in sites situated in a lozenge delimited by sites

- $< 0, (m+3)+0 >$ where $a_0 b_0$ first appears
- $< m, (m+3)+m >$ where $a_m b_0$ appears
- $< n, (m+3)+2n >$ where $a_0 b_n$ appears
- $< n+m, (m+3)+2n+m >$ where $a_m b_0$ appears

In each site of this lozenge we shall have at most 3 terms to add, the contents of S and C, and some $a_i b_j$. In all the sites which are above we have no carry and no $a_i b_j$, so the value of S is definite. We could have a last carry in site $< n+m+1, (m+3)+2n+m+1 >$, so it is only at time $(m+3)+2n+m+1+1 = 2m+2n+5$ that we are sure to have the last digit of the product computed.

If we want to output the product, it is quite easy here : we just decide that the contents of place S travel leftwards at speed 1 if there is no a_i and b_j appearing both. Thus the last digit of the product arrives in cell 0 at time

$$2m+2n+5 \quad + \quad m+n+1 = 3m+3n+6$$

which is a time linear in the length of the input, about 3 times this length.

4.4.4 In bases other than 2

The usual multiplication algorithm in basis 2 has given us the idea and the features for a multiplier c.a. Now we forget it, because of the carries in the computation of lines. We just examine how the preceding multiplier c.a works if we feed it with an input word corresponding to representations in some basis d.

In each site of the lozenge where the $a_i b_j$ appear, we have to add $a_i b_j$ which is at most $(d-1)^2$, the content of place S which is at most $d-1$ and the carry in place C. Let us show by induction that the carry is always at most $d-1$. Indeed, in all sites on the left side of the lozenge, there is no carry. And if the carry is less than $d-1$ in a site $< c, t >$, then the carry in site $< c+1, t+1$ will be the quotient by d of a sum

$$\leq (d-1)^2 + (d-1) + (d-1) = d^2 - 1$$

so it is indeed at most $d-1$.

The c.a works exactly as in basis 2. Only there are more states because more values are possible in all places, but the result is obtained more quickly because the lengthes of the representations, m and n are less.

4.5 Atrubin's c.a for multiplication

A much cuter c.a !

As before, the two integers to be multiplied are given at first in binary notation, which is lighter to handle :

$$a = \sum_{i=0}^{m} a_i 2^i \qquad b = \sum_{j=0}^{n} b_j 2^j.$$

In Atrubin's c.a [1], a and b are entered together, lower digits first, so inputs are

$$x_0 = (a_0, b_0),\ x_1 = (a_1, b_1), \ldots\ldots .$$

After input of x_m there is no more of a, after input of x_n there is no more of b, so after time $max(m,n)$ there is simply no more input. Outputs will be, from time 1 up, the digits of p :

$$p_0,\ \ p_1, \ldots\ldots\ldots p_{m+n},\ \ p_{m+n+1}.$$

Last digit is output at time $m+n+1$ or $m+n+2$.

We could think of ending each input word by an end marker. Such end markers though are not necessary if input 0 should not be confused with no input. Anyway, we delay discussing how the computation ends.

4.5.1 The usual multiplication algorithm revisited

As it will be our guide for conceiving the c.a, we shall examine it once more very carefully (Figure 13). On the successive lines of the product area we find : ab_0, ab_1, ... on line i ab_i which is a if $b_i = 1$ and 0 if $b_i = 0$ (let us underline that in base 2, computation of the lines uses no carry, which will not be the case in base 3 or more). More precisely, in square (i,j) of this area we have the product $a_{j-i}b_i$.

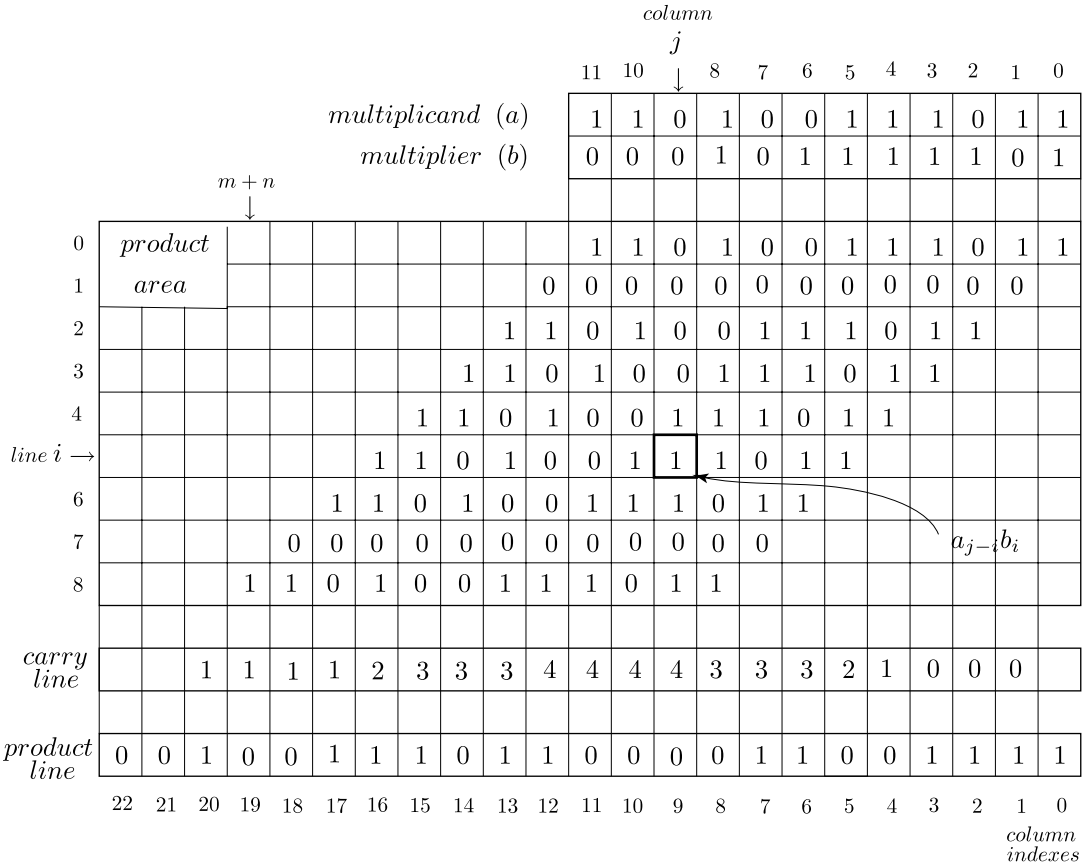


Fig.13

Usually, digits of the product are obtained by adding the elements of these lines by columns (column j corresponding to implicit factor 2^j), starting with column 0. Results of these additions are generally greater than 2, so that these results split into digit p_j (0 or 1) of the product and a carry r_{j+1} for next column. This carry we note in the carry line, in any notation, decimal in Figure 13. We have

$$p_j = (\sum_{i=0}^{j} a_{j-i}b_i \ + \ r_j) mod 2$$

$$r_{j+1} = (\sum_{i=0}^{j} a_{j-i}b_i \ + \ r_j) div 2 \quad \text{(integer quotient).}$$

Notice that r_0 is absent and $r_1 = 0$. Notice also that the last non empty column is that of rank $m + n$.

Let us now observe that we can add terms of the product area in any order we wish, grouping them as we wish, to add them little by little to what constitutes a partial product, without omitting to compute the carry as well. When we add terms

$$a_{j-i_1}b_{i_1}, \ldots\ldots a_{j-i_k}b_{i_k} \text{ and } r_j$$

of column j to the partial product present in the product line, changes will be precisely :

$$\text{if } \; s = a_{j-i_1}b_{i_1} + \ldots\ldots + a_{j-i_k}b_{i_k} + r_j$$

$$r'_j = 0$$

$$p'_j = (p_j + s) mod 2$$

$$r'_{j+1} = r_{j+1} + (p_j + s) div 2.$$

Product is achieved when product area and carry line are emptied. But before this finally happens, each digit p_j of the product is the final one as soon as product area and carry line are emptied from column 0 to column j included.

We shall see that Atrubin's c.a proceeds in this progressive manner. At time 17 of our example (Figure 16), in cell 0 it achieves computing p_{16}, while in cells 1, 2, ... it increases p_{17}, p_{18} and p_{19}.

But in what order does it add terms ? Term p_k (and carry r_{k+1}) result from summing

$$a_k b_0 + a_{k-1}b_1 + a_{k-2}b_2 + \ldots\ldots + a_2 b_{k-2} + a_1 b_{k-1} + a_0 b_k.$$

The c.a will start with the first 1 to 4 (depending on k mod 4) central terms of this sum (indexes of a and b equal or close), it will add packs of 4 terms (the 2 next ones on the left and the 2 next ones on the right) and end with the 2 end terms $a_k b_0 + a_0 b_k$ (indexes of a and b distant).

4.5.2 The multiplier c.a

The successive inputs will be, if we suppose $n \leq m$

$$x_0 = (a_0, b_0) \quad x_1 = (a_1, b_1) \quad \ldots\ldots \quad x_n = (a_n, b_n)$$

b_n (and some preceding b_i's) being absent if n is strictly inferior to m.

Each state will have two parts : one part for conveying and stocking the inputs (beneath in Figures), and one part for progressive computing of the product (above, see Figure 14).

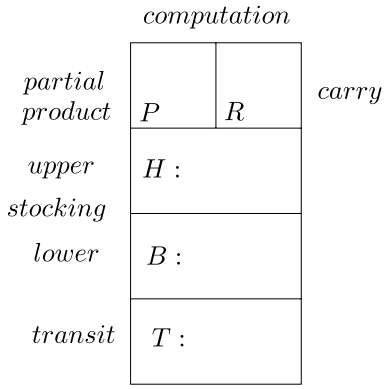


Fig.14

Each input x_i enters transit square T of cell 0 at time $i+1$, then travels rightwards at maximal speed until it reaches a cell where parking place B, or otherwise H, is free. There it settles (see Figure 16).

When an input x_j traveling rightwards crosses an input x_i $(i < j)$ already settled (in cell $\lfloor i/2 \rfloor$), products $a_j b_i$ and $a_i b_j$ can be computed. Sum of these two terms is the contribution $c(i,j)$ supplied to the final product by the two inputs x_i and x_j, if $i \neq j$. As for products $a_i b_i$, they are computed when x_i reaches its parking place ($\lfloor i/2 \rfloor$).

As a cell may compute with contents of its two left and right neighbours, we have some choice for deciding of the site where contribution i, j will be computed. We shall follow Atrubin's choice, which proves to be the wiser.

Example of Figure 16 will help us understand the rules : we have encircled sites where contributions of column 11 will be picked up, and coloured sites where partial product p_{11} and carry r_{12} are computed. Calculation of product p_{11} starts on site $< 2, 10 >$, at each time step it increases with two contributions and moves back one cell, it finally ends on site $< 0, 12 >$.

In Figure 15 we show the corresponding rules, which incorporate to a partial product expected contributions and carry from preceding column, so as to determine next partial product $P < c, t+1 >$. Rule for cell 0 is a bit different : indeed we also incorporate there contribution $c(0,t)$, which is possible because x_0 is here and x_t is input at preceding time.

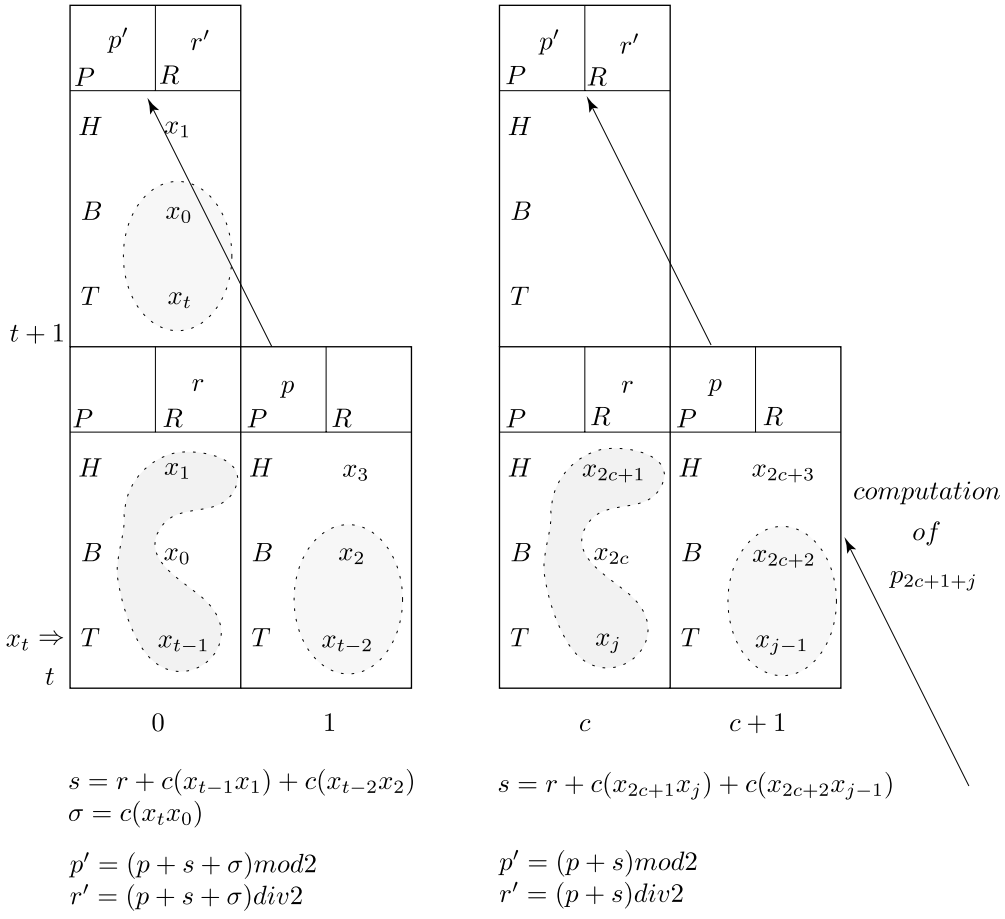


Fig.15

11 being an odd index, in the first example of Figure 16 we have no central term $a_i b_i$. For the case of even indexes (16, second example of Figure 16) preceding rules must be completed as follows : if one of the places giving a contribution contains x_j in T and its (still empty for the while) parking place , corresponding contribution must be $a_j b_j$. Atrubin uses a little notation trick to make the task of reading or writing the s.t.d easier for the reader, and we shall keep the idea but change the trick : when x_j arrives before its parking place, we put there (on the s.t.d) $x_j/2$, so that when we apply rules of Figure 15, we obtain contribution

$$a_j.b_j/2 + a_j/2.b_j = a_j b_j$$

which is exactly the desired one.

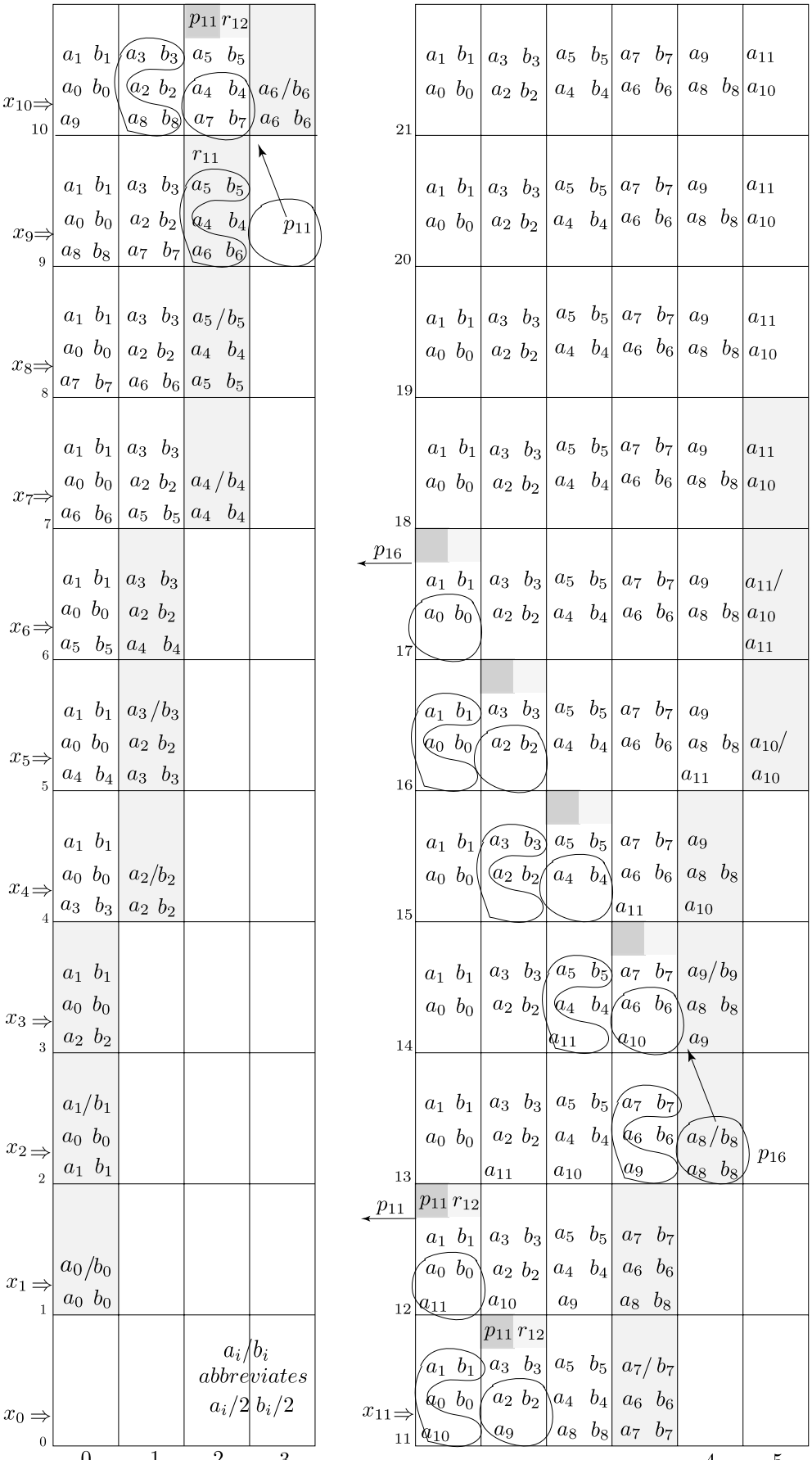


Fig.16 $m = 11$ $n = 8$

On Figure 16 we have lightly coloured sites where summing of the successive columns start. They are situated on a line of slope 3.

We also notice that places R and P on the anti-diagonal of these sites contain the successive partial summings of corresponding column (carry will rather be written in binary), the last one on site $< 0, k+1 >$.

We see, by examining Figure 13 of the usual multiplication, not the s.t.d, that column $m+n$ is the last one to receive a contribution $(a_m b_n)$, column $m+n+1$ can only receive a carry : antidiagonals of the c.a receiving the same quantities, next anti-diagonal $m+n+2$ will receive nothing more, so there can be no more output from time $m+n+2$ or $m+n+3$=$length(a)+length(b)+1$. (No output is not the same thing as output 0). At that we know that the product is achieved. So the time for computing the product is, to within one unit,

$$length(a) + length(b) \approx log_2 a + log_2 b.$$

4.5.3 Number of states

Each two places T, H or B and place P of the states may contain : nothing, or 0 or 1.

Let us examine the possible values of the carry : each carry is the integer quotient by 2 of a sum formed of

- a preceding carry
- 4 terms $a_i b_j$ having value 0 ou 1
- a partial product of value 0 or 1

so we have

$$r' \leq \frac{r+5}{2}.$$

The first carry being 0, we see by induction that the carry cannot exceed 4. Place R of the states can so contain : nothing or one of the integers 0, 1, 2, 3, 4. So the number of states is at most

$$3^7 \times 6 = 13122.$$

Atrubin counts only 1250 states, but he does not distinguish input 0 and no input, digit 0 or no digit, in short he does not worry about the c.a notifying that the product is ended. Surely, we know its length ! But we prefer that an automaton should do its task to the last.

In a basis other than 2

We can choose usual basis 10 to represent any basis. C.a will function exactly in the same way, with the only difference that the carry is much bigger. Indeed

it results from summing 4 terms $a_i b_j$, the preceding carry, and the preceding partial product. If basis is 10 we can bound it as follows

$$r' \leq \frac{r + 4 \times 81 + 9}{10} = \frac{r}{10} + 33,3$$

which gives by induction $r \leq 36$. The number of states is much bigger, but the time for computing the product is about

$$\log_{10}(a) + \log_{10}(b),$$

much smaller naturally. Figure 13 gives an example.

4.5.4 Comparison with Turing machines

A Tm reproducing the usual multiplication algorithm takes a time of order (see [12])

$$mn = \log a \times \log b = \text{length}(a) \times \text{length}(b)$$

with a probably very considerable coefficient. This time can be reduced by linear speeding up, but this needs grouping, makes things still more complicated, and multiplies states and symbols.

Atrubin's c.a, so naturally and so simply does the same computation in time

$$\text{length}(a) + \text{length}(b) \ \ !$$

(which is not the length of its input, the latter being $\max(\text{length}(a), \text{length}(b))$)

Should we mention that the usual multiplication algorithm takes a time of order mn too, about $2mn$?

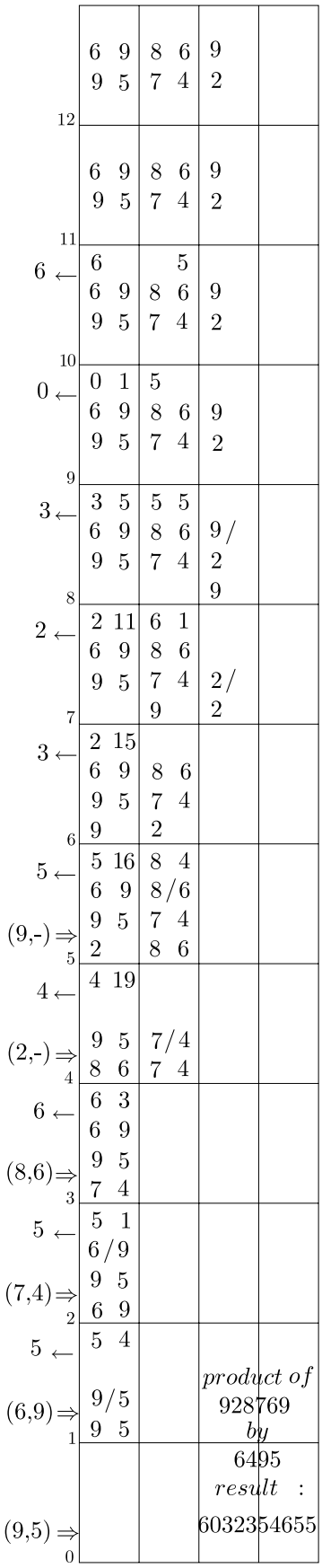

12
6 9
9 5
8 6
7 4
9
2
11
6 9
9 5
8 6
7 4
9
2
6 ←
10
6
6 9
9 5
5
8 6
7 4
9
2
0 ←
9
0 1
6 9
9 5
5
8 6
7 4
9
2
3 ←
8
3 5
6 9
9 5
5 5
8 6
7 4
9 /
2
9
2 ←
7
2 11
6 9
9 5
6 1
8 6
7 4
9
2 /
2
3 ←
6
2 15
6 9
9 5
9
8 6
7 4
2
5 ←
(9,-) ⇒
5
5 16
6 9
9 5
2
8 4
8 /6
7 4
8 6
4 ←
(2,-) ⇒
4
4 19
9 5
8 6
7 /4
7 4
6 ←
(8,6) ⇒
3
6 3
6 9
9 5
7 4
5 ←
(7,4) ⇒
2
5 1
6 /9
9 5
6 9
5 ←
(6,9) ⇒
1
5 4
9 /5
9 5
product of
928769
by
6495
(9,5) ⇒
0
result :
6032354655

Fig.17

Chapter 5

Signals and waves

5.1 Definitions

5.1.1 The notion of a signal

In the preceeding chapters we have already made a large use of signals, without which we could hardly have managed. It is time we study them more systematically.

The general and intuitive idea of a signal is that of some information propagating through space. We shall try to make precise what this can be in the c.a landscape, with the hope of arriving to some definition.

Dealing with c.a's, a piece of information at some determined instant can only be some state on some cell, or else some k-tuple of states (some word-of-states to avoid precising its length) on several consecutive cells. At different times this information may appear under different shapes and so the signal will be caracterized by a set of states (necessarily finite) or a set of words-of-states, which must be finite, otherwise nearly anything could be a signal ! So first of all,

a signal corresponds to a set of states or words-of-states which is finite.

But as we shall see little by little, not any such set !

If the words-of-states of a signal all have length 1, so find themselves on one single cell, we shall say the signal has *thickness 1* or is *threadlike*. In this case we easily distinguish the states of the signal, forming subset S of the set of states, from the other states, subset $Q \setminus S$, which form what we shall call the background. In the general case it is mandatory that the words-of-states of the signal should be clearly distinguished from the background. This does not imply that states of these words do not appear in the background, so this requirement we have just made seems a bit vague . . . .

We now come to propagation : the mean for it is transition, and so we have the s.t.d in mind.

First thing, propagation evokes more or less continuity, we wouldn't like to call signal some set of states appearing from time to time, intermittently disappearing and arising anew. So

a signal must be present at each instant of time.

In the case of a threadlike signal, if the signal is in cell i at time t, at time $t+1$ it will be in one of the three cells i, $i-1$ or $i+1$, the only ones that have perceived it, progress of the signal is consequently limited. On the space-time diagram it will be possible to follow the trajectory of the signal, which is the succession of the sites it occupies. From another point of view, the trajectory may be seen as the succession of the moves of the information, no move, left or right move. In this case, to a representation where cells take all the place, we shall prefer a representation where they have shrinked to points, leaving all the place for an intercellular space where we can draw the moves (Figure 1).

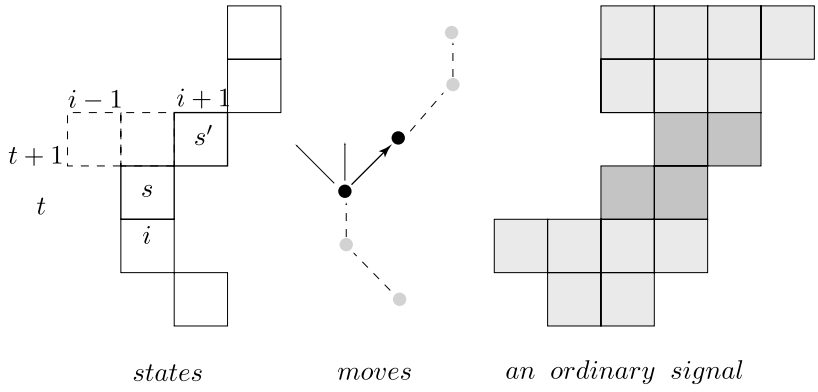


Fig.1

For a thicker signal, whose thickness may vary, the word-of-states at time $t+1$ results from the word of states at time t : consequently, its right border cannot progress of more than one cell on the right (but it seems it could regress more) and likewise its left border cannot progress of more than one cell on the left. The trajectory will appear as a ribbon of variable width (Figure 1). The first example we have met of such signals was in Mazoyer's c.a (chapter 2), signals AB, BC, CA of slope -1.

Now then, transition function, by which the signal progresses, takes into account the four states (two at the left, two at the right) surrounding the signal. But in the idea we have, unless exceptionally, a signal crosses the place without being pushed to and fro by surrounding states that happen to be there ; this implies that the result of transitions involving states at the left or at the right of the signal be independant of these states. Our new requirement is then

a signal is indifferent to its environment.

For an example, if signal is threadlike and goes rightwards from state s to state s' (as on Figure 1), we must have

$$\delta(s, ?, ?) = s'$$

for ? any state not belonging to the signal, while

$$\delta(?, ?, s) \quad \text{and} \quad \delta(?, s, ?)$$

should not belong to the signal.

These three requirements are the basical ones.

A very simple example of s.t.d areas that are not signals is given by the areas between two signals, which may have same or different slopes : these areas are shaped by the signals which delimit them (Figure 2), they probably do not satisfy the first requirement and clearly not the last one.

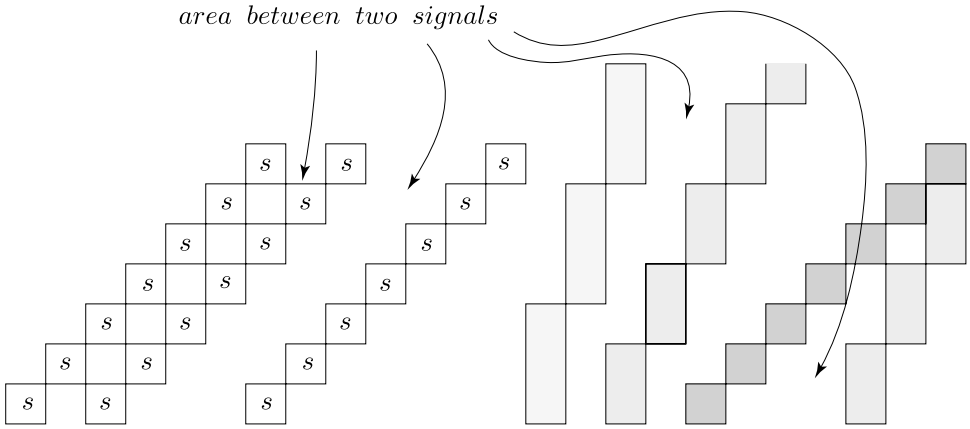


Fig.2

A more interesting example can be found in Mazoyer's c.a (chapter 2, Figure 9) : S_1 is the area where state S developes after stopping (which shows that state S is not totally inert) a first wave of signals AB, BC, CA which gives it its shape. This phenomenon appears more clearly still when we look at the next S_k's, formed with the same state S, and nevertheless following a different path.

So except accidentally (after bumping on other elements), a signal progresses without being influenced by its environment, by its own strength.

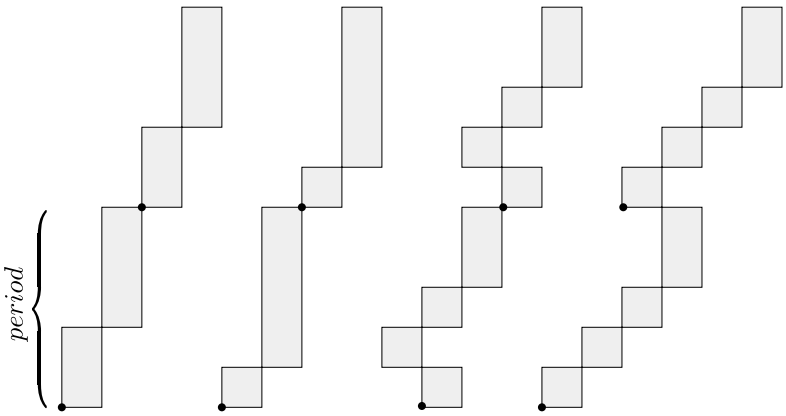
This is the reason why we study its normal progress on a quiescent background.

Out of the several requirements we have made no neat definition emerges, but nevertheless a precise description.

5.1.2 Description of a signal

let us first examine a threadlike signal : the number of states being finite, if the signal doesn't extinguish, it must be periodic. We then define its speed, which is the ratio of the sum of the moves (+1 rightwards, -1 leftwards or 0) during one period divided by the period, and its slope, which is the inverse number. These two numbers are rational numbers. The maximum speed possible is 1.

The trajectory of a signal, which does not mention the states, does not characterize it entirely, though it is an essential feature for the tool a signal represents, the slope which is an average number still less (on Figure 3 we see several signals having same slope). The trajectory can be seen as a discrete approximation of a straightline of rational slope.



several trajectories of signals of speed 2/5

Fig.3

A thicker signal also will be periodic, as his set of words-of-states is finite. It is possible to define in the same way its speed, ratio of the number of cells and the number of time units between two sites at a period-distance (Figure 4). As its right (/left) border progresses rightwards (/leftwards) of at most one cell per time unit, this speed has (absolute) value at most 1.

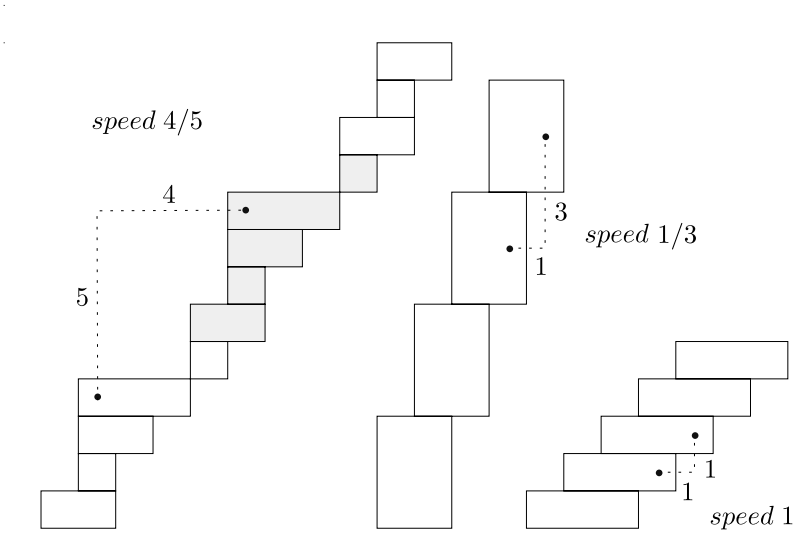


Fig.4

The description we have just made will permit us to spot signals in s.t.d's. They are very simple objects, that we must not confuse with more intricate ones that we shall examine further on.

5.1.3 Construction of signals

Can we implement in a c.a a signal approximating any straightline with positive slope (negative slope will result by a symmetry) that we choose ?

It is clear by now that the answer is **no** if the straightline has speed greater than 1 or not a rational number.

But if the speed is no greater than 1 and a rational number

$$\frac{a}{h} \qquad a \leq h \qquad (a,h) = 1$$

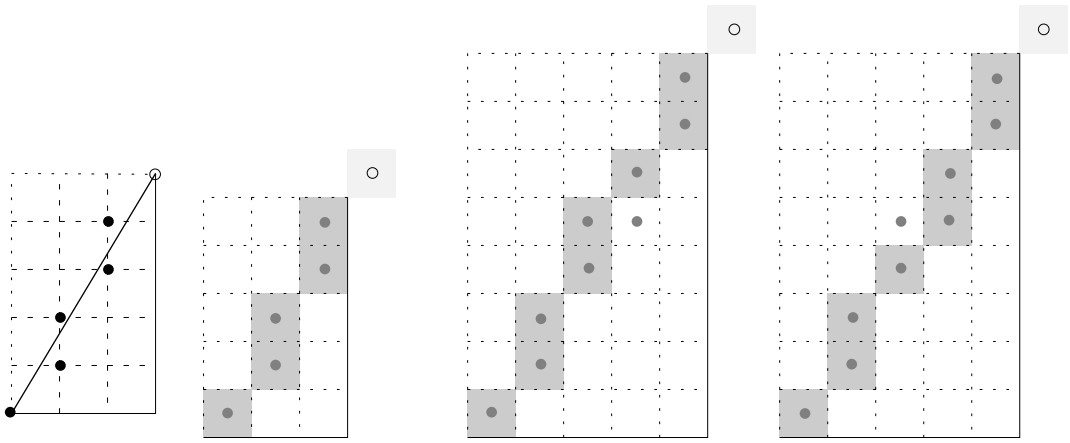
we shall find numerous possibilities. The simplest ones will involve

a right moves (and $h - a$ standstills) during h units of time.

The trajectory may vary according to the order in which right moves alternate with standstills. Let us mention that 2 standstills could be replaced by one move right and one move left, but as a rule simple objects are preferred to artificially complicated ones. In any case, whatever the trajectory, h states (or words-of-state) will be needed.

Among the simplest approximations, there seems though to be a best one, the one nearest the geometrical straightline : in the representation where cells are reduced to points, we draw the straightline from site $(0,0)$ to site (a,h),

which passes at time t at the distance ta/h from cell 0, and we choose for our signal to pass at this time on cell $[a/h]$, that is $\lfloor a/h \rfloor$ or $\lceil a/h \rceil$. In case $h = 2k$, at time $t = k$ we may hesitate between the 2 cells $\lfloor a/2 \rfloor$ or $\lceil a/2 \rceil$, but the two choices give the same global trajectory. See Figure 20.



$a = 3 \quad h = 5$ $\qquad\qquad$ $a = 5 \quad h = 8$

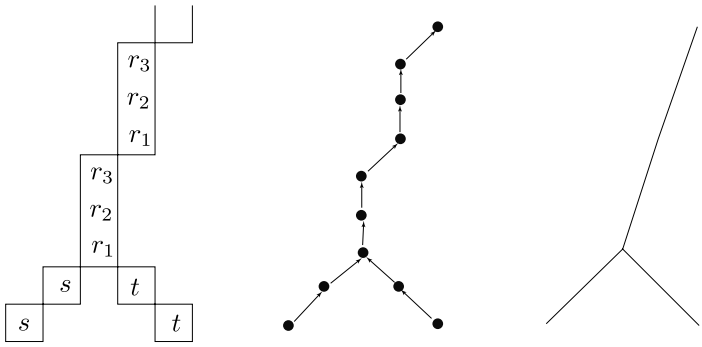
Fig.5

5.1.4 Geometry of signals

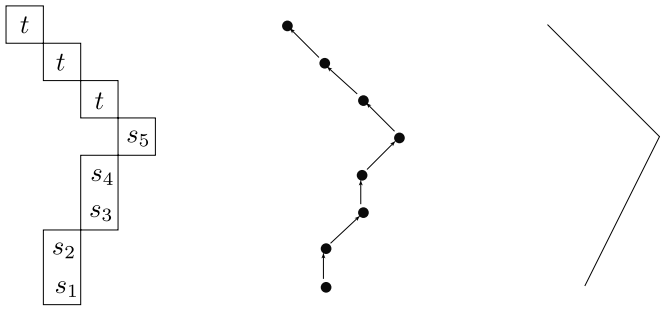
The fact that the trajectories of threadlike signals, the most frequently used ones, are approximations of straightlines in the s.t.d's gives us the idea that some c.a problems may be solved by a geometrical approach, and there precisely lies the very interest of this notion of signals.

Now then geometry problems solve with straightlines that meet.

Likewise, if our signals progress straightforward, this must be with the aim of arriving somewhere, on a border, a particular area, to a meeting point with some other signal. Exceptional events will then take place : reflections, extinction of signals, springing of new signals, delays ... (Figure 5). We have made such experiences since chapter 1.



The meeting of s of speed 1 and t of speed -1 generates r of speed 1/3



A signal s of slope 2 and duration 5 emits as it extinguishes t of slope -1

Fig.6

So signals, indifferent though they will be to most neighbourhoods, will react to a certain number of particular neighbourhoods, expressing various events.

Now then, unfortunately, trajectories of signals are only discrete approximations of straightlines, submitted moreover to specific constraints : maximal speed 1, rational slopes.

Will it always be possible to come back from a geometric solution to some c.a s.t.d, as we already did a few times ? Probably if the geometrical figure contains only straightlines or segments of rational slopes, and meeting points having integer coordinates. We shall then obtain a s.t.d by replacing the straightlines or segments by signals of same slope. If coordinates of the meeting points are not integers, but are rational, with a finite number of denominators, a magnifying, that is a division of cells can be thought of.

5.1.5 Networks of signals

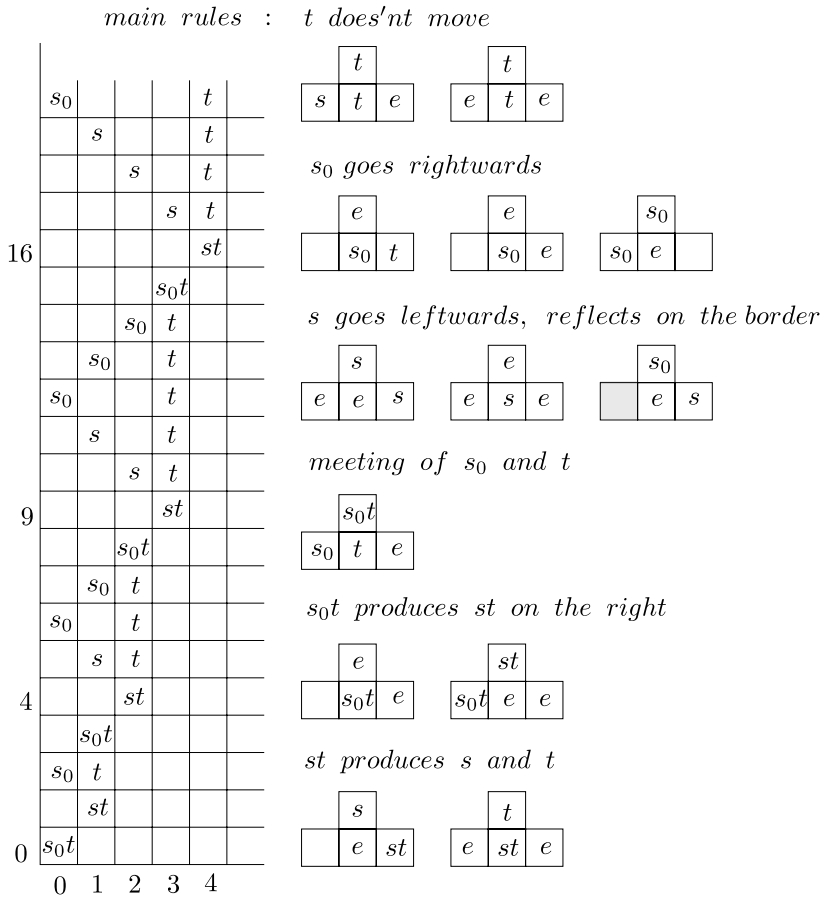


Fig.7

Let us examine Figure 6 : couldn't signal $\{s_0t, st, t\}$ be called a parabolic signal ? indeed it goes through all the $< c, t = c^2 >$ sites. But this would be forgetting the second signal $\{s_0t, st, s, s_0\}$ without which it can not exist : in fact the two signals cannot be separated from one another, they even have states in common, they form what we shall call a network of signals.

Perhaps we could describe this example a bit differently, admitting that two signals, s (or s_0) and t can find themselves together on a cell. We would then say : s and t present together on a cell move together one step right, after what, t staying on the cell s goes left at maximal speed, is reflected by the border, joins t and everything repeats .... This does not alter the fact that the two signals cannot be separated.

We shall study a very beautiful example of a network a bit later.

5.1.6 A curious example

Let us consider c.a having set of states $\{e, 0, 1_0, 1, 2, 3\}$, initial impulse state 0, and the following transition rules :

0	|	e	1_0	|	e	1	|	e	2	|	e	3	|	e
−	+	−	−	+	−	−	+	−	−	+	−	−	+	−
e	|	e	e	|	e	e	|	e	e	|	e	e	|	e
β	|	e							β	|	e			

e	|	e	0	1_0	1	2	3
−	+	−	−	−	−	−	−
β	|	e		2		e	
e	|	e		2		e	1
0	|	1_0					
1_0	|	0					
1	|	e					
2	|	2	2	3	3		
3	|		e				

Its s.t.d is to be found in Figure 7, with two different representations, the first one the usual one where sites and states are to be seen, the secund one where cells have shrinked and moves appear, which will be more convenient for geometric reasoning.

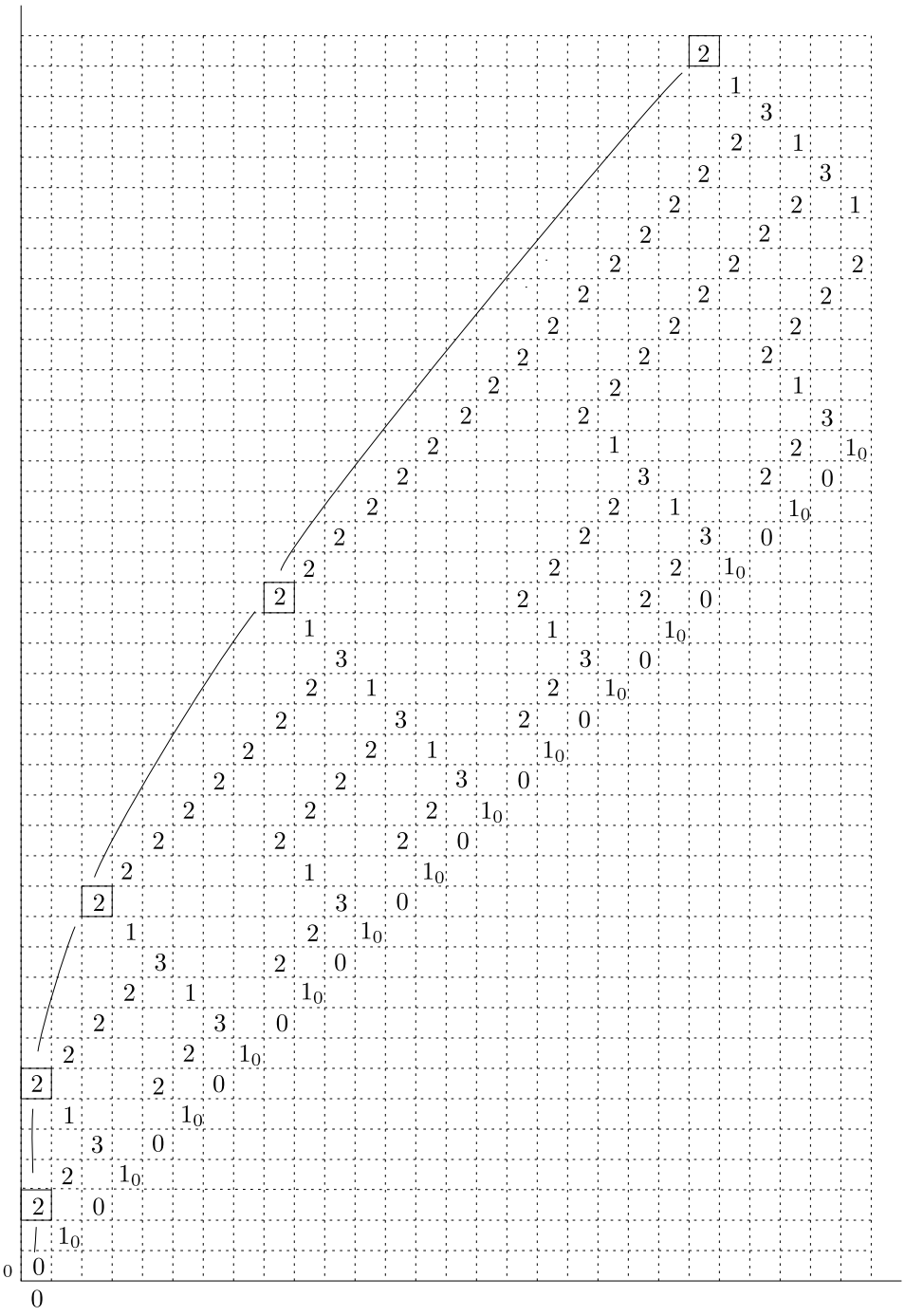


Fig.8a

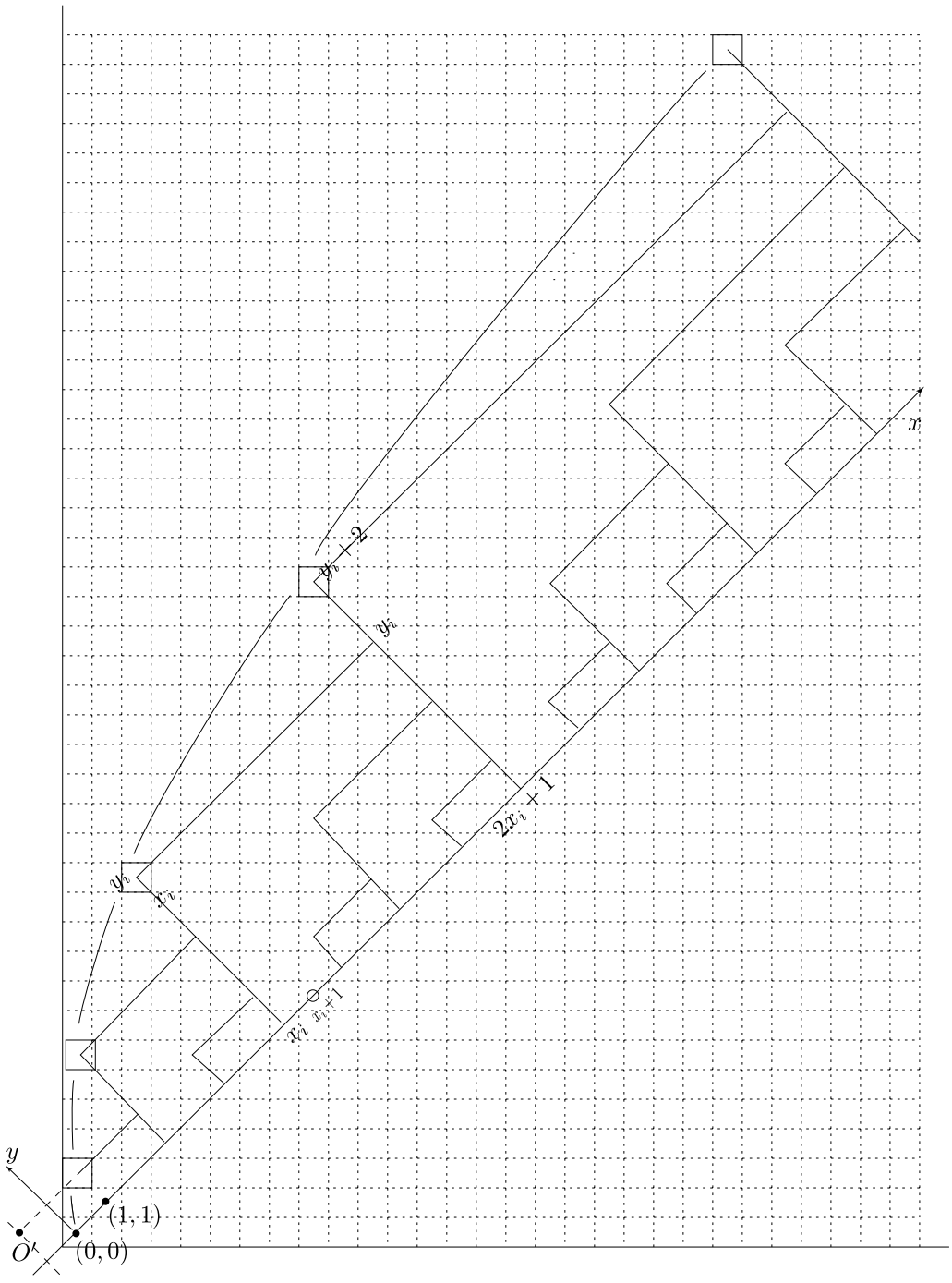


Fig.8b

We are particularly interested by sites of the convex envelope of the figure formed by the active sites the first of which are :

$$(c,t) = (0,2),(0,6),(2,12),(8,22),\ldots$$

To study the geometry of this figure we shall prefer the axis-system determined by

$$\text{origin } (0,0),\ \text{unit vectors of extremities } (1,1) \text{ and } (-1,1)$$

We are interested in points (squared in Figures 7)

$$\begin{aligned}(x,y) \quad = \quad & (1,1)\\ & (1+2,1+2)\\ & (1+2+4,1+2.2)\\ & (1+2+4+8,1+3.2) = (1+2+2^2+2^3,1+3.2)\\ & \ldots\end{aligned}$$

We can observe that, if one of these points is (x_i,y_i), from point $(x_{i+1},0)$ a new figure similar to the principal one developes, and that

$$x_{i+1} = 2x_i + 1$$

$$y_{i+1} = y_i + 2$$

and this leads us to ascertain that the points of the envelope are those of coordinates

$$x_i = 1 + 2 + \ldots + 2^i$$

$$y_i = 1 + 2i$$

for $i = 0,1,2,\ldots$. If we change the origin of axes for O' on site $(-2,0)$ and the unit-vectors for vectors two times longer, we find that our points are situated on curve

$$Y = log_2 X.$$

If we really want to come back to the axis-system corresponding to cells and time units, the curve will have parametrical equation

$$c_i = 2.2^i - 2i - 2$$

$$t_i = 2.2^i + 2i$$

for $i = 0,1,\ldots$, which is not as suggestive.

So set of states $\{0, 1_0, 1, 2, 3\}$ draws in the s.t.d a figure whose outline is a logarithmic curve.

But the interesting question is in fact the following : should this set of states be called signal ? It satisfies the definition we have given of a signal, but not our intuitive idea of a signal, because it has branches and junctions and looks more like a tangled hank. Well then, is it not a network, a little more complicated then our first example ? well no, because we cannot manage to separate the states in two (or more) subsets which would form distinct signals.

So this example leads us to make our definition of a signal more precise, particularly as concerns propagation, so as to exclude any branchings and junctions : the state (or word of states) at any time t must produce at time $t + 1$ one single state (or word of states), and this state (or word of states) must be produced by one single state (or word of states) at time t.

We shall not give a name to such a set as we have just described, and just marvel before the great variety of what c.a's can produce, that we have certainly not entirely explored.

5.1.7 Waves

For the needs of constructing Mazoyer's c.a in chapter 2, we have introduced particular waves : they are not signals for the reason that they are guided by their environment. They nevertheless are clearly visible objects, and active objects too, each of them generating new A, B, C signals.

To characterize them we shall define a new and larger category, the waves, simply by dropping the last requirement, that of indifference to environment.

So waves will be characterized by a finite number of states (or words- of-states), they will have continuous trajectories, but these trajectories, submitted to environing influences, will fluctuate and be more fanciful then the periodic trajectories of signals. Signals are particular waves.

The S_k waves of Mazoyer have state and word-of-state S and SS.

First examples of waves are to be found in the F.S.S.P solutions of Waksman [60] and Balzer [2], that we have not presented.

5.2 Fischer's c.a

Signals in networks may have very interesting trajectories,and we shall find a beautiful example in Fischer's c.a for recognizing prime numbers in strictly real time.

The c.a will have output 1, that is be in accepting state, at each prime time t. If we want the c.a to correspond exactly to the definition of a recognizer c.a, we just choose to represent numbers by as much 1 symbols (the nursery school representation), and agree that input is 1 at each time : so the accepted words are the sequences of t 1-symbols with t prime, that is the words on alphabet $\{1\}$ representing prime numbers.

As a first step we shall build a c.a where cell 0 is in accepting state at each time $3t$ with t prime, and in rejecting state at all other times. To obtain the announced for c.a we must then speed up three times. Speeding up we shall study in chapter 7 only, so the last section may be left over for later reading, but it can also give a first idea of speeding up in a very simple case.

5.2.1 Geometrical solution

To begin with let us remind ourselves of Eratosthenes' riddle : starting point is the infinite sequence of integers

$$2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ \ldots\ldots\ldots,$$

we retain the first one, 2, and cross out all its multiples, we then retain the first next integer left, which is a prime, and cross out its multiples, and so on, progressively eliminating all integers which are not primes

$$\underline{2}\ 3\ \not{4}\ 5\ \not{6}\ 7\ \not{8}\ 9\ \not{10}\ 11\ \not{12}\ 13\ \not{14}\ 15\ \not{16}\ \ldots$$

$$\underline{2}\ \underline{3}\ \not{4}\ 5\ \not{6}\ 7\ \not{8}\ \not{9}\ \not{10}\ 11\ \not{12}\ 13\ \not{14}\ \not{15}\ \not{16}\ \ldots$$

$$\underline{2}\ \underline{3}\ \not{4}\ \underline{5}\ \not{6}\ 7\ \not{8}\ \not{9}\ \not{10}\ 11\ \not{12}\ 13\ \not{14}\ \not{15}\ \not{16}\ \ldots$$

Fischer uses a similar but less efficient method : he crosses out the multiples of every integer, (thereby loosing his time in vain for integers not prime), starting from their square, which is the first multiple possibly not yet crossed out.

The first idea we can have to try implement this is that cell 0 should emit at each time t a signal criss-crossing in a $t/2$ large corridor and crossing out cell 0 at all kt times (Figure 8). Unfortunately such signals will accumulate more and more numerous with time on the first cells, and such a situation cannot be managed by a c.a, whose number of states is finite.

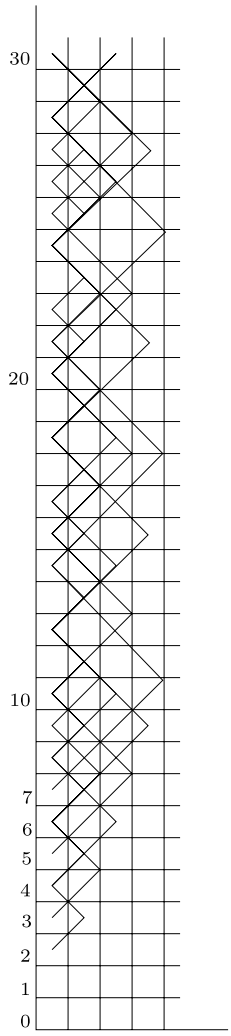


Fig.9

So next we think about separating this hank of signals by giving each of them a separate corridor. These corridors will be further and further from cell 0, so cell 0 can no more be crossed out by the criss-crossing signals, new signals regularly started by the latter must do that. Figure 9 tries to suggest the main features of the mechanism.

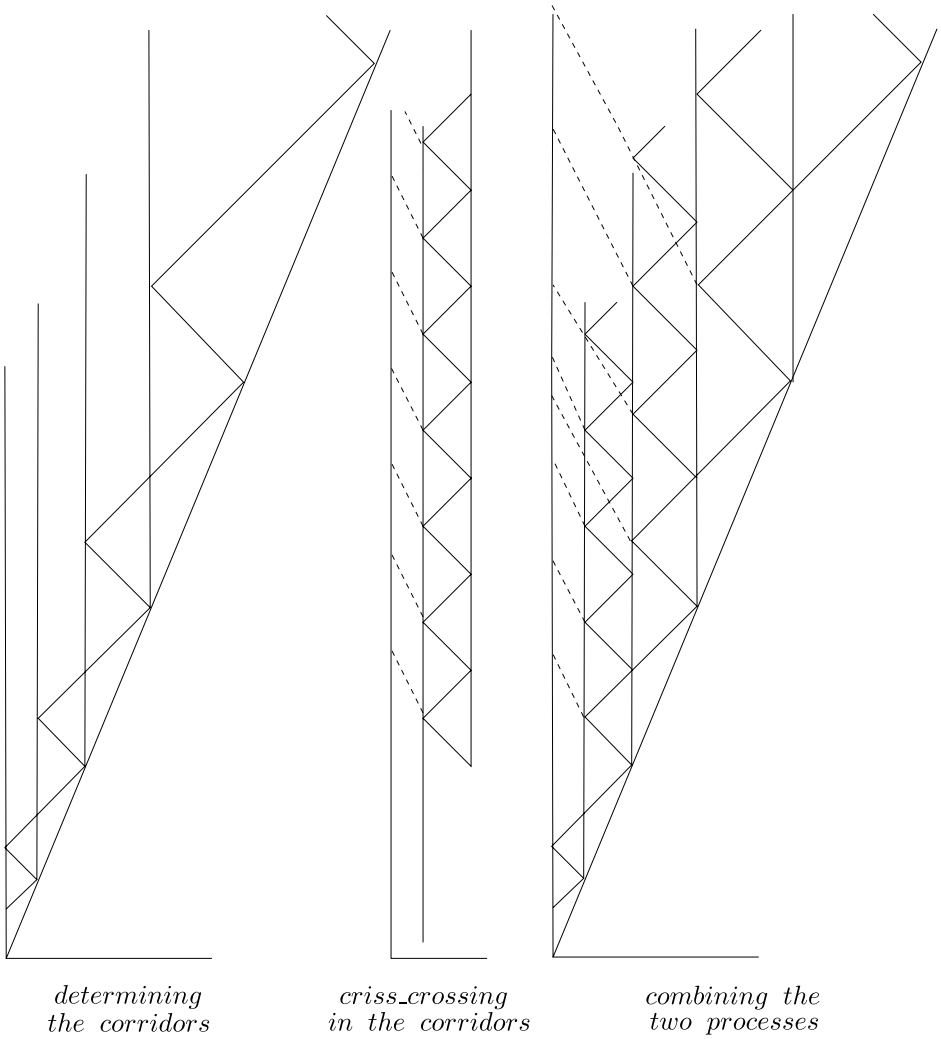


Fig.10

We shall have to

- delimit the corridors
- start the criss-crossing signals
- start the crossing out signals

If we solve the preceeding little problems with ordinary continuous geometry, we find that

- if width of the successive corridors must increase regularly of one unit, slope of the main signal must be 3, and a small correction

must be made : main signal should gain one time unit each time it receives the delimiting signal, and each reflection of the latter on the preceeding corridor should be delayed by one time unit (Figure 10)

- if signal criss-crosing in k-wide corridor should determine $3k$-time intervals, we can give them slopes 2 and -1 in turn

- for crossing out signals to arrive on cell 0 at times $3(k^2 + \mathbb{N}k)$ we must give them slope -3

This is how we find the skeleton of Fischer's c.a, but it is not like this that Fischer gives it.

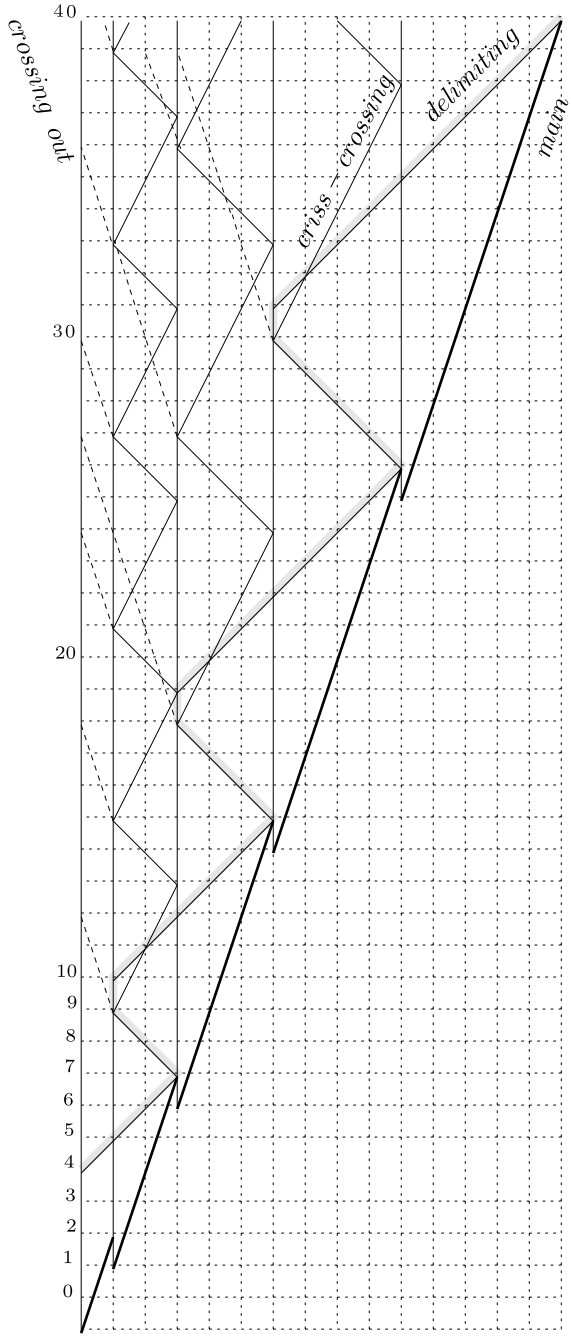

40
30
20
10
9
8
7
6
5
4
3
2
1
0
crossing out
criss - crossing
delimiting
main

Fig.11

5.2.2 Fischer's c.a

Fisher presents his c.a [16] by directly giving the rules, in terms of intercellular signals, which is a little unusual. We shall now try analyzing his rules.

Cells themselves may be in two different states, N the normal state and P the partition state (indicating that the cell delimits a corridor). Besides he uses 3 signals, which can go left or right, so this amounts really to 6 signals

- b rightwards for the main signal, b leftwards for the crossing- out signals (slopes 3 and -3)

- a for creating the corridors (slopes 1 and -1)

- c for the criss-crossing (slopes 2 and -1).

Let us analyse Fischer's words to recollect their meaning concerning the cells and states. When he writes : "if b arrives from the left on some cell in state N, it leaves this cell 3 time units later going rightwards ", we imagine something like Figure 11a. To translate this way of expressing in terms of cells and states, a signal being a state propagating from cell to cell, we can draw Figure 11b. To implement this, b must be separated in 3 states, b, b_1, b_2 (Figure 11c). At last we may choose for the states symbols more suggestive than letters (Figure 11d).

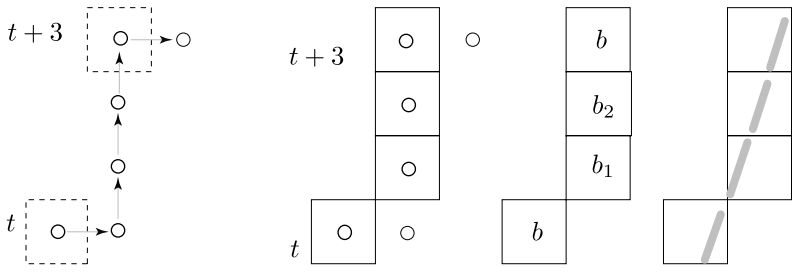


Fig.12

In Figure 12 we give the 12 rules of Fischer and we translate them in the same way, then in Figure 13 we use them to build the beginning of the s.t.d.

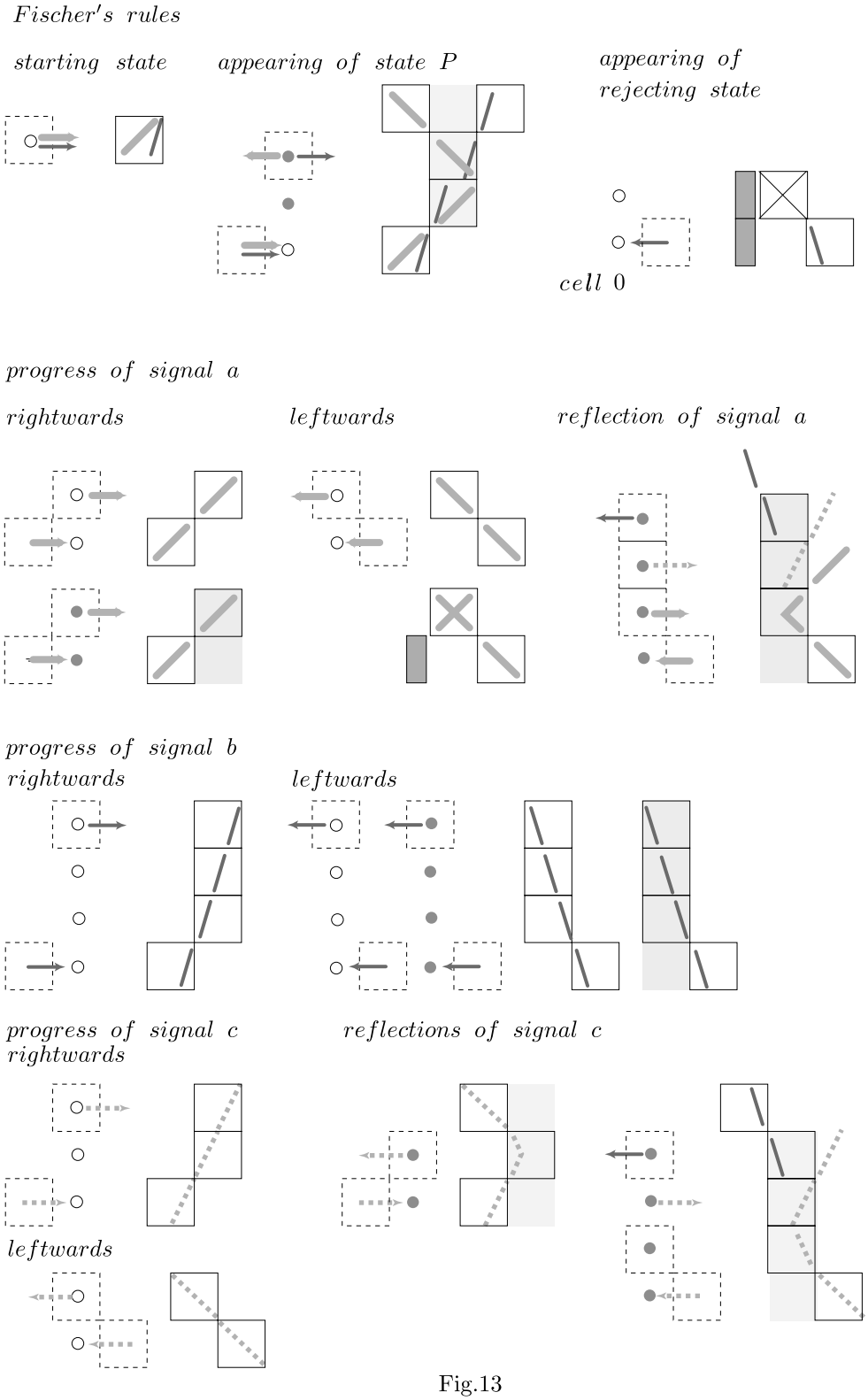

Fischer's rules
starting state
appearing of state P
appearing of
rejecting state
cell 0
progress of signal a
rightwards
leftwards
reflection of signal a
progress of signal b
rightwards
leftwards
progress of signal c
rightwards
reflections of signal c
leftwards

Fig.13

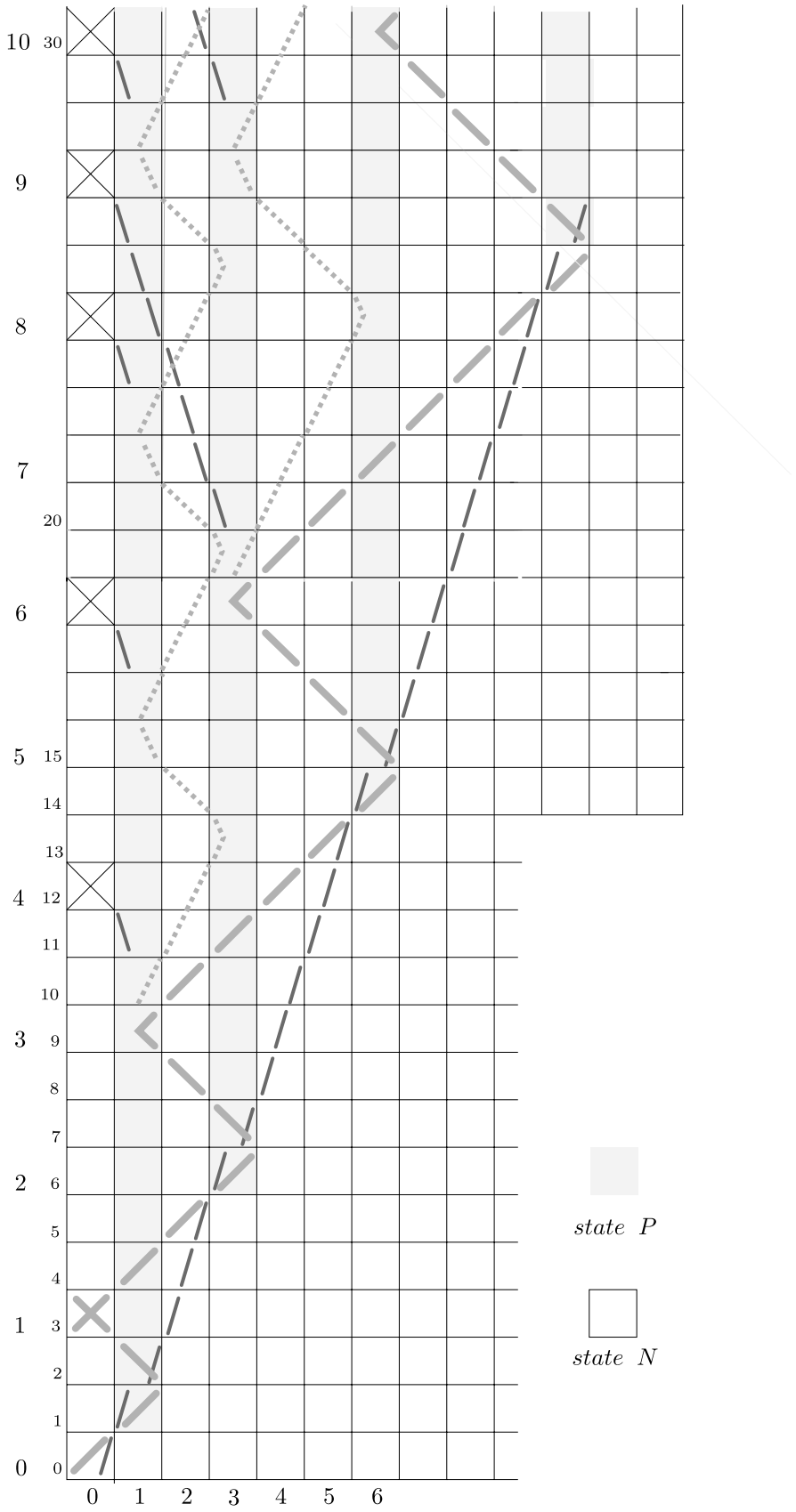

10
30
9
8
7
20
6
5
15
14
13
4
12
11
10
3
9
8
7
2
6
5
4
1
3
2
1
0
0
0 1 2 3 4 5 6
state P
state N

Fig.14

On Figure 12 we can count the states, and we find there are 25 of them.

5.2.3 Proof

We must now check that the c.a does what it is meant to. The next two lemmas will do for that.

Lemma 5.2.1 *for all $k \geq 2$, cell $k(k+1)/2$ enters state P at time $(3k^2+k)/2-1$ and sends signal a back on cell $(k-1)k/2$ at time $(3k^2+3k)/2$.*

Calculations are elementary. This result is correct from $k = 1$ (cell 1) up. It is clear that only cells $k(k+1)/2$, $k \geq 1$ can enter state P.

Lemma 5.2.2 *signals b emitted by cell $k(k-1)/2$ arrive on cell 0 at times $3(k^2 + \mathbb{N}k)$.*

Once more, calculations are elementary.

So, at all times $3(k^2 + nk)$, $n \in \mathbb{N}$, cell 0 is in rejecting state r.

5.2.4 Speeding up

From this c.a we have just built, we shall now build a new one : its cells will be the triples of three consecutive cells

$$(c_0, c_1, c_2) \dots (c_{3i}, c_{3i+1}, c_{3i+2}) \dots \dots .$$

So states will be triples of the states we had : the number of triples is 25^3, but it is clear that all of them do not appear in the s.t.d, where we could count the useful triples.

Now then, to speed things up, we simply rub out 2 time steps out of 3, so that the new s.t.d has only the following sites left

$$< (3i, 3i+1, 3i+2), 3t >= (< 3i, 3t >, < 3i+1, 3t >, < 3i+2, 3t >).$$

These two operations we have done to obtain the s.t.d of Figure 14.

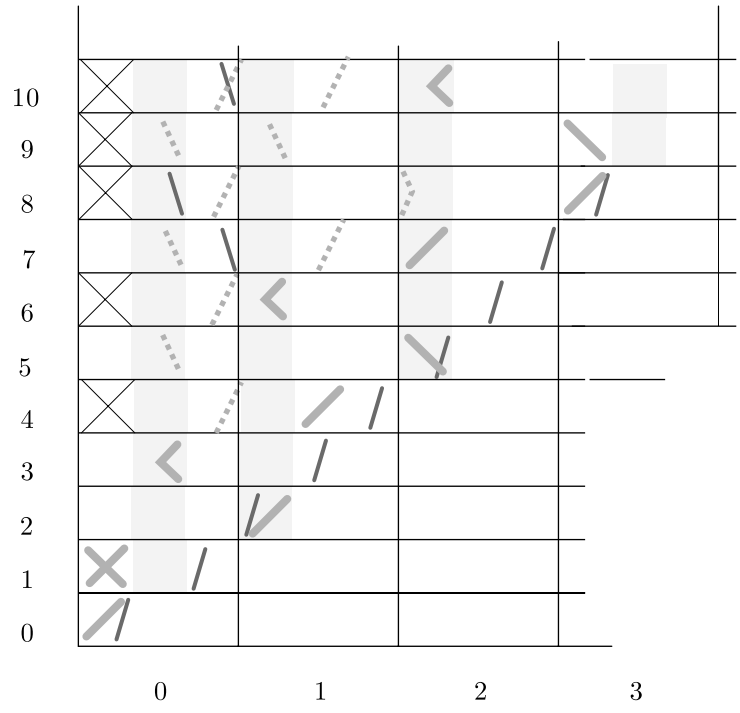


Fig.15

The rules of the new c.a, (which is also semi-infinite, of scope 1, with an impulse initial state), can be obtained by operating three (old) transitions on the 9 (old) states of the 3 triples, and forgetting the two first steps. We guess the rules are many, quite unreadable, progression of signals being chopped by a stroboscopic process. But for Fischer's result only their existence is important : the output function having value 0 on all triples begining with rejecting state r, and value 1 on the others, will have value one at all prime times.

And in the first c.a signals a, b and c form a beautiful example of a network.

5.3 Waves generated by an impulse c.a

5.3.1 Functioning of an impulse c.a

In this first part we shall discover a periodical functioning, whose origin is basically the quiescent state of all sites under the diagonal. We particularly insist here that the initial state must be an impulse on cell 0.

Let D_i be the diagonal from site $(0, i)$, so D_0 is the principal diagonal and D_{-1} is the diagonal just under, from site $(1, 0)$. And let q be the number of states of the c.a. At last, let us note once and for all that the number of cells before cell c is c, as the first cell has number 0.

□

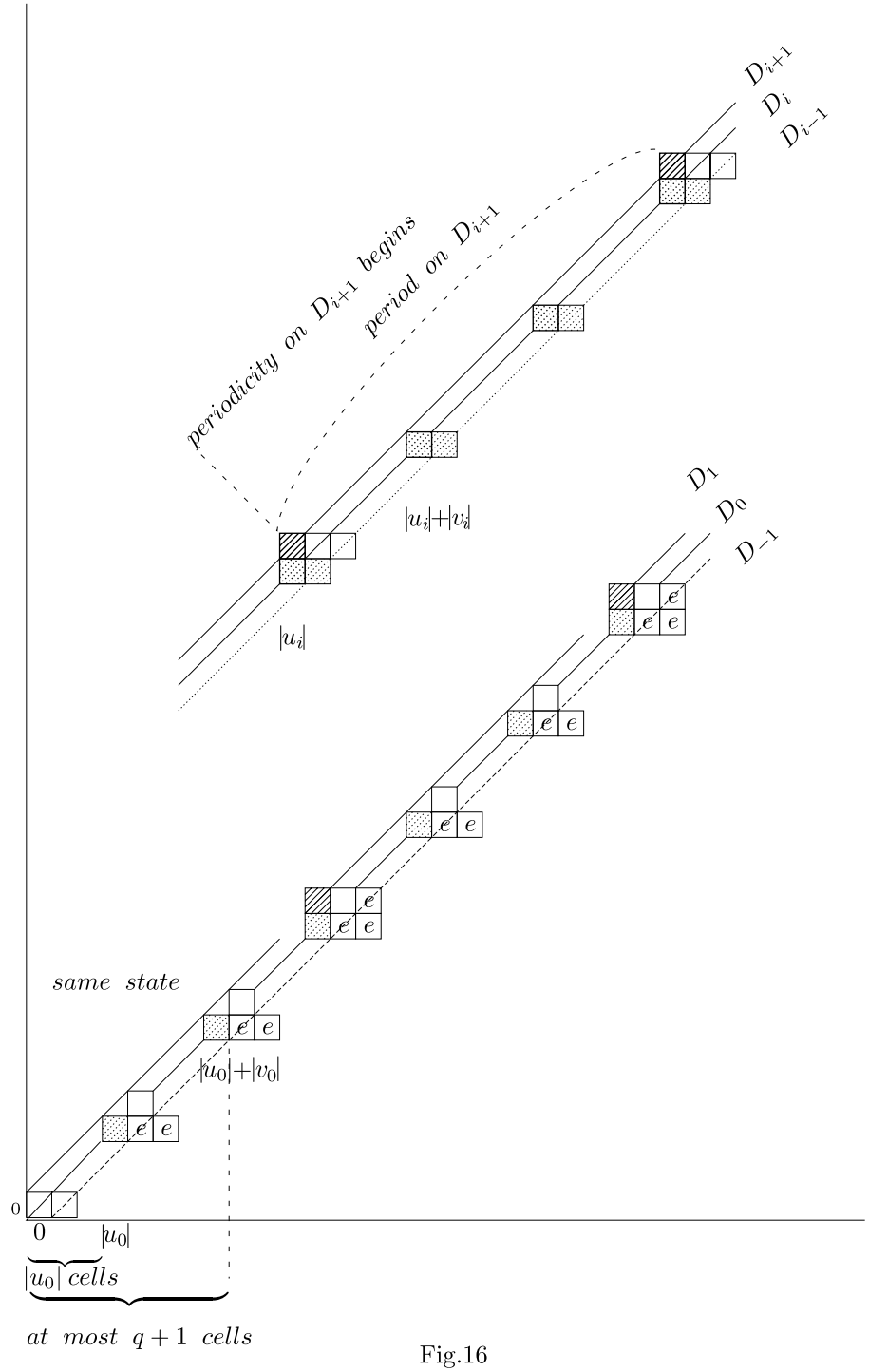

D_{i+1}
D_i
D_{i-1}
periodicity on D_{i+1} begins
period on D_{i+1}
$|u_i|+|v_i|$
$|u_i|$
D_1
D_0
D_{-1}
same state
$|u_0|+|v_0|$
0
0
$|u_0|$
$|u_0|$ cells
at most $q+1$ cells

Fig.16

Let us now remind observations made by V.Terrier [54]. First we notice that, among the $q+1$ first sites of D_0, two must be in the same state (see Figure 15). As a consequence, the next states on the diagonal, resulting from transition applied to the same triples (the state and two quiescent states on the right), are also identical, and so on for the next ones .... So the semi-infinite word w_0 of states on D_0 is of the form

$$w_0 = u_0 v_0^\omega$$

where v_0^ω is the infinite repetition of word v_0, with

$$|u_0| \geq 0 \quad |v_0| \geq 1 \quad et \quad |u_0| + |v_0| \leq q.$$

Let us now examine diagonal D_1, and more precisely the $q+1$ sites of cells

$$|u_0| \ , \ |u_0| + |v_0| \ \ \ldots \ \ |u_0| + q|v_0|.$$

Two of these sites ar necessarily in the same state. The two states on the right of these are also identical, the first ones being on D_0 after perodicity has started and at a distance of several periods, the second ones being quiescent. As a consequence the following sites on D_1 are identical, so the word of states on D_1 is of the form

$$w_1 = u_1 v_1^\omega$$

with

$$|u_1| \geq |u_0|$$

and

$$|u_1| + |v_1| \leq |u_0| + q|v_0| \leq q(|u_0| + |v_0|) \leq q^2$$

$$v_1 \quad \text{a multiple of} \quad v_0.$$

Let us now do for diagonal D_i the following induction hypothesies :

$$w_i = u_i v_i^\omega \quad |u_i| + |v_i| \leq q^{i+1}$$

$$D_{i-1} \text{ has periodicity } v_i$$

$$\text{from cell } |u_i| \text{ (or before)}$$

and let us notice that these hypothesies are satisfied for D_1. On diagonal D_{i+1}, among sites on cells

$$|u_i| \ , \ |u_i| + |v_i| \ \ \ldots \ \ |u_i| + q|v_i|,$$

two must be in the same state : as this is also the case for the two sites on the right because of the periodicities on D_i and D_{i-1}, so we have on D_{i+1} a periodicity which is a multiple of $|v_i|$:

$$w_{i+1} = u_{i+1}v_{i+1}^{\omega}$$

with, clearly :

$$|u_{i+1}| \geq |u_i|$$

$$|u_{i+1}| + |v_{i+1}| \leq |u_i| + q|v_i| \leq q(|u_i| + |v_i|) \leq q^{i+2}$$

D_i certainly has periodicity $|v_{i+1}|$

from cell $|u_i| \leq |u_{i+1}|$

So we have prooved by induction

Proposition 5.3.1 (V.Terrier) *on each diagonal D_i of the s.t.d of an impulse c.a the word of states is of the form $w_i = u_i v_i^{\omega}$ with $|u_i| + |v_i| \leq q^{i+1}$.*

5.3.2 Gap theorem for a wave

We focus now on waves that an impulse c.a could produce in its active area. And for the while to threadlike waves.

The tool we shall use here is preceding proposition. According to this proposition, a wave S appearing anywhere in the active area of an impulse c.a, if it stays q^{i+1} units of time on diagonal D_i, must then stay on this diagonal definitely : indeed as a complete period of the states on D_i is formed of states of S, on all the rest of D_i we have states of S, the wave is on D_i. A wave that wouldn't, after possibly hesitating at first, have maximal speed, cannot progress at maximal speed for too long, its stay on each diagonal is limited in time.

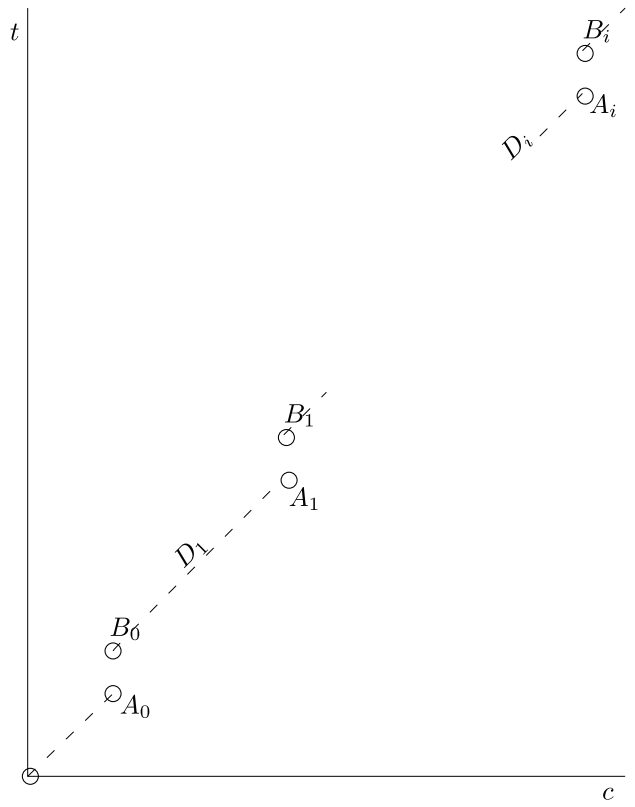


Fig.17

Let us examine a wave starting on site $(0, 0)$. The line formed (see Figure 16) by q sites of D_0 (last site A_0), q^2 sites of D_1 (first site B_0, last site A_1), ... q^{i+1} sites of D_i (last site A_i) is a frontier that the wave cannot overstep, nor even step on. (Mark here that we do not choose the tighter formulas, with $q-1$, q^2-1 ..., because the result would not arrange so neatly !) We shall now make this frontier more precise :

$$A_0 = (q-1, q-1) \qquad B_0 = (q-1, q-1+1) = (q-1, q)$$

$$A_1 = (q-1+q^2-1, q+q^2-1) \quad B_1 = (q-1+q^2-1, q+q^2)$$

...

If cell and time corresponding to point (site in fact) B_i are c_i and t_i we have

$$t_i = q + q^2 + \cdots + q^{i+1} = q\frac{q^{i+1}-1}{q-1}$$

$$c_i = t_i - (i+1)$$

then (from first of the two preceding formulas)

$$i + 1 = \log_q \left[1 + (1 - \frac{1}{q}) t_i \right]$$

and at last

$$c_i = t_i - \log_q [1 + (1 - \frac{1}{q}) t_i].$$

As for points A_i, they are situated on curve $\mathcal{C}$ (Figure 17), whose equation is

$$c = (t+1) - \log_q [1 + (1 - \frac{1}{q})(t+1)].$$

Every straightline having equation $t = c + a$ sooner or later crosses $\mathcal{C}$, (at time $t = \frac{q}{q-1}(q^{a+1} - 1) - 1 \approx q^{a+1}$).

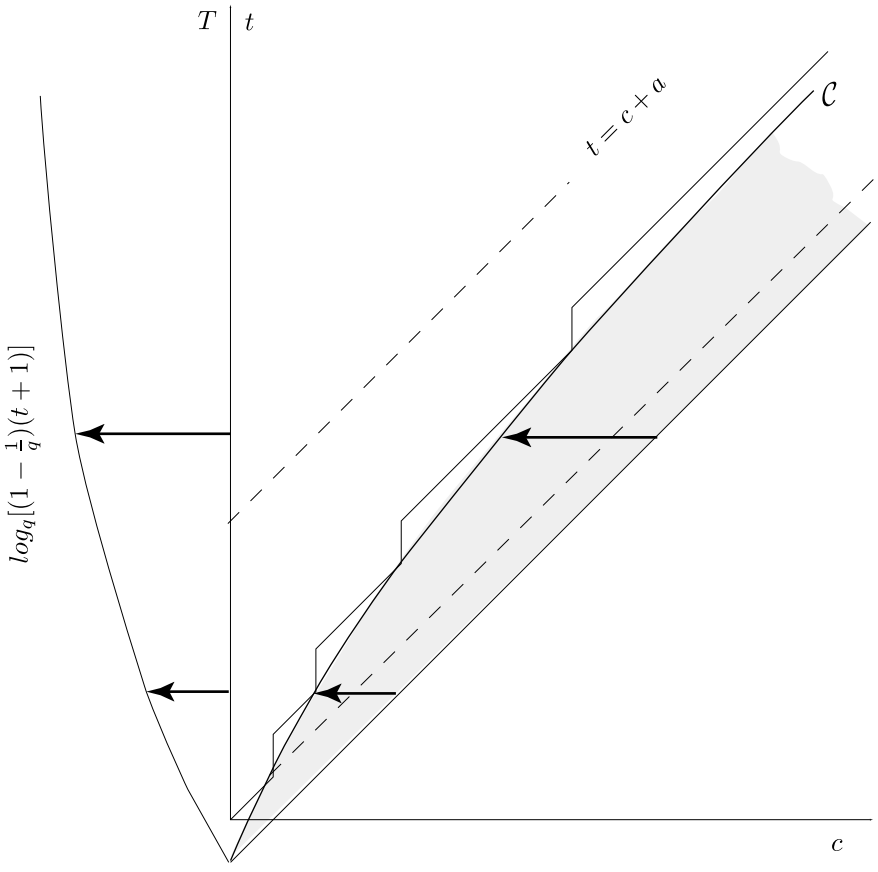


Fig.18

From all this we draw the gap theorem for a threadlike wave, and remoter conclusions.

Theorem 5.3.2 (V.Terrier) *no wave starting on site* $(0,0)$*, unless it becomes the maximal speed signal, can penetrate between diagonal* D_0 *and curve* $\mathcal{C}$*.*

This means that the wave cannot progress too quickly, it must stay at a distance from D_1 logarithmic in $t+1$, at least.

In fact this result is not limited to waves issued from site $(0,0)$. It extends (Figure 18) to any wave from any of its sites, $(c, c+i)$ provided periodicity on D_i has started before cell c, which is certain if $c \geq q^{i+1} \geq |u_i|$: indeed, the wave cannot linger q^{i+1} units of time on D_i, nor q^{i+2} units of time on $D_{i+1} \ldots$, so it cannot penetrate in the area situated between D_i and the curve obtained by shifting the portion of curve $A_i\mathcal{C}$ from site $(c+q^{i+1}-1, c+i+q^{i+1}-1)$. But really this takes place very far away because q^{i+1} must be very big !

At last, the same result can also extend to thicker waves. Let us indeed consider the right border of a wave : its states belong to set B of states appearing as last state of the words of states of the wave. In as much as the wave must distinguish itself from the background, it is reasonable to admit that the background has no states of B too near this right border. In this respect, the right border, exactly like the case was for a threadlike wave, unless it goes at maximal speed, cannot linger on the diagonals, and then it stays behind by a delay at least logarithmic in t.

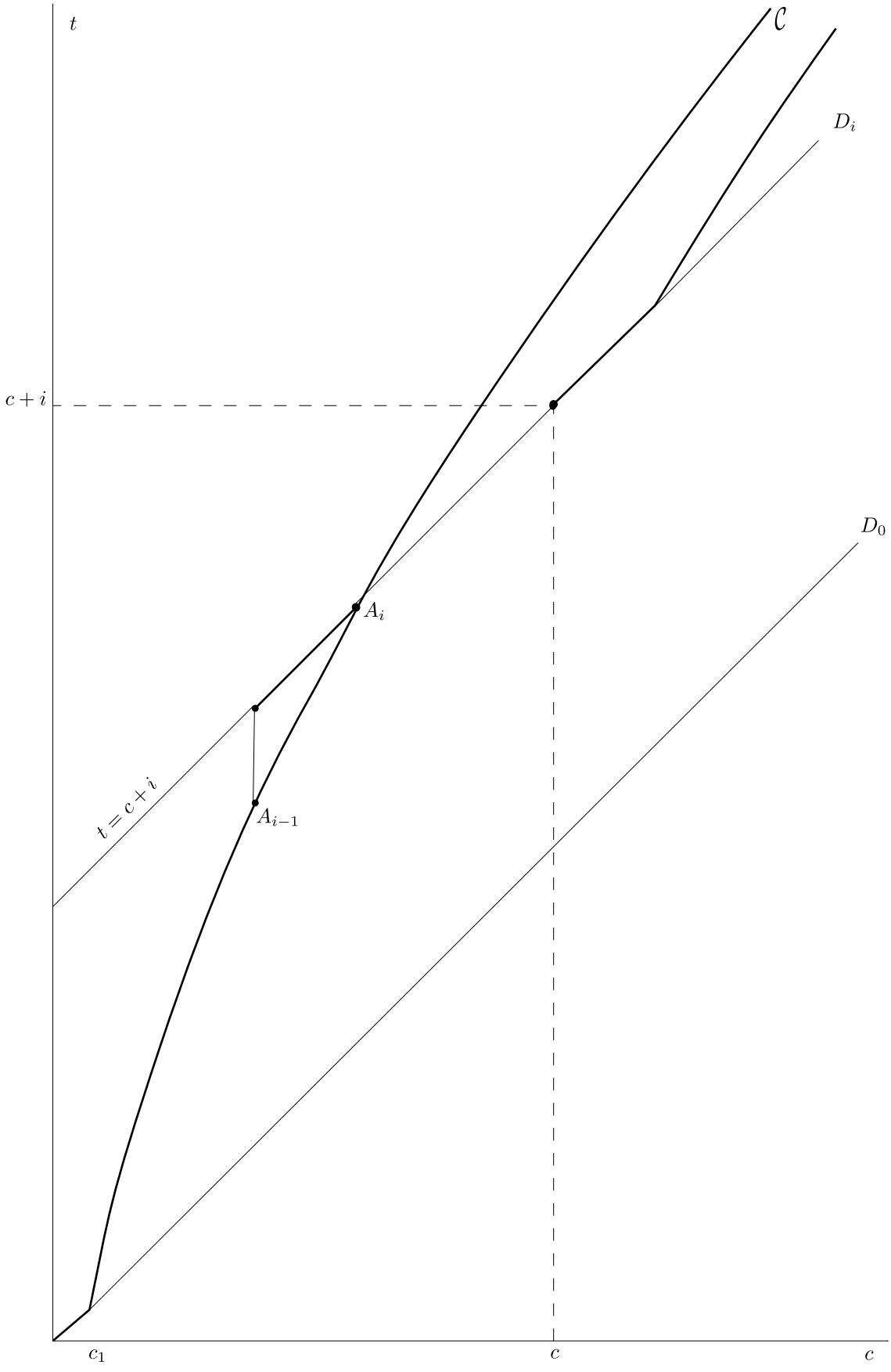


Fig.19

Chapter 6

Slowing down

Ideas for this chapter were found in Véronique Terrier's thesis [54].

6.1 Weak slowing down

It may seem queer, in a world where saving time and speeding up is a major concern, to study how to manage slowing down. We have however three reasons for doing it :

- slowing down will be useful for synchronization problems

- its study is remarkably easy, and nevertheless instructive before we study speeding up

- it accidentally leads to solving some other interesting problems.

So we want to construct a new c.a $\mathcal{A}'$ accomplishing the same task as c.a $\mathcal{A}$, but k times slower. We have the very simple idea of equiping $\mathcal{A}$ with a k-steps cyclic counter, which will lock $\mathcal{A}$ for $k-1$ steps and let it run only one time step : let this step be 0, and the other ones 1, 2, ... , $k-1$. If counter is started on 0, the active time-steps will be : 0, k, ... , kt, ... If it is started on 1, the active time-steps will be : $k-1$, ... , $kt+k-1$, ...

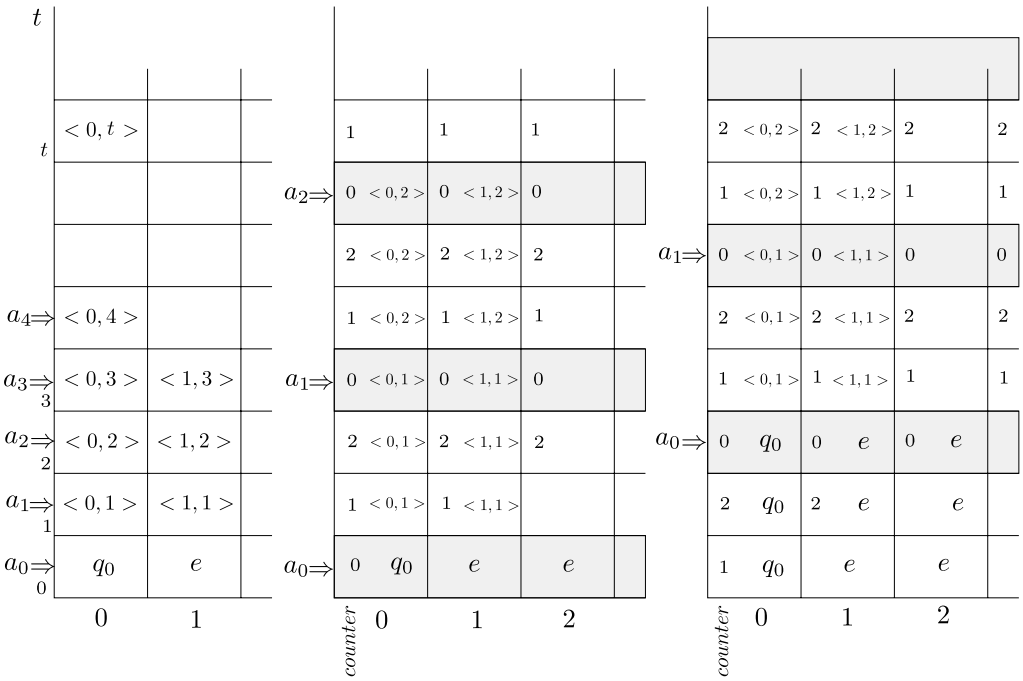


Fig.1

In Figure 1 (where $k = 3$), $< c, t >$ stands for the state of site (c, t) of $\mathcal{A}$. Initial state of $\mathcal{A}'$ will be $(0, q_0)$ or $(1, q_0)$, q_0 being initial state of $\mathcal{A}$; the counter of a cell, once started, starts the counter of next cell. In Figure 1 the active time steps are coloured. As for the inputs, we must enter them one time -step out of k, inputs at inactive times being indifferent : so the new input word may be

$$\underbrace{a_0, a_0, \ldots, a_0}, \underbrace{a_1, a_1, \ldots, a_1}, \ldots$$

or, if counter starts on 0

$$\underbrace{a_0, a, a, \ldots, a}, \underbrace{a_1, a, a, \ldots, a}, \ldots$$

and if counter starts on 1

$$\underbrace{a, a, \ldots, a, a_0}, \underbrace{a, a, \ldots, a, a_1}, \ldots$$

where a is any element of input alphabet . We could even take as new input any k-coding of the old input letters : cell 0 could memorize them, recognize the old input after reading k symbols, work with it, or enter a rejecting state if the sequence of k code symbols corresponds to no old input letter.

In any case, the input has to undergo some transformation : there lies the defect pointed at by the *weak* adjective.

As for the outputs, they could admit interruptions. But our main concern here will be recognizers and synchronizers.

The new recognizing or synchronizing time will be

- $k(T-1)+k-1+1 = kT$ if counter was started on 1
- $k(T-1)+1 = kT-(k-1)$ if counter was started on 0

T being the old recognizing or synchronizing time (Figure 2).

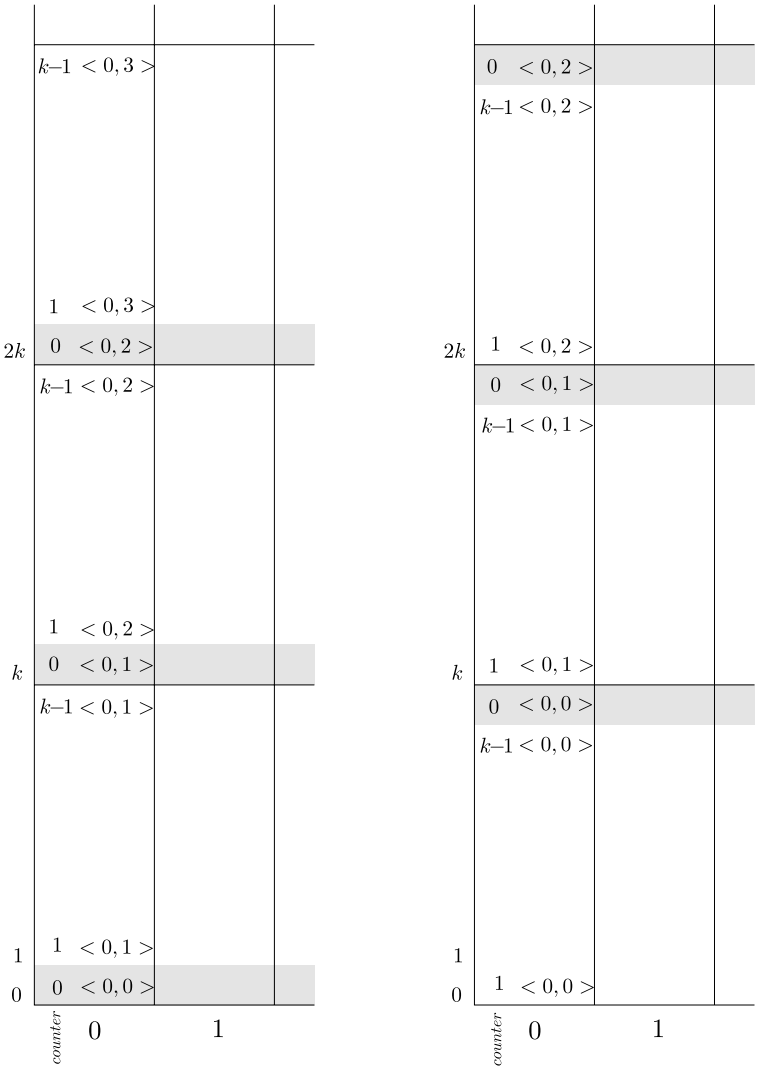


Fig.2

6.2 Strong slowing down

To obtain this strong slowing down, we shall simply add to the slow c.a a little mechanism sending every input at time t back on cell 0 at time kt. Idea of this mechanism is suggested in Figure 3, representing space $\mathbb{R}^2$ with k any real number.

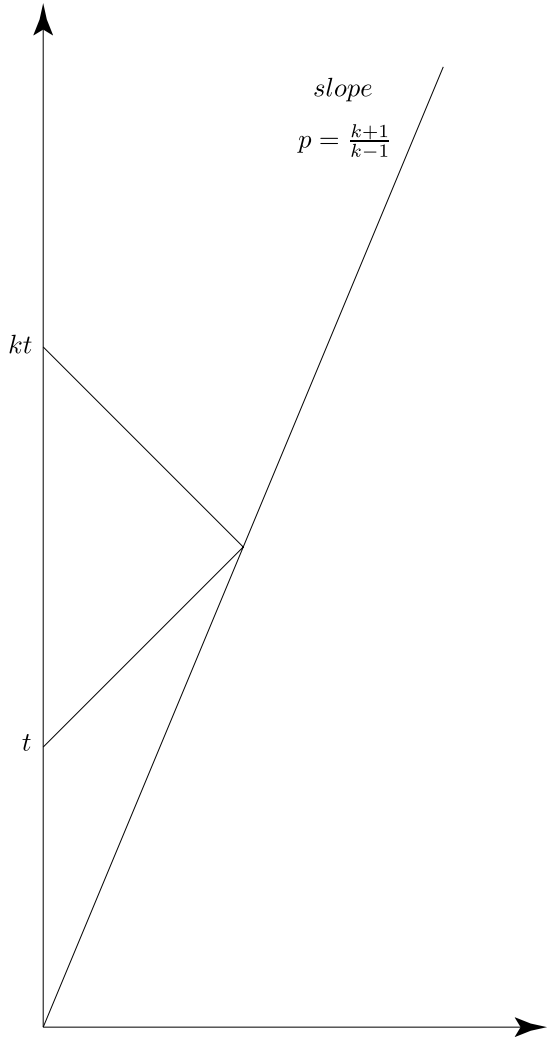


Fig.3

We shall adapt it in our discrete environment and implement it precisely in the case when counter starts on 1 and the new (recognizing or synchronizing) time is exactly kT. For the case when the counter starts on 0 we just give the solution.

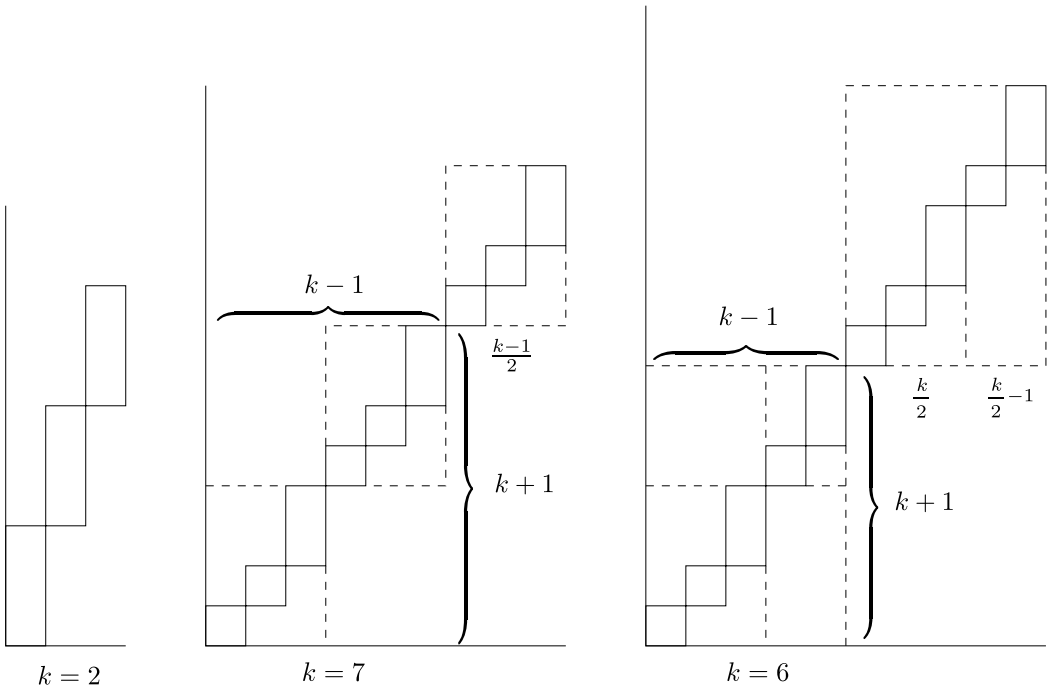


Fig.4

Let us first set up signal S starting from site $(0,0)$ and having slope $p = \frac{k+1}{k-1}$ (Figure 4) :

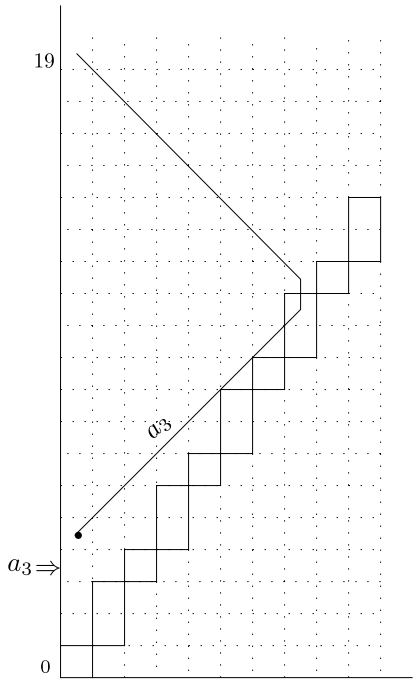
- if k is odd : to progress by $k-1$ cells in $k+1$ time-steps, S will repeat $(k-1)/2$ moves rightwards and one stand still

-if k is even : to progress by $k-1$ cells in $k+1$ time-steps, S will repeat

- $k/2$ moves rightwards and one stand still (fast segment)
- then $k/2-1$ moves rightwards and one stand still (slow segment)

For signal S to start exactly as we wish (that is as in Figure 4), we must suppress the very first move rightwards, or equivalently, take as fictive starting site $< -1, -1 >$.

Input entering at time t will travel, from time $t+1$, at maximal speed and reflect, possibly with some delay, on S. We want it to arrive on cell 0 at active time t which is in fact time $kt+k-1$ (see Figure 2).



$k = 5$

Fig.5a

In case k is odd (Figure 5a) : sites of S that may receive an input arriving at maximal speed are sites situated after a stand still

$$\left(-1 + i\frac{k-1}{2}, -1 + \frac{k+1}{2}\right) \qquad i \geq 1,$$

trajectory of input a_t is formed of sites

$$(x, t + 1 + x),$$

site where meeting takes place is

$$\left(-1 + \frac{k-1}{2}(t+1), -1 + \frac{k+1}{2}(t+1)\right),$$

input reflected with delay 1 locates on sites

$$\left(-1 + \frac{k-1}{2}(t+1) - i, -1 + \frac{k+1}{2}(t+1) + 1 + i\right),$$

it arrives on cell 0 at time

$$-1 + \frac{k+1}{2}(t+1) + 1 - 1 + \frac{k-1}{2}(t+1) \;=\; kt + k - 1.$$

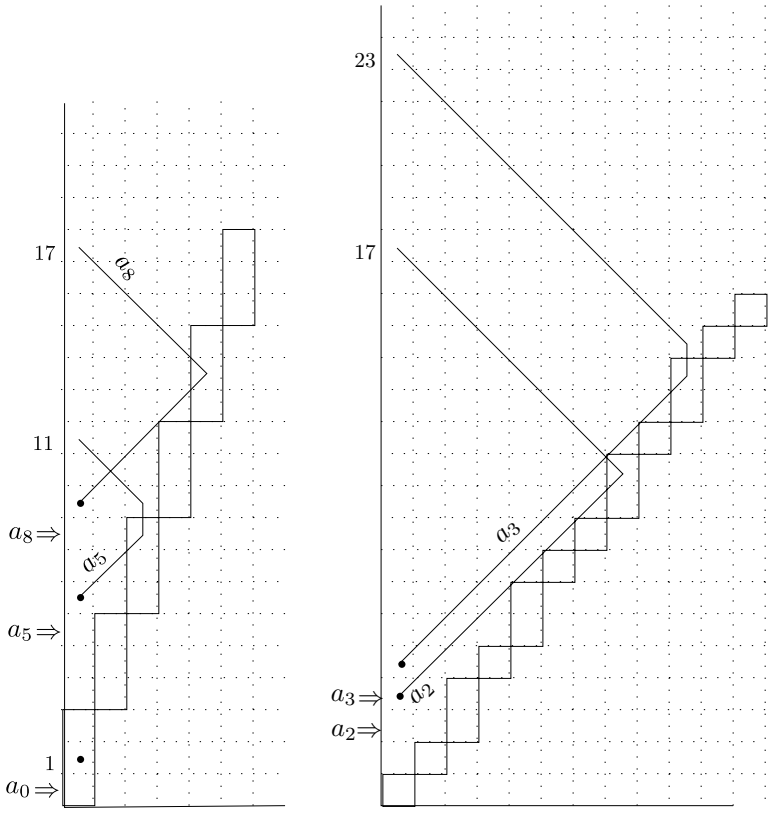


Fig.5b

In case k is even (Figure 5b) : sites of S that may receive an input are those of type 1 (at the end of a fast segment)

$$\left(-1 + i(k-1) + \frac{k}{2}, -1 + i(k+1) + \frac{k}{2} + 1\right) \qquad i \geq 0$$

and those of type 2 (at the end of a slow segment)

$$(-1 + i(k-1), -1 + i(k+1)) \qquad i \geq 1,$$

the first will receive inputs a_t entered at even t times, the second inputs entered at odd t times, and this on sites respectively

$$(-1 + (k-1)\frac{t}{2} + \frac{k}{2}, -1 + (k+1)\frac{t}{2} + \frac{k}{2} + 1)$$

$$(-1 + (k-1)\frac{t+1}{2}, -1 + (k+1)\frac{t+1}{2}).$$

Inputs reflected

without delay on type 1 sites (at the end of a fast segment)

with delay 1 on type 2 sites (at the end of a slow segment)

arrive on cell 0 at time $kt + k - 1$.

Let us sum up :

- for k odd : reflection is with delay 1

- for k even : reflection is without delay on type 1 sites and with delay 1 on type 2 sites

In case we had wished the counter to start on time 0, and so inputs to arrive on cell 0 at times kt, we should have used signal S', one time-step late after S. Input a_0 should be used directly.

Slow c.a $\mathcal{A}'$ will have threefold states :

- a first part will devote to reflecting inputs, it may contain two elements, a state of S and an input letter

- a secund part will hold the k states counter

- in a third part we shall find old c.a $\mathcal{A}$ (locked by the counter most of the time)

6.3 C.a computing a word morphism

We shall now make the most of the mechanism reflecting inputs that we have just set up, which mainly uses signals of slope $p = k + 1/k - 1$. We remind this mechanism in Figure 6, where we can see two speed 1- signals following at a distance of one time unit reflected by signals of slope $k + 1/k - 1$ and becoming two signals of speed -1 following at a distance of k time-units, this for different values of k.

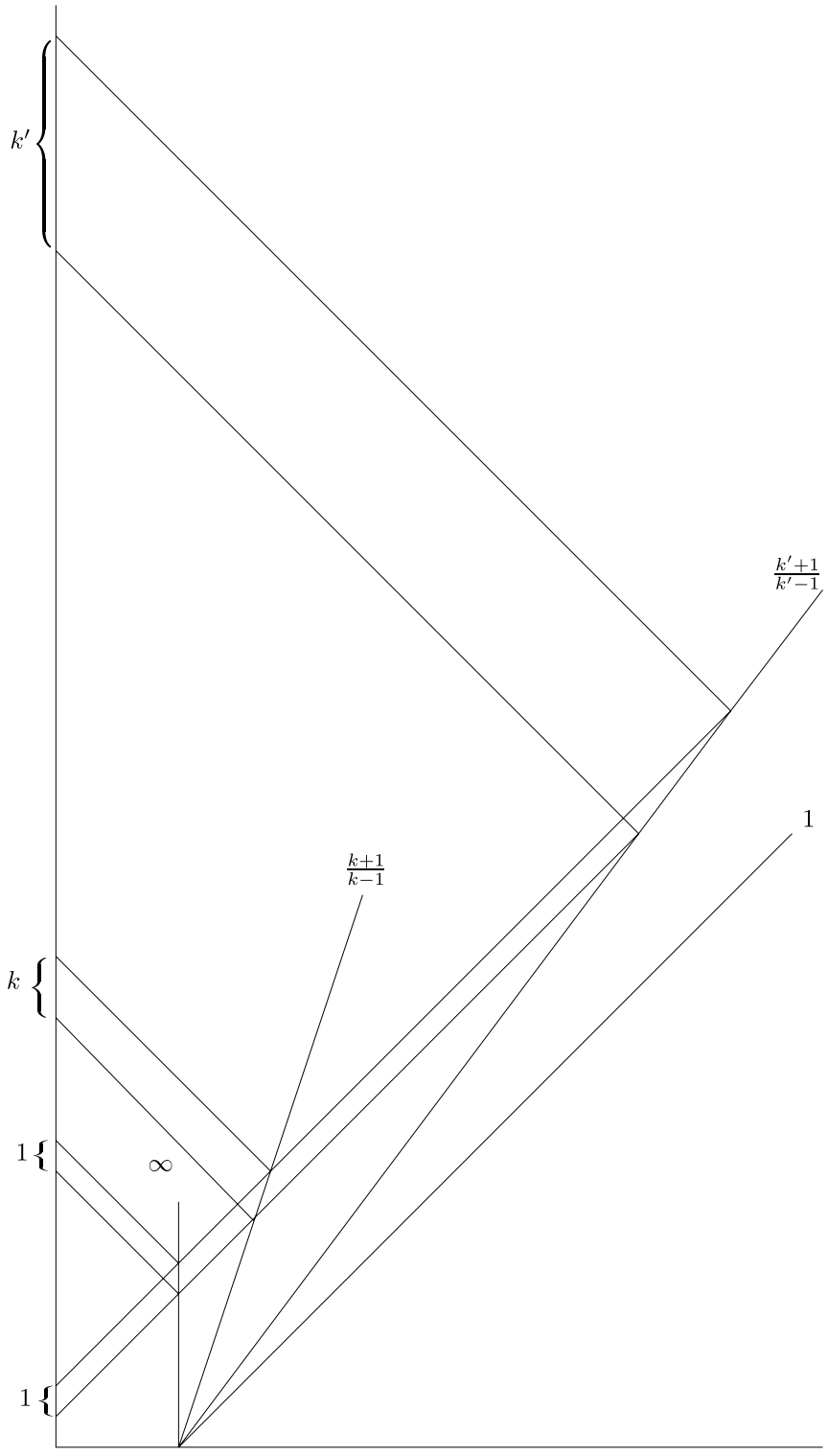


Fig.6

Let us remind that a morphism h of the monoid of words on alphabet A onto the monoid of words on alphabet B is a mapping that sends a product (concatenation) of words on the product of their images, and the empty word (that has no letter and is designed by 1 whatever the alphabet) on the empty word. It is caracterized by the set of the images of the letters of A : if w is the word $a_0a_1 \dots a_n$, then

$$h(a_0a_1 \dots a_n) = h(a_0)h(a_1) \dots h(a_n)$$

6.3.1 A first simple model

with which we get $h(w)$ with interruptions.

We choose k an integer larger than all lengths $|h(a)|$ of the words that are images of the letters in A. The main feature of the c.a will be a signal S from site $(0, 0)$ and having slope $p = k + 1/k - 1$.

In Figure 7 we have shifted this signal 2 cells rightwards so things are less confused near the origin, the flaw we thus introduce being that initial state of the c.a extends on more than one cell, which is not in accordance with our general definition. This shifting will no more be made in next figures.

An input a_t entered at time t is reflected by S (with a possible delay 1). As it comes back, it deposits on cell 0 the word $h(a_t)$. Output function outputs at each time-step the first letter of the word deposited (no letter if nothing, or nothing more, is deposited), which letter is retrieved from the word. Input a_{t+1} arrives on cell 0 k time-steps later, and as we have choosed k larger or at least equal to $|h(a_t|$, cell 0 is emptied before word $h(a_{t+1})$ is loaded.

The output word is eventually the image of the input word $a_0a_1 \dots a_{n-1}$, with possible interruptions. These and the computation time will be least if k is the smallest possible, that is $k = max(|h(a)|)$. Interruptions in the inputs would also be possible.

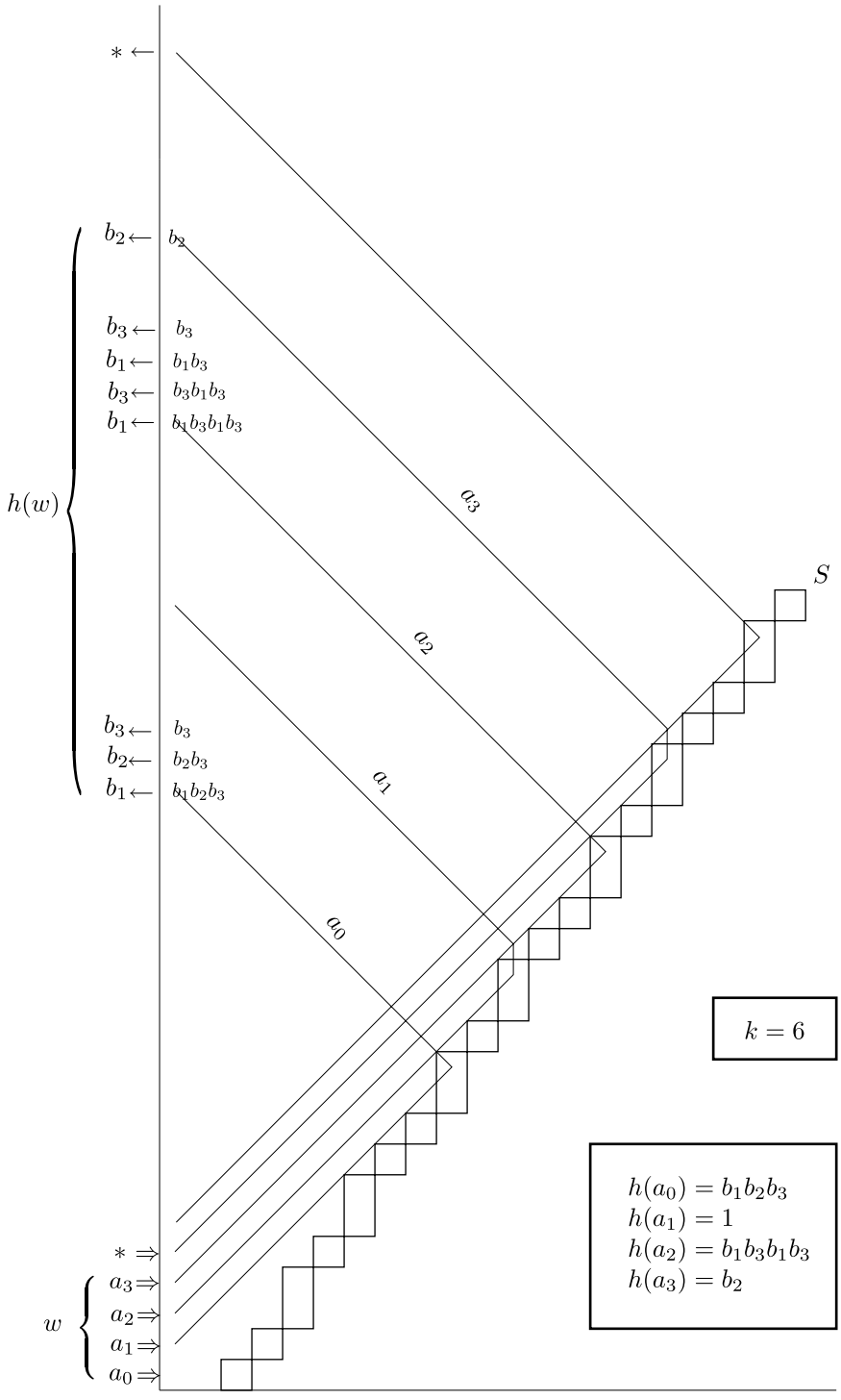

$* \leftarrow$
$b_2 \leftarrow$ b_2
$b_3 \leftarrow$ b_3
$b_1 \leftarrow$ b_1b_3
$b_3 \leftarrow$ $b_3b_1b_3$
$b_1 \leftarrow$ $b_1b_3b_1b_3$
$h(w)$
a_3
S
a_2
$b_3 \leftarrow$ b_3
$b_2 \leftarrow$ b_2b_3
$b_1 \leftarrow$ $b_1b_2b_3$
a_1
a_0
$k = 6$
$h(a_0) = b_1b_2b_3$
$h(a_1) = 1$
$h(a_2) = b_1b_3b_1b_3$
$h(a_3) = b_2$
$* \Rightarrow$
$a_3 \Rightarrow$
$a_2 \Rightarrow$
$a_1 \Rightarrow$
$a_0 \Rightarrow$
w

Fig.7

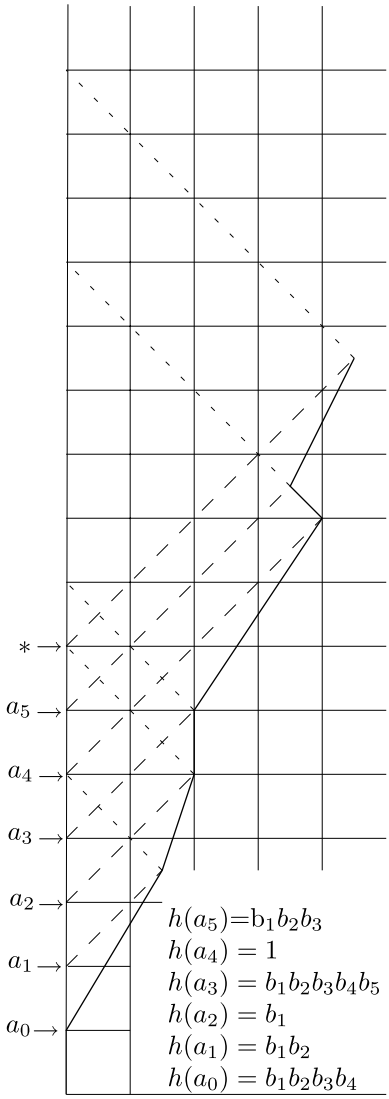


Fig.8a

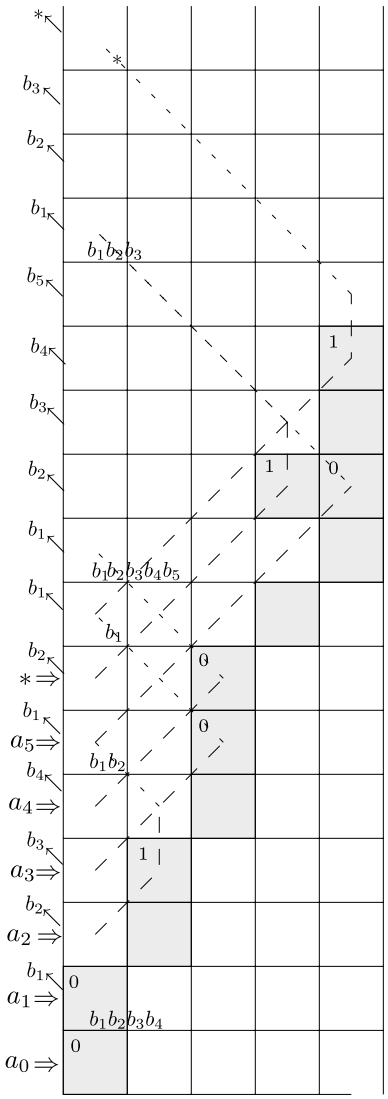


Fig.8b

6.3.2 A more sophisticated model

with which we get $h(w)$ without interruptions.
We should now like output

- to start at time 1
- and go on with no interruption

Then the morphism would be computed in real time. Having this in view, we replace the straightline signal S by a Z signal formed of segments having slopes precisely adjusted to the foreseen length of the word to output : as a_t arrives on Z it dictates to this signal its new slope $p_t = (h_t+1)/(h_t-1)$, where $h_t = |h(a_t|$. Signal Z will start from site $(0,0)$ with infinite slope (and no delay).

If geometrically there is no difficulty whatever (see Figure 8a), returning to discreet constructions needs attention.

Let us first remind how signals corresponding to the different values of k are set up : they are formed with states that never reflect anything because they cannot receive any input arriving at speed 1, and others that reflect with delay $d = 0$ *or* 1. For odd k, all delays are 1, but, as only time intervals matter, we shall declare all delays to be 0 if we please (Figure 9).

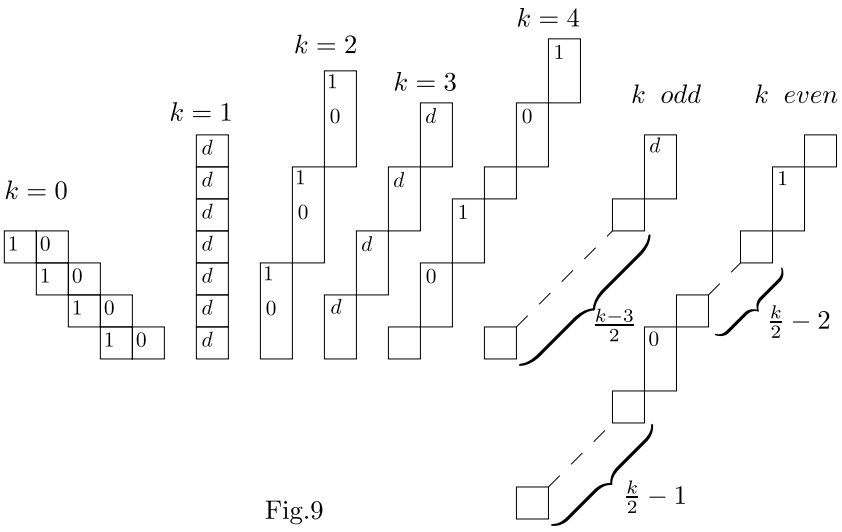


Fig.9

The delicate task is that of connecting segments having different slopes. Rule will be the following : if a_t arrives on Z on a state of delay d, the present meeting site must correspond to a d-delay site for next segment of Z. This next segment will end when next input arrives. All possible cases are represented in Figure 10, even the tricky case when $h(a_t)$ is the empty word, $h_t = 0$ so $p_t = -1$: in case a_t arrives on a null-delay state of signal Z, transition must set up next state of Z at time preceding, when a_t whose image is empty arrives left of Z.

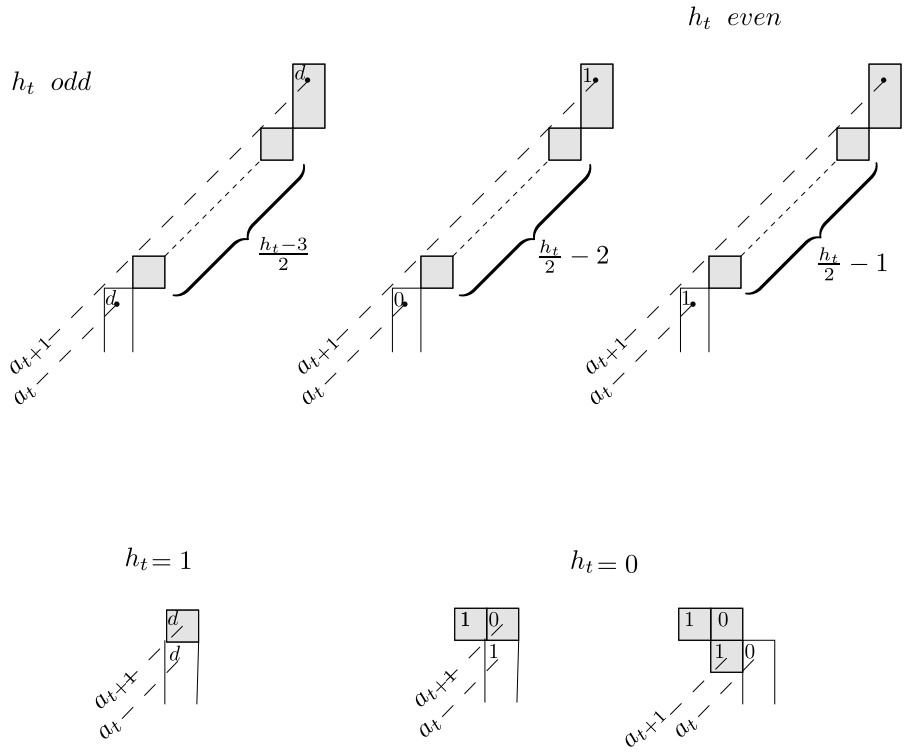


Fig.10

All these rules have been applied to draw the s.t.d of Figure 8b. Perhaps Figure 8c, which is a little film of events will help understand how things work out.

Let it be clear that the rules we have just mentioned are only rules in the common sense of the word, they are not c.a transition rules. In fact, we are describing new c.a's and how they work in terms of signals : we hope it is sufficiently clear by now that all this could be translated into transition rules. But what a very dull task that would be !

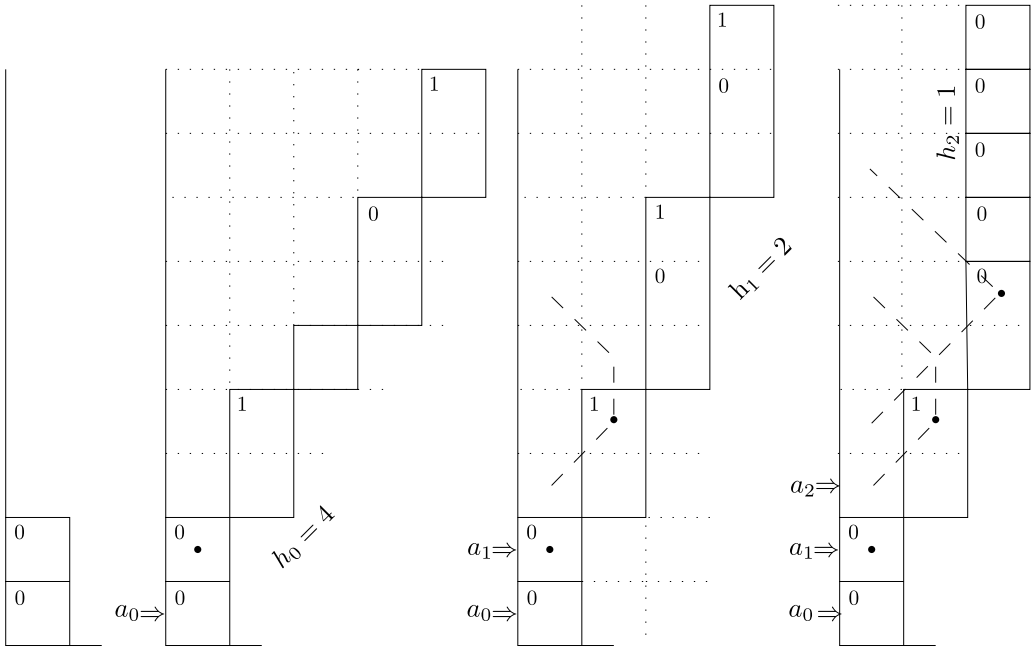


Fig.8c

Beware now that we cannot go left of cell 0 ! (If we don't want to use more cells). This implies a condition on the morphism and the input word that we find quite easily on the geometric Figure (reminded in Figure 11, with an illustration by example of Figure 8) :

$$\frac{h_0 - 1}{2} + \frac{h_1 - 1}{2} + \ldots + \frac{h_i - 1}{2} \geq 0 \quad i = 1, \ldots, n$$

That is to say : for any prefix u of $w = a_0 a_1 \ldots a_{n-1}$

$$|h(u)| \geq |u|$$

or else : word w must not have too many accumulated empty-image letters.

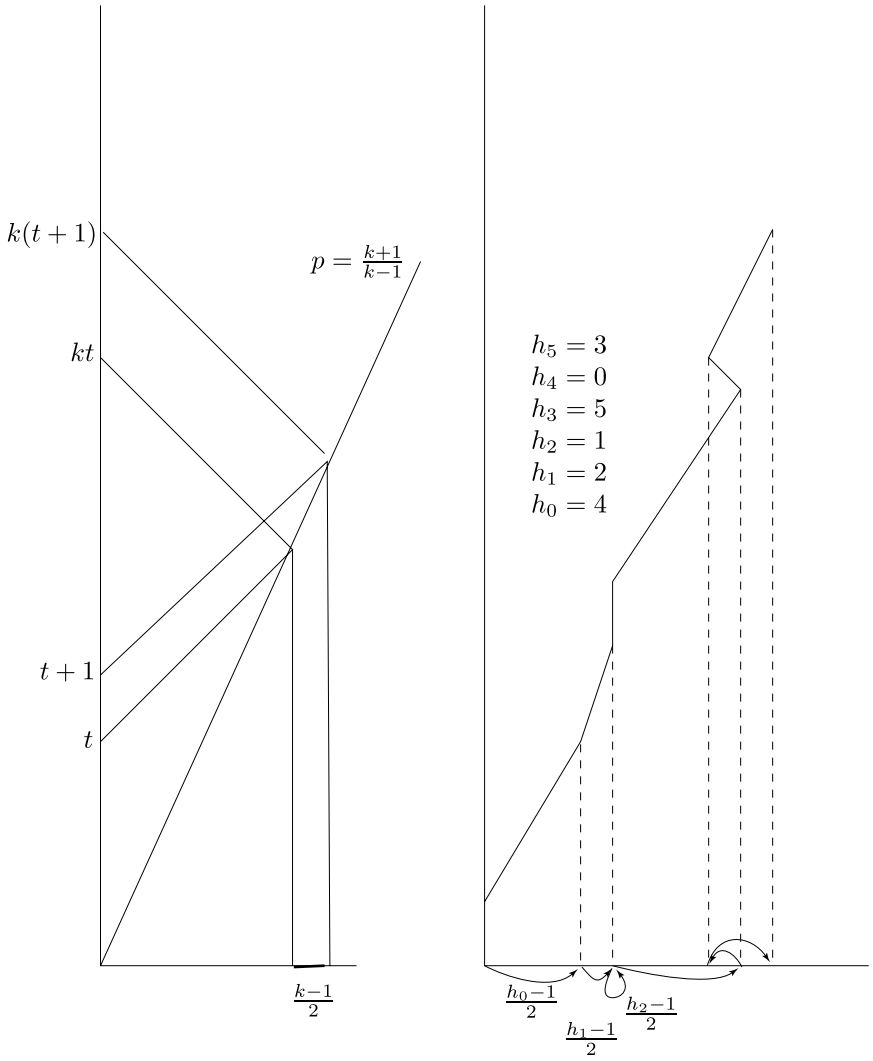


Fig.11

6.3.3 An application

We are now able, from a c.a recognizing any language L, to build a c.a recognizing language $h^{-1}(L)$: it will simply be the product of the c.a calculating h and the recognizer for L, the former producing little by little the image of the input word needed by the latter. Thus we have the

Theorem 6.3.1 *If a language L in A^* is c.a-recognizable and $h : A^* \longmapsto B^*$ is a word morphism, then $h^{-1}(L)$ is c.a-recognizable.*

Moreover, we can effectively build the recognizing c.a.

6.4 A.c computing the semi-infinite word defined by a morphism

Here we have a nice use of preceding c.a.

Let us first remind the subject : if some morphism $h : A^* \longrightarrow A^*$ and a word $w \in A^*$ are such that w is a prefix of $h(w)$,

let us write :

$$h(w) = wu$$

then :

$$h^2(w) = wuh(u)$$

$$\ldots\ldots$$

$$h^n(w) = wuh(u)h^2(u)\ldots h^{n-1}(u)$$

so the $h^n(w)$ are prefix each of the next, and may be obtained by concatenating to w the $h^i(u)$'s one after the other.

If for all $n \geq 0$, $h^n(u)$ is not the empty word 1, (the hypothesis most frequently encountered is $u \neq 1$ and $\forall a \in A,\ h(a) \neq 1$), the $h^n(w)$ have strictly increasing lengthes and define a semi-infinite word which is their common extension, denoted $h^\infty(w)$. The case being, we say that h and w form a *prefixal system*. And we point out that there exists then a smaller word ω that generates the semi-infinite word.

- example: $h : \{a, b, c\}^* \longrightarrow \{a, b, c\}^*$

  $$h(a) = ab \quad h(b) = 1 \quad h(c) = cac$$

  and $w = abcaca$ generate a word $h^\infty(w)$. a does not generate this word, nor ab, the smaller word to do it is $\omega = abc$.

- two famous examples are the Morse word which contains no cube, defined by

  $$h : \{a, b\}^* \longrightarrow \{a, b\}^* \quad h(a) = ab \quad h(b) = ba \qquad \omega = a$$

  and the Thue word which contains no square, defined by

  $$h : \{a, b, c\}^* \longrightarrow \{a, b, c\}^* \quad h(a) = b \quad h(b) = ca \quad h(c) = cba \qquad \omega = c$$

For our c.a, which computes morphism h, to produce $h^\infty(\omega)$ from input ω, it suffices that, once ω is used out, it uses at each time-step the first letter of the word deposited in cell 0, instead of the inputs.

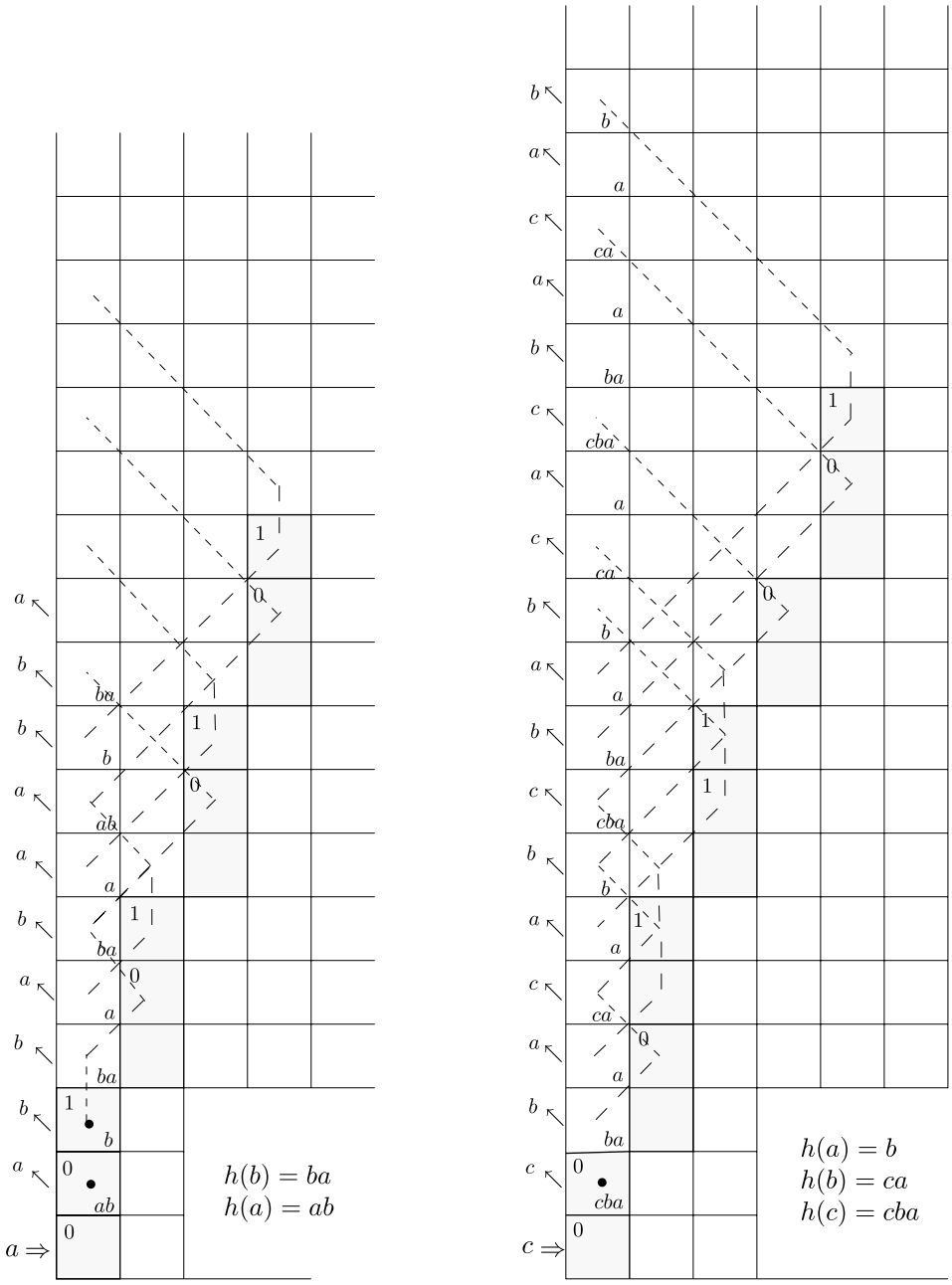


Fig.12

Figure 12 gives the s.t.d's for the words of Morse and Thue. So as not to

get mixed up at the beginning, reflections being achieved even before they have started, the two actions impulsed by each input must be carefully executed one after the other :

- input arriving on Z determines next segment of the signal (with its delay)

- when its reflection comes back on cell 0, it deposits word $h(a)$ which will then be re-used letter by letter.

Should we wish the c.a to run without any input, we could memorize the initial inputs in the initial state.

6.5 A special category of frequency signals

We refer here to Véronique Terrier's thesis [54]. Let us begin with some vocabulary :

If f is a strictly increasing mapping of $\mathbb{N}$ into $\mathbb{N}$, we shall call *frequency signal* for f, any signal that marks cell 0 at times $f(n)$, $n \in \mathbb{N}$. Such a signal, or rather the c.a that produces it, thus computes the successive values of f.

We shall call *growth function* of the prefixal system (h,w) function defined by

$$f(n) = |h^n(w)|.$$

Let us return to c.a producing word $h^\infty(w) = h^\infty(\omega)$: it never ceases calculating the image by morphism h of the word it produces, with no interruptions.

As we have seen, for any word w longer than ω and a prefix of $h^\infty(\omega)$, computing of $h(w)$ is achieved when there remains in cell 0 only the last letter of the image $h(x_{n-1})$ of the last letter x_{n-1} of w.

Let us then consider some signal T starting from cell 0 at time $|w|$ at speed 1, reflecting on Z like the inputs (i.e with the same delays) and on cell 0 with a delay equal to the number of letters deposited minus one : it leaves at time $|h(w)|$, and will continue criss-crossing between Z and cell 0, leaving the latter at times $|h^n(w)|$. It is a growth signal for prefixal system (h,w), provided it marks cell 0 only when leaving, that is when only one letter is left in the cell.
Let us mention a few examples of growth functions for prefixal systems :

- exponential functions $f(n) = k^n$ obtained with

$$A = \{a\} \quad h(a) = a^k \qquad \omega = a$$

- Fibonacci function (Figure 13) obtained with

$$A = \{a, b\} \quad h(a) = ab \;\; h(b) = a \qquad \omega = a$$

let us remind that this function is defined by : $f(1) = 2$, $f(2) = 3$, $\ldots$, and the induction formula $f(n+2) = f(n) + f(n+1)$

- the square function $f(n) = n^2$ (Figure 14) obtained with

$$A = \{a, b, c\} \quad h(a) = abcc \;\; h(b) = bcc \;\; h(c) = c \qquad \omega = a.$$

Frequency signals of these functions are thus c.a-computable.

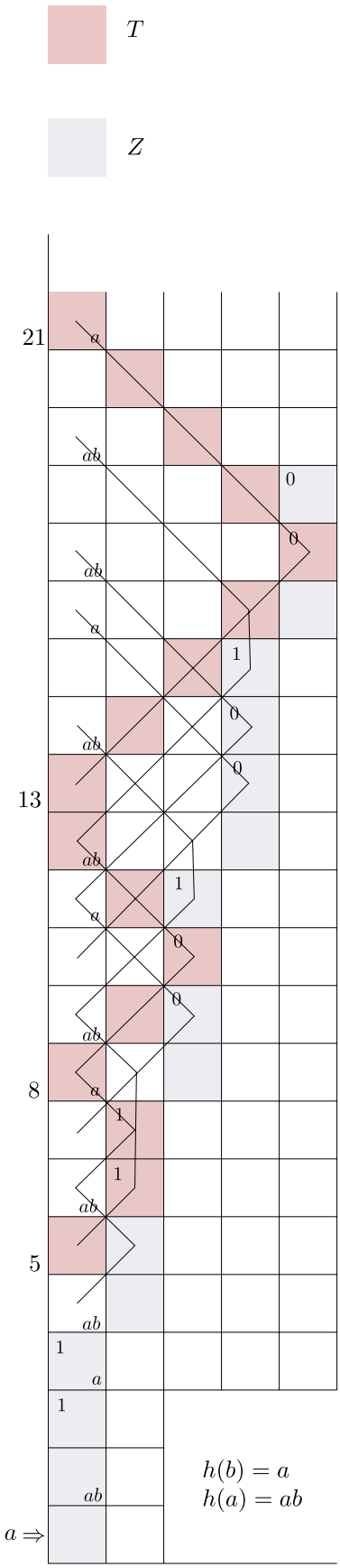


Fig.13

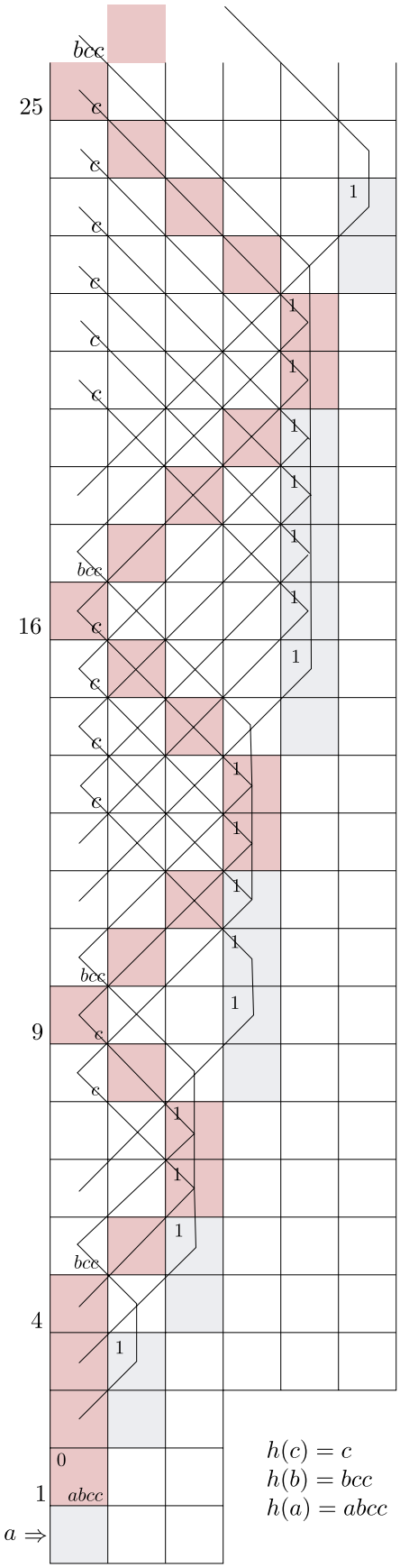


Fig.14

6.6 Slowing down with parallel input

whatever the initial state (quiescent or impulse), locking the c.a with a counter $k-1$ time-steps out of k implies no change in the input (see Figure 6 with modified input). If ever for some reason we should want the counter to start at time 1, we would have to retain the inputs in each cell for one time-step, nothing more.

For a synchronizer, which has no input, there is no problem at all. In all these cases we have

$$T' = kT \quad \text{or else} \quad T' = kT - (k-1)$$

6.7 Slowing down by a constant

To realize a slowing down of a recognizer or synchronizer of h time-steps, it suffices that only one time-step be replaced by $k = h+1$, and this we can manage with a counter, at the beginning or at the end. Only at the end for a sequential recognizer : every halting state will be changed in $k+1$ states, only the last one being a halting state.

$$T' = T + h$$

This operation is not very exciting, but it will be useful for adjusting times.

Chapter 7

Speeding up

7.1 Weak speeding up

To do the same thing in less time we must do in one time unit what was done in more than one. As we consider the s.t.d, two ideas may come to our mind :

- grouping time-units by k, and then new states of a cell are k-tuples of states

- selecting one time-step out of k

A glimpse on figures (Figure 1) reveals that to know the state of a cell, in both cases, we must know the states at preceding time of the cell itself and k neighbours on either side. As we do not want to increase the scope, that is we want neighbourhoods to be of one cell at the left and one cell at the right, we are lead to group cells by k (Figure 2). This means that we consider new cells which are segments formed of k original cells. So the number of new states will be, in the first case $|Q|^{k^2}$ and in the second $|Q|^k$.

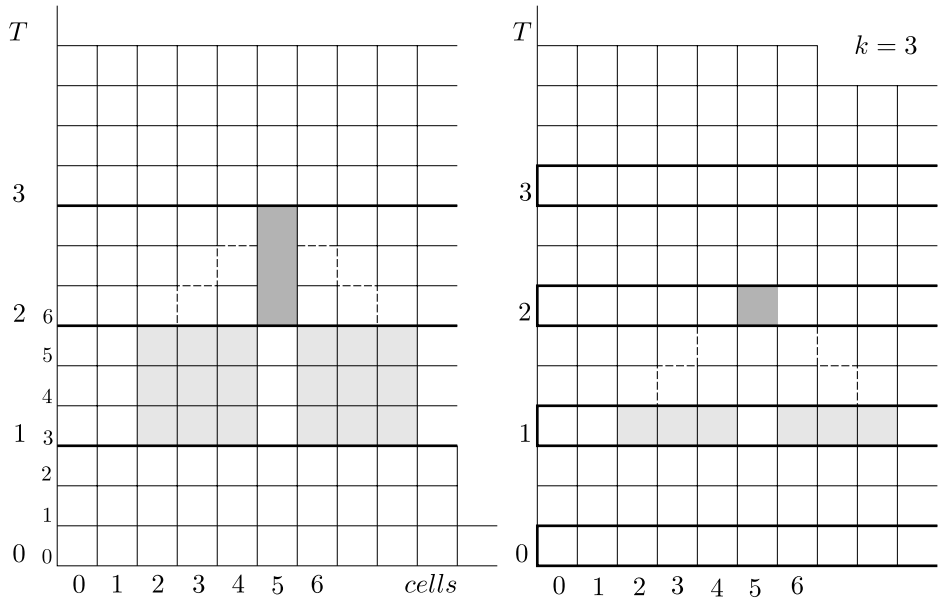


Fig. 1

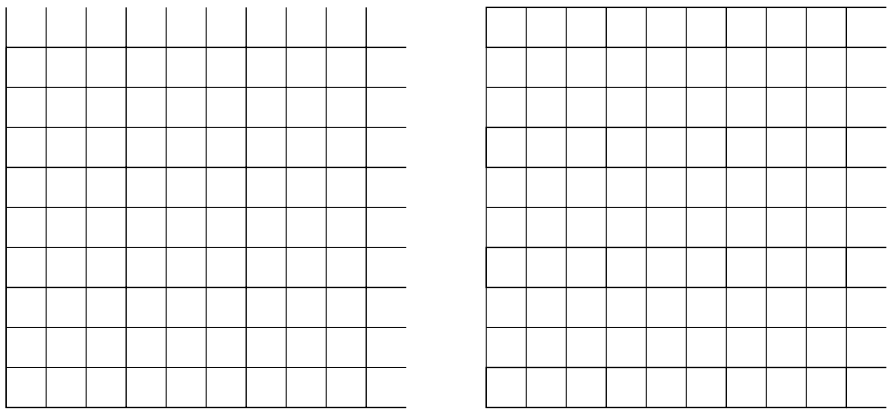


Fig. 2

We have the general idea, but we must fix all the details, what we shall do for the two big families of c.a's that we have encountered, the synchronizers on the one hand, and the recognizers or computers on the other hand.

7.1.1 For recognizers

(Computers have already disappeared from the title as it will not be long before we exclude them.)

Initial state. With the first solution, initial state is a block of k^2 states, which immediately appears to have a disqualifying defect : with different inputs, be they sequential or parallel, the initial state would not be the same (see Figure 3) !

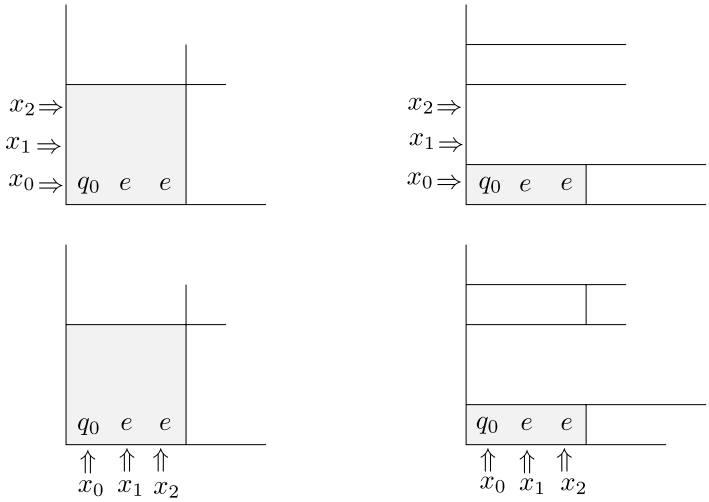


Fig. 3

This phenomenon, which by the way would not be limited to the initial state, leads us to abandon the first possibility. In the second, initial state is the k-tuple

$$(q_0, e, \ldots, e)$$

Inputs. We are lead to group them by k :

- with a sequential input this is because we select one time-unit out of k
- with a parallel input it is because we group the cells.

In the case of a sequential input - which is the normal case - input at time 0 for the fast c.a will be the k-tuple of inputs from time 0 to time $k-1$, and input at any time t the k-tuple of inputs from time kt to time $kt+k-1$. So if the input word was

$$x_0 x_1 \ldots\ldots x_{n-2} *$$

the new input word will be

$$(x_0 \dots\dots x_k) \dots\dots\dots (\dots x_{n-2} * - - \dots -)$$

(where - is nothing) of length $\lceil n/k \rceil$. This new input word, which is well determined and unique, will be called the k-grouping of the old input word.

And there we are precisely touching the weak point of the c.a we try to conceive : we must previously change its input word.

The halting states. A halting state may appear at a time which is a multiple of k, and it is clear that any k-tuple whose first element is a halting state q_a will be a halting state for the fast c.a.

But if the halting state q_a appears at a time which is not a multiple of k, which shall we consider a halting state, the preceding or the following state of the fast c.a ?

- it cannot be the preceding state, because the arrival of q_a may depend on further inputs (Figure 4).

- to be honest, the next state is not defined. So we shall define new states and complete the fast transition : for each halting state q_a we define a corresponding state Q_a, and agree that the fast transition leads to Q_a if q_a appears on cell 0 at one of the $k-1$ preceding time-steps (which somehow are the history of the state).

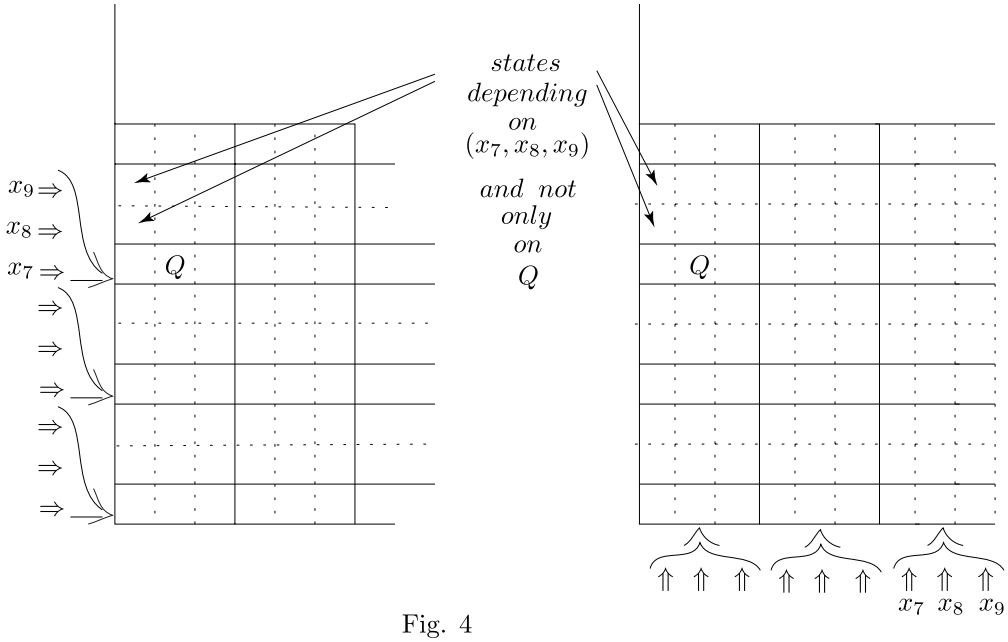


Fig. 4

Now then, we do not need to distinguish Q_a and the different k-tuples with first component q_a, so we shall confuse them all by deciding that the fast c.a

arrives in state Q_a if the original c.a has just arrived in state q_a, or has arrived in this state at one of the $k-1$ preceding time-steps (Figure 5). Naturally, Q_a will be accepting or rejecting as q_a.

If the halting time was T for the original c.a, for the fast one it will be $\lceil T/k \rceil$.

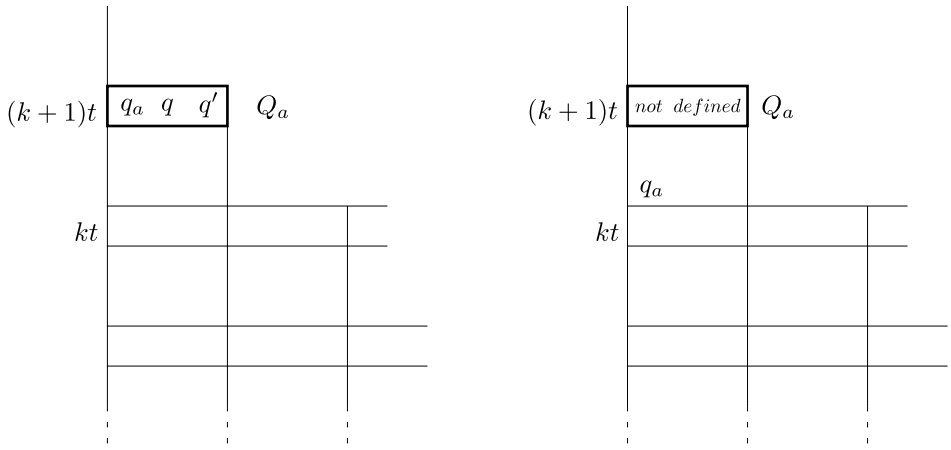


Fig. 5

Outputs. We could be tempted to group them like the inputs. But if the output should depend on the state, only on the state, of cell 0, we may be embarrassed with the inputs, because a same state receiving different inputs could produce different outputs (see Figure 4). So we shall simply lay computer c.a's aside in this chapter.

7.1.2 For synchronizers

They have no outputs and no inputs, so none of the corresponding problems. They work with a finite number of cells, starting with an initial state which is not necessarily reduced to an impulse (as examples : a general at either end, a general in the middle of the line)

States. We have rejected block states for recognizers, so we do not want to introduce them anew here, first because they are particularly cumbersome and numerous, then for the sake of coherence in structures, and finally because we shall often have to combine recognizers and synchronizers.

So states are the k-tuples of states, on k-tuples of cells. But a new problem arises here : how must the cells be grouped ? from the left ? from the right ? we shall reserve ourselves all possibilities by adding to the k-tuples all the incomplete h-tuples ($h \leq k$) containing a left or right border state, both denoted β

$$(\beta, q_1, \ldots, q_i) \qquad (q_1, \ldots, q_j, \beta).$$

For a better presentation we shall replace these β's by several β's so as to restore k-tuples

$$(\beta, \ldots, \beta, q_1, \ldots, q_i) \qquad (q_1, \ldots, q_j, \beta, \ldots, \beta).$$

So states will be elements of $[Q \cup \{\beta\}]^k$. And in applications we shall be free to group cells as we please.

The fire state. We shall define it in the same way as we have defined the halting states for recognizers. We shall confuse in a single state : all k-tuples containing the fire state and β's, and a new state for the fast synchronizer, to which transition leads if the fire state has appeared at one of the $k-1$ preceding time-steps (Figure 6), (in the history).

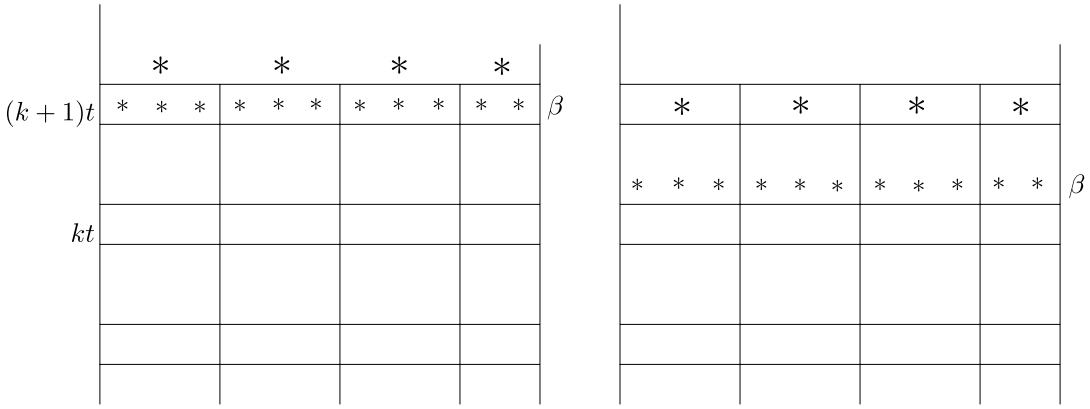


Fig. 6

If n cells were synchronized in time $T(n)$ by the original c.a, the new synchronization time is $\lceil T(n)/k \rceil$. But what exactly have we synchronized ? $\lceil n/k \rceil$ or $\lceil n/k \rceil + 1$ of our new cells. So we must not deceive ourselves, in k times less time we have synchronized k times less cells !

Speeding up seems thus senseless for synchronizers. Let us be patient, it will show useful when we know how to manage it properly, after we have gained experience with the parallel recognizers (cf section 7.4).

7.2 Strong speeding up for recognizers

We want to speed up by a factor $k \geq 2$: this can only be thought of if recognizing time for an input word of length n is $T(n) > n + 1$, greater than the real time.

Our aim here is to use the preceding weak accelerated c.a after having changed the original input into the k-grouped input.

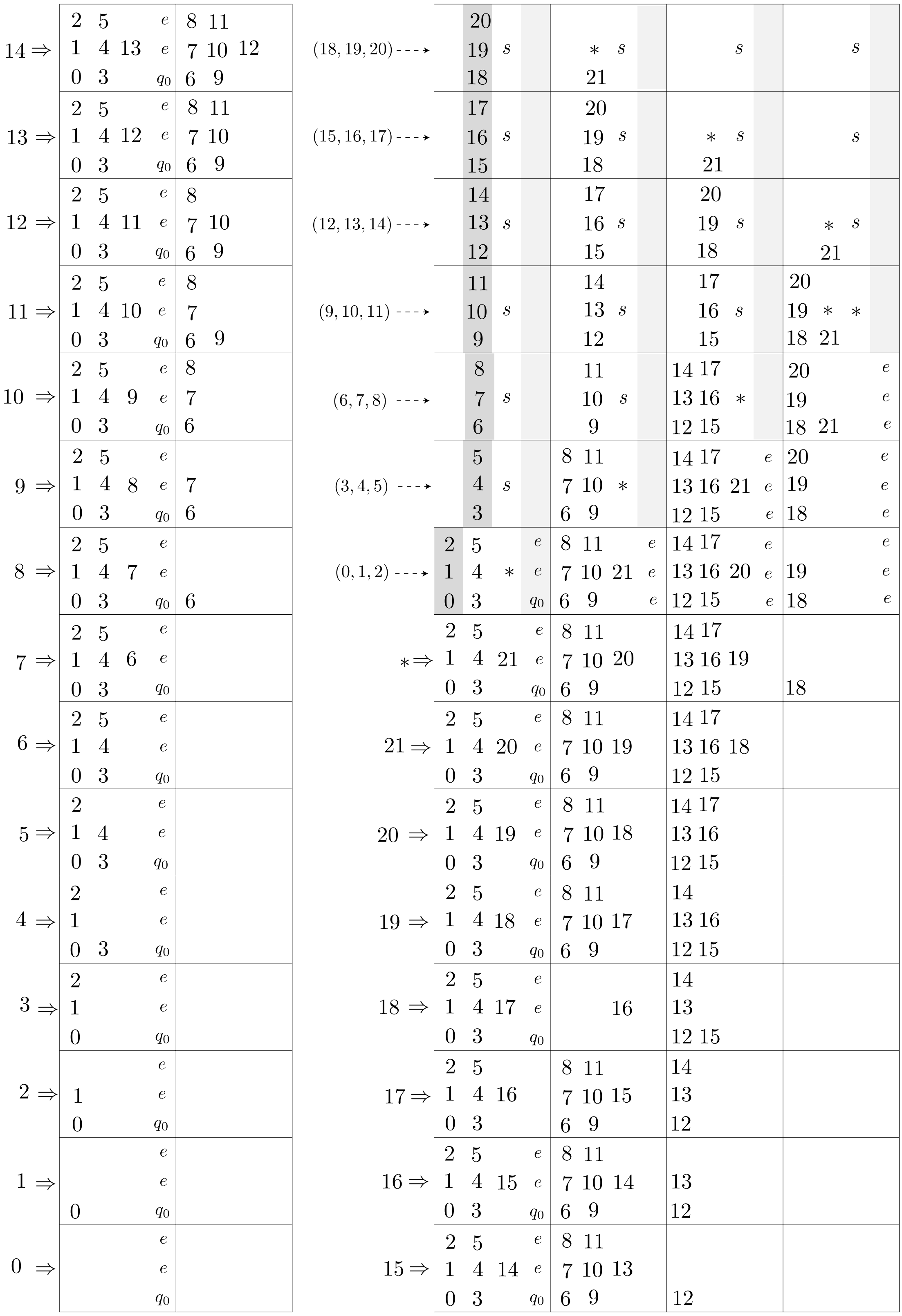

14 ⇒
13 ⇒
12 ⇒
11 ⇒
10 ⇒
9 ⇒
8 ⇒
7 ⇒
6 ⇒
5 ⇒
4 ⇒
3 ⇒
2 ⇒
1 ⇒
0 ⇒
(18, 19, 20)
(15, 16, 17)
(12, 13, 14)
(9, 10, 11)
(6, 7, 8)
(3, 4, 5)
(0, 1, 2)
∗ ⇒
21 ⇒
20 ⇒
19 ⇒
18 ⇒
17 ⇒
16 ⇒
15 ⇒

Fig. 7

We shall do this very simply, by piling up inputs in stacks of k, as they arrive. As shown on Figure 7, where only the indexes of the inputs have been written down), two stacks in each cell will be enough. If input enters a cell where the two stacks are already full, it goes in a transit place to travel rightwards till it finds the first free place.

As soon as the end marker $*$ has entered cell 0, at time $n+1$, then grouped inputs can be used : like the other inputs, $*$ travels right till it finds its place, but it also definitely marks cells, with s in transit place. Now, when $*$ (or s) is in cell 0, left (or right) stack is used as input, and when $*$ (or s) is in any other cell, left (or right) stack travels left.

One can easily check here that one stack in each cell would not suffice to get the grouped inputs at the desired rythm.

Initial state of our new c.a is $Q_0 = (q_0, e, \dots, e)$, it is stored in another part of the new states (at the right in Figure 7). It is inhibited as long as the end marker has not entered cell 0, so is awakened at time $n+1$. From there on, the weak acelerated c.a works, in the right part of the states.

The new recognizing time will then be

$$n + 1 + \left\lceil \frac{T(n)}{k} \right\rceil .$$

As a consequence, any recognizing taking a linear time $T(n) = an$ (a an integer) can be speeded up into a recognizing in time $2n+1$, by choosing speeding factor a. Even better, with a speeding factor $\lceil a/\varepsilon \rceil$, the new recognizing time could be

$$\leq n + 1 + \lceil \varepsilon n \rceil = 1 + \lceil n(1+\varepsilon) \rceil.$$

We underline here that speeding up sequential recognizers needs no auxiliary synchronizing, it is ideally simple.

7.3 Strong speeding up for parallel recognizers

Method of this section is due to N.Reimen and J.Mazoyer [45, 46].

Here we must proceed to a horizontal k-grouping of the input word

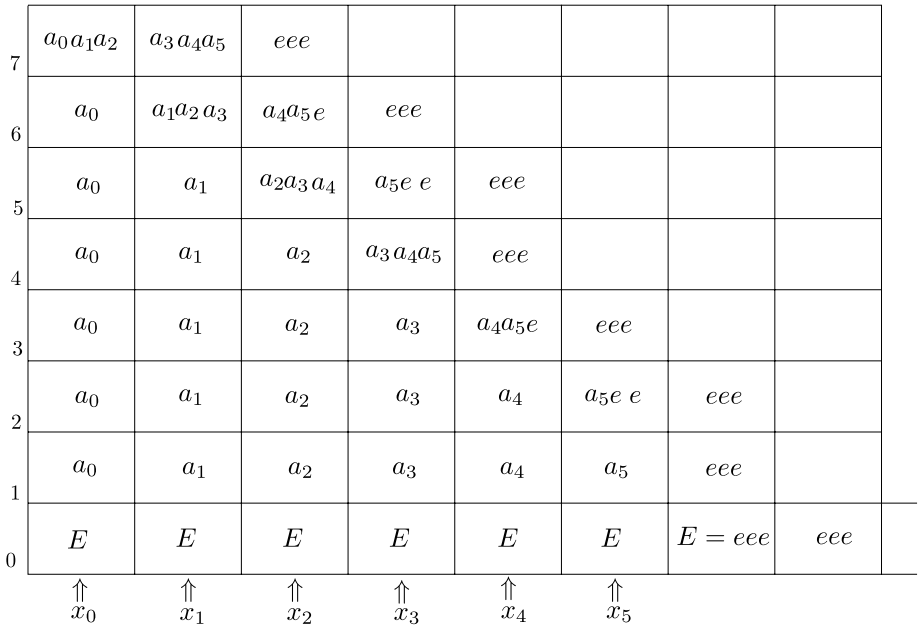
$$(x_0, x_1, \dots\dots, x_{n-1}), \quad \text{of length n,}$$

(x_{n-1} might be an end marker if we wish), or rather of the states

$$(a_0, a_1, \dots\dots, a_{n-1})$$

resulting at time 1 from entering of the inputs. Quiescent state of our new c.a will be E, which, in view of further use, we detail in form $(e, e, \dots, e)$. Inputs entering on state E begin by settling the same states a_i as in the original c.a. Then the grouping will proceed from the rightside by the following mechanism :

any grouped state pushes all its elements except the last one into the left neighbour, the process being started by $E = (e, e, \ldots, e)$ at time 1 (figure 8a, where $k = 3$).



7	$a_0a_1a_2$	$a_3a_4a_5$	eee					
6	a_0	$a_1a_2a_3$	a_4a_5e	eee				
5	a_0	a_1	$a_2a_3a_4$	$a_5e\ e$	eee			
4	a_0	a_1	a_2	$a_3a_4a_5$	eee			
3	a_0	a_1	a_2	a_3	a_4a_5e	eee		
2	a_0	a_1	a_2	a_3	a_4	$a_5e\ e$	eee	
1	a_0	a_1	a_2	a_3	a_4	a_5	eee	
0	E	E	E	E	E	E	$E = eee$	eee
	$\Uparrow$ x_0	$\Uparrow$ x_1	$\Uparrow$ x_2	$\Uparrow$ x_3	$\Uparrow$ x_4	$\Uparrow$ x_5		

Fig. 8 a

7	$<2,7>$ $<1,7>$ $<0,7>$	$<5,7>$ $<4,7>$ $<3,7>$	$<8,7>$ $<7,7>$ $<6,7>$	$<11,7>$ $<10,7>$ $<9,7>$	eee	eee	eee	eee
6	$<0,6>$	$<3,6>$ $<2,6>$ $<1,6>$	$<6,6>$ $<5,6>$ $<4,6>$	$<9,6>$ $<8,6>$ $<7,6>$	e e $<10,6>$	eee	eee	eee
5	$<0,5>$	$<1,5>$	$<4,5>$ $<3,5>$ $<2,5>$	$<7,5>$ $<6,5>$ $<5,5>$	e $<9,5>$ $<8,5>$	eee	eee	eee
4	$<0,4>$	$<1,4>$	$<2,4>$	$<5,4>$ $<4,4>$ $<3,4>$	$<8,4>$ $<7,4>$ $<6,4>$	eee	eee	eee
3	$<0,3>$	$<1,3>$	$<2,3>$	$<3,3>$	$<6,3>$ $<5,3>$ $<4,3>$	e e $<7,3>$	eee	eee
2	$<0,2>$	$<1,2>$	$<2,2>$	$<3,2>$	$<4,2>$	e $<6,2>$ $<5,2>$	eee	eee
1	$<0,1>$	$<1,1>$	$<2,1>$	$<3,1>$	$<4,1>$	$<5,1>$	eee	eee
0	$<0,0>$ $=$ q_0	E	E	E	E	E	E	E
	$\Uparrow$ x_0	$\Uparrow$ x_1	$\Uparrow$ x_2	$\Uparrow$ x_3	$\Uparrow$ x_4	$\Uparrow$ x_5		

Fig. 8 b

The k-grouping will be achieved at time $n+1$. The process must then be stopped, what only a synchronizer can do, by a fire state at time $n+1$. As we already know, a one general synchronizer cannot synchronize n cells (which is certainly more than needed) in time $n+1$, so we shall use two generals, one at the left and one at the right. If this synchronizer is minimal time, it does the job in $n-1$ time units, so it must start at time 2. To this purpose we decide that our fast c.a

- produces generals through transition with triples

$$(\beta, a_i, a_j) \qquad and \qquad (a_i, a_j, E)$$

- produces a special state playing the part of a border for the synchronizer from triple (a_j, E, E).

The grouping process and synchronization develop simultaneously but independently in two separate parts of the new states : to this point, our c.a is the product of the grouping c.a and the synchronizer.

Now then we decide that the fire, not only stops the grouping, but starts the weak speeded up c.a, by activating initial state which was stored from the beginning, in a third component, in cell 0.

The new recognizing time is then

$$n+1+\left\lceil \frac{T(n)}{k} \right\rceil .$$

But we can still do a little better without much strain. Indeed, nothing prevents us from operating transition while grouping : we just let the c.a, at each time-step, do its transitions (possibly grouped), before the states are pushed. It is recommended to get convinced with the help of figure 8b, that all states of any site are obtained from the three sites of the neighbourhood at preceding time. Here fire state will stop the grouping and start the fast transition, from a state which is no more the initial state, but state at time $n+1$ of the original c.a.

The recognizing time is then

$$n+1+\left\lceil \frac{T(n)-(n+1)}{k} \right\rceil .$$

It seems advisable here to denote $D(n)$ the delay between minimal recognizing time (which is $n+1$) and the recognition time $T(n)$. The new recognizing time then writes

$$n+1+\left\lceil \frac{D(n)}{k} \right\rceil .$$

7.4 Speeding up synchronizers

The following solution is due to O.Heen [23, 24].

If we try to apply the k-grouping process of preceding section to a synchronizer $\mathcal{S}_0$ (original synchronizer, having synchronizing time $T(n)$), it could be started from the right by some state $(q, \beta, \dots, \beta)$, element β, reminding of the border, taking the place of the quiescent state in preceding section.

If a grouped state of form $(q, \beta, \dots, \beta)$ is set in cell n at time 0, grouping will be achieved at time $n-1$, on the $\lceil n/k \rceil$ left cells, remaining cells being all in state $(\beta, \beta, \dots, \beta) = B$.

The weak accelerated c.a which takes the relay can then bring, in time

$$n - 1 + \left\lceil \frac{T(n)}{k} \right\rceil \quad (\text{or} \quad n - 1 + \left\lceil \frac{T(n) - (n-1)}{k} \right\rceil)$$

the $\lceil n/k \rceil$ left cells in fire state, cells on the right staying in state B.

But why not do a same grouping from the left to the right , which would permit us to bring to fire the $\lceil n/k \rceil$ right cells ? In case $k = 2$, the conjunction of the right and left synchronizations suffices to synchronize the entire line of cells. But if $k \geq 3$ the central cells remain in state B. So we have the idea of grouping from the left and the right towards a central region, and even towards several central regions covering the entire line. This needs some work, which we do in next paragraphs.

Let us notice that we have not specified which family of synchronizers we want to speed up, the one with one general, or the one with two, or the one where the general is somewhere on the line. For this reason we shall denote the initial state of S_0 by $(q_1, \dots, q_n)$, a general and imprecise notation.

7.4.1 Conjugate signals of slopes $\frac{b}{a}$ and $-\frac{b}{b-a}$, $0 \leq a \leq b$

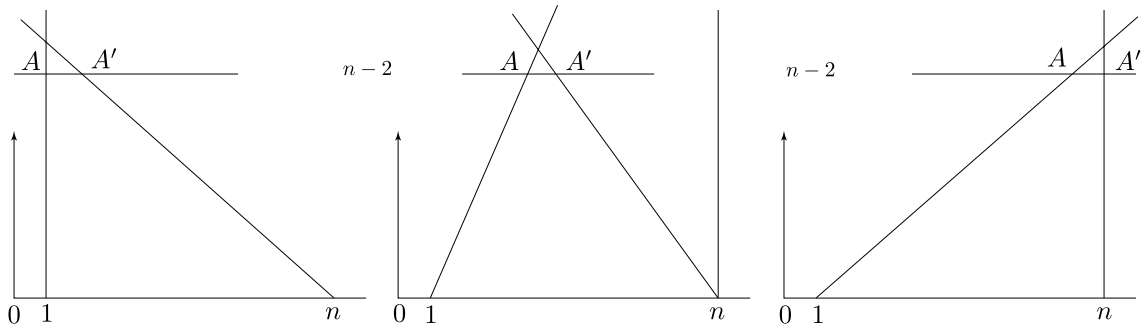


Fig. 9

In Figure 9 we show, in the space $\mathbb{R}^2$, and in cases $a = 0$ (left figure), $a = b$ (right figure), and general case (center figure)

- the straightline D from point $(1,0)$ having slope b/a
- the straightline D' from point $(n,0)$ having slope $-b/b-a$
- the points A, A' where they cross horizontal line $y=n-2$:

$$\text{abscissa}(A) = 1 + (n-2)\frac{a}{b} \qquad \text{abscissa}(A') = 2 + (n-2)\frac{a}{b}.$$

Now we want to come back on our study ground, that of s.t.d sites, that is $\mathbb{N}^2$. To approximate straightlines D and D' we shall replace them by the sites immediately on the right of their points of integer ordinates (Figure 10).

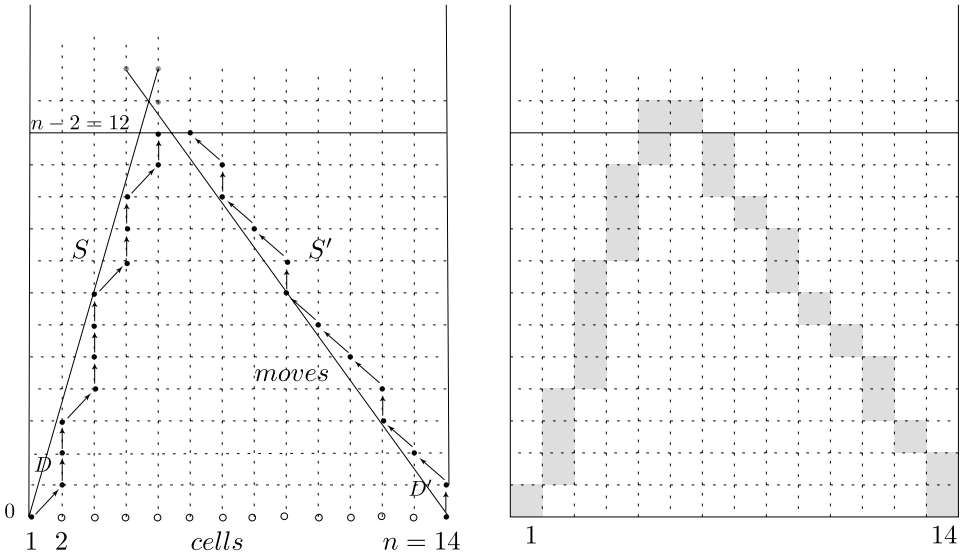


Fig. 10

We thus obtain the broken lines S and S' formed with points

$$(1 + \left\lceil \frac{a}{b}t \right\rceil, t) \quad \textit{and} \quad (\left\lfloor n - t + \frac{a}{b}t \right\rfloor, t).$$

The successive moves of S from time 0 are the periodic repetition of the sequence of b 0- or 1-moves (for $a \leq b$) that follow :

$$\left\lceil \frac{a}{b} \right\rceil, \left\lceil \frac{2a}{b} \right\rceil - \left\lceil \frac{a}{b} \right\rceil, \left\lceil \frac{3a}{b} \right\rceil - \left\lceil \frac{2a}{b} \right\rceil, \cdots, a - \left\lceil (b-1)\frac{a}{b} \right\rceil.$$

So S may be realized in a s.t.d by a signal formed by (at most) b states $s_1, s_2, \ldots, s_b$. S' in the same way, with states $s'_1, s'_2, \ldots, s'_b$. These associated signals, S and S', will be called *conjugate signals* of slope b/a and $-b/b-a$.

At time $t = n - 2$, signals S and S' issued from cells 1 and n are

$$S \text{ on cell } \lceil \frac{a}{b}(n-1) \rceil \text{ and } S' \text{ on cell } \lceil \frac{a}{b}(n-1) \rceil + 1$$

and $n - 2$ is the first time when S and S' are on neighbouring cells, S on one cell and S' on its right neighbour.

In the sequel we shall want this to occur at time $n-1$ and not at time $n-2$, so we shall complete S and S' each with a state that sets them up at time 0, s_0 in cell 1 and s'_0 in cell n.

7.4.2 k-grouping guided by a signal

On Figure 8 we see how grouping progresses from right to left along speed(-1)-signal as frontline.

To achieve an S-guided grouping, we only need each grouped cell to possess a periodical counter where the states of S succeed one another : depending on wether it contains a 1-move,-1-move or 0-move state, inputs will push rightwards, or leftwards, or stand still. Let us be very precise :
at time 0 we set up

- not the initial state of $\mathcal{S}_0$

$$(q_1, q_2, \ldots\ldots, q_n)$$

but the initial state "grouped with the border"

$$(\beta, \ldots, \beta, q_1)\ ,\ q_2, \ldots\ldots,\ (q_n, \beta, \ldots, \beta)$$

- and in cells 0 and n respectively states s_0 and s'_0 of S and S',(corresponding to 0-moves)

Figure 11 illustrates this on 5 cells in the two cases of conjugate signals of slopes $-\infty, -1$ and $7/2, -7/5$ of Figure 10.

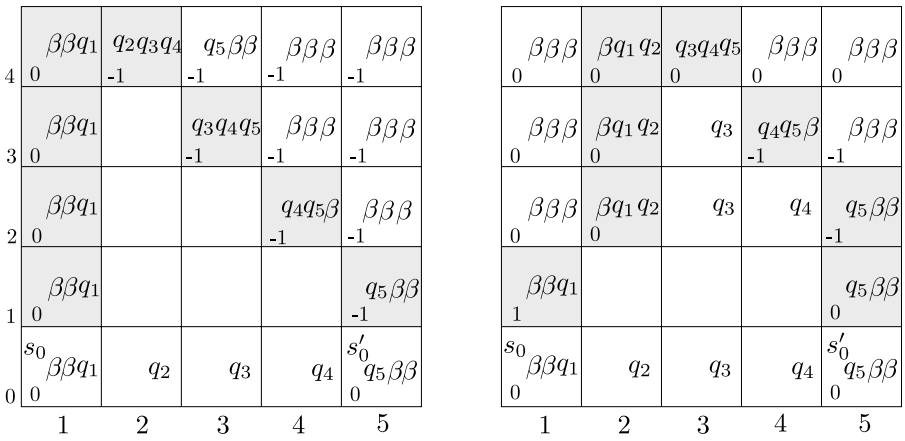


Fig. 11

Then, at time $n-1$ the left frontline of the grouping, S, has arrived in cell $1+\lceil(n-1)a/b\rceil$, while the right frontline has arrived on the right neighbour, so they have met, and there we must stop the grouping. As previously, nothing prevents us from computing at the same time, whatever the move.

At this time, which are the cells not in state $B=(\beta,\ldots,\beta)$? States other than β being k together in a grouped cell, the number of cells not in state B is

$$\left\lceil\frac{1+\lceil(n-1)\frac{a}{b}\rceil}{k}\right\rceil \text{ on the left, and } \left\lceil\frac{n-1-\lceil(n-1)\frac{a}{b}\rceil}{k}\right\rceil \text{ on the right.}$$

Index of the first cell which is not in state B is

$$g=2+\left\lceil(n-1)\frac{a}{b}\right\rceil-\left\lceil\frac{1+\lceil(n-1)\frac{a}{b}\rceil}{k}\right\rceil$$

and of the last one is

$$d=1+\left\lceil(n-1)\frac{a}{b}\right\rceil+\left\lceil\frac{n-1-\lceil(n-1)\frac{a}{b}\rceil}{k}\right\rceil.$$

7.4.3 Fast synchronization

We must now choose the couples of conjugate signals along which we shall group cells in central regions. As each couple will fill about n/k cells, we foresee about k couples, that we must distribute conveniently. We so choose $b=k$ and the $k+1$ values $a_0=0\ ,\ldots,\ a_i=i\ ,\ldots,\ a_k=k$. For $i=0$ and $i=k$, we recognize the groupings to the borders. Let us now verify that the sequences of cells not in state B produced by the S_i,S_i' couples cover the entire line. g_i and d_i being the first and last cells not in state B of the ith grouping, we are left to compare d_i and g_{i+1}. but

$$d_i=1+\left\lceil(n-1)\frac{i}{k}\right\rceil+\left\lceil\frac{n-1-\lceil(n-1)\frac{i}{k}\rceil}{k}\right\rceil$$

and

$$g_{i+1}=2+\left\lceil(n-1)\frac{i+1}{k}\right\rceil-\left\lceil\frac{1+\lceil(n-1)\frac{i+1}{k}\rceil}{k}\right\rceil.$$

Recall that

$$\left\lceil\frac{\lceil\frac{x}{p}\rceil}{q}\right\rceil=\lceil\frac{x}{pq}\rceil$$

(cf [31] exercise p.40) and

$$\lceil x+y\rceil\leq\lceil x\rceil+\lceil y\rceil\leq\lceil x+y\rceil+1$$

$$\lceil x \rceil - \lceil y \rceil \leq \lceil x - y \rceil$$

thus

$$\begin{aligned}
g_{i+1} - d_i &= 1 + \left\lceil (n-1)\frac{i+1}{k} \right\rceil - \left\lceil (n-1)\frac{i}{k} \right\rceil - \left\lceil \frac{1 + \lceil (n-1)\frac{i+1}{k} \rceil}{k} \right\rceil - \left\lceil \frac{n-1-\lceil (n-1)\frac{i}{k} \rceil}{k} \right\rceil \\
&\leq 1 + \left\lceil (n-1)\frac{1}{k} \right\rceil - \left\lceil \frac{\left\lceil \frac{k+(n-1)(i+1)}{k} \right\rceil}{k} \right\rceil - \left\lceil \frac{\left\lceil \frac{k(n-1)-(n-1)i}{k} \right\rceil}{k} \right\rceil \\
&\leq 1 + \left\lceil \frac{n-1}{k} \right\rceil - \left\lceil \frac{k+(n-1)(i+1)}{k^2} \right\rceil - \left\lceil \frac{k(n-1)-(n-1)i}{k^2} \right\rceil \\
&\leq 1 + \left\lceil \frac{n-1}{k} \right\rceil - \left\lceil \frac{1}{k} + (n-1)\frac{i+1}{k^2} \right\rceil - \left\lceil \frac{n-1}{k} \right\rceil + \left\lceil \frac{(n-1)i}{k^2} \right\rceil \\
&\leq 1 - \left\lceil (n-1)\frac{i}{k^2} - \frac{1}{k} - (n-1)\frac{i+1}{k^2} \right\rceil \\
&\leq 1 - \left\lceil -\frac{1}{k} - \frac{n-1}{k^2} \right\rceil \\
&= 1 - \left\lfloor \frac{1}{k} + \frac{n-1}{k^2} \right\rfloor \leq 1
\end{aligned}$$

so finally $g_{i+1} \leq d_i + 1$.

This guarantees that any cell will receive a grouped state from one of the $(k+1)$ k-groupings.

Each k-grouping must be stopped at time $n-1$ and switched on the weak accelerated c.a : a single 2-ends synchronizer, Σ_2, will switch the $k+1$ processes. This synchronizer must be in fire state at time (n-1) : we know since chapter 1 that only a 2-ends synchronizer can achieve this and that such a synchronizer does indeed exist. But no synchronizer can do better, and this is the reason why we have delayed all the systems of conjugate signals by one time-step !

The fast synchronizer will be the product of Σ_2 and the $k+1$ groupers (or computer-groupers) followed by the weak accelerated c.a's.

For each cell, one at least of the $k+1$ weak accelerated c.a's brings it to the fire state at time

$$n - 1 + \left\lceil \frac{T(n)}{k} \right\rceil \qquad (\text{or} \quad n - 1 + \left\lceil \frac{T(n) - (n-1)}{k} \right\rceil .$$

We shall confuse all the product states where at least one component is fire, and declare this is the FIRE of our new c.a.

In Figure 12 we have sketched a few instants of a few cells of the fast synchronizer. If K denotes the set of states of Σ_2, Q_i and Q'_i the states of S_i and S'_i, its states belong to

$$K \times \prod_{i=0}^{k} \{Q_i \cup Q'_i\} \times Q^k.$$

To be precise we must add an $\emptyset$ symbol to sets Σ_2, Q_i and Q'_i, because the places reserved for them in the states of the fast c.a may be empty.

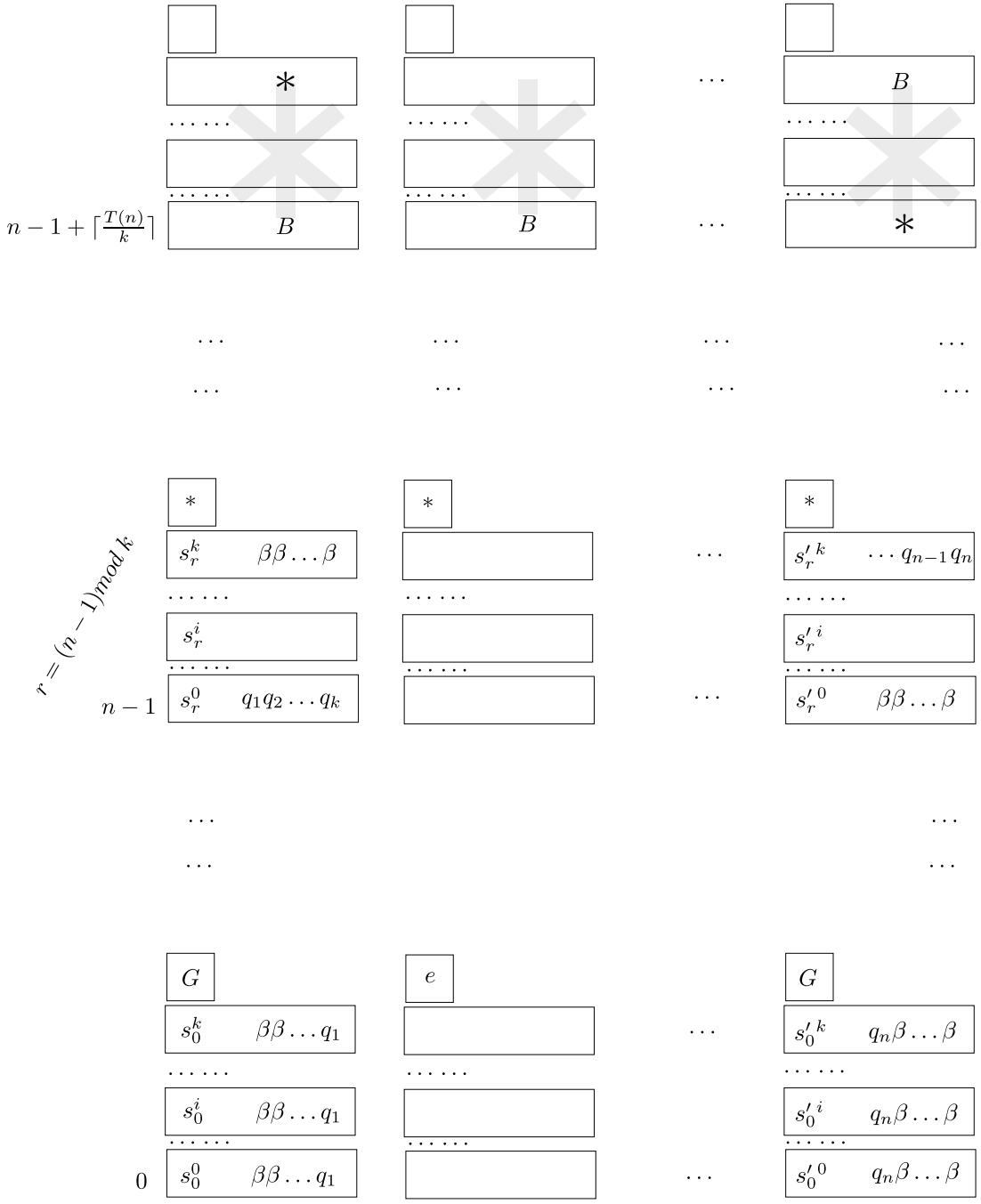


Fig. 12

Now that the construction is completed, let us take the time of examining

the result we have been lead to by our first idea : it is a new synchronizer, obtained by completing S_0 with

- not only a 2e-synchronizer
- but by state systems s_0^i and ${s'}_0^i$, in the end cells, meant to generate the conjugate signals.

Thus this synchronizer has non quiescent extremities. And so this speeding up makes sense only for 2e-synchronizers.

Theorem 7.4.1 *A 2e-synchronizer in time $T_{2e}(n)$ $(> n-1)$ can be speeded up into a 2e-synchronizer in time*

$$n - 1 + \left\lceil \frac{T_{2e}(n) - (n-1)}{k} \right\rceil .$$

Namely, any 2e-synchronizer in linear time $a.n$ can be speeded up into a 2e-synchronizer in time $2n$, or even better in time $\lceil n - 1 + \varepsilon n \rceil$, for any positive rational ε.

7.4.4 Fast synchronization for one end-synchronizers

But we shall not give up for one-end synchronizers : Grigorieff has suggested a way to adapt the preceding method, by using conjugate signals starting from sites $< 1, 0 >$ and $< n, n-1 >$, the first one with slope two times greater, i.e. $2b/a$, the second one with the same slope as before. We guess that they should join up approximately at the same place, and at time about $2n$. Let us now set this up neatly.

We shall prop our new signals (Figure 13) along

- the straightline D from point $(1, 0)$ having slope $2b/a$
- the straightline D' from point $(n, n-1)$ having slope $-b/b - a$
- which both cross the horizontal line $y = 2n - 2$ at point A

$$\text{abscissa}(A) = 1 + \frac{a}{b}(n-1).$$

We shall approximate these straightlines not exactly with the sites on the right of their points of integer ordinates, because for D we shall replace these sites by their left neighbours (except for the first sites which are on cell 0), so that the frontlines of the grouping should arrive not on the same cell but on two neighbouring cells. The signals S and S' thus obtained are the new *conjugate signals* for this case. At time $2n-2$, S' is on cell $\lceil 1 + \frac{a}{b}(n-1) \rceil$ and S is on the left neighbour $\lceil \frac{a}{b}(n-1) \rceil$.

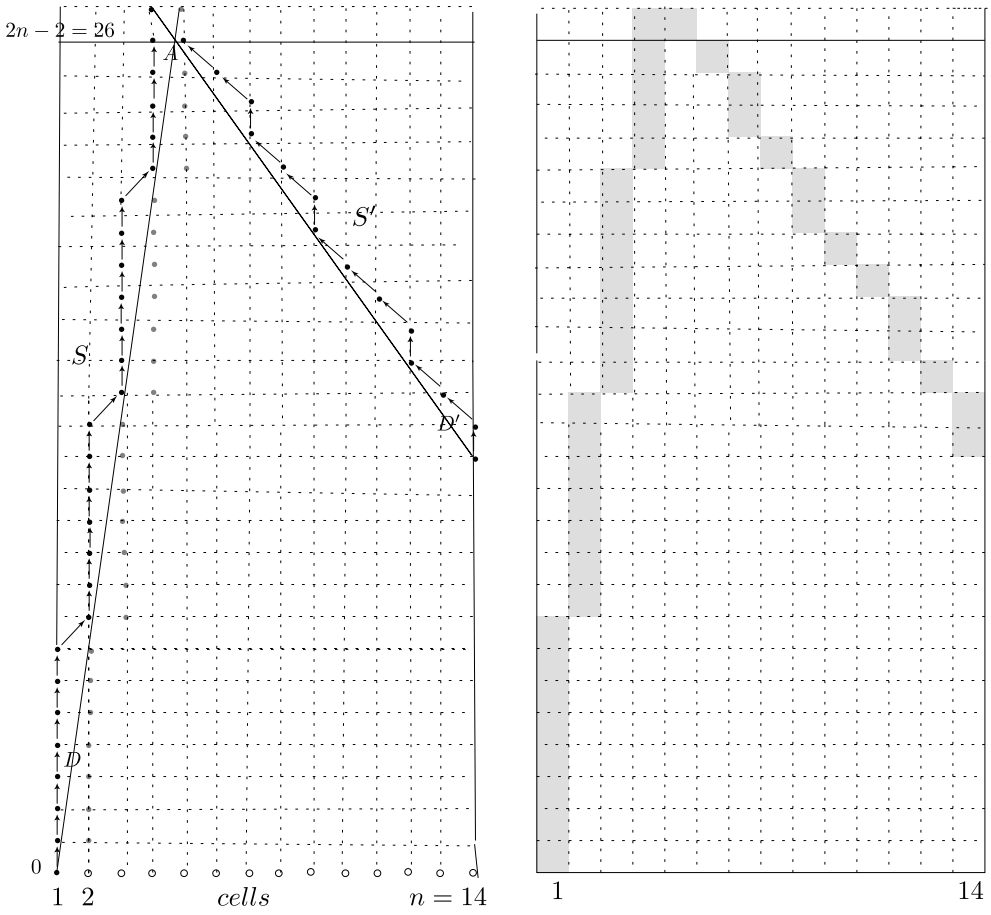


Fig. 13

The k-grouping will be guided by the new conjugate signals. It is started by state $(\beta, \ldots, \beta, G)$ on site $< 1, 0 >$, and state $B = (\beta, \ldots, \beta, \beta)$ set on site $< n, n-1 >$ by a speed-1 signal sent by the general. Each pair of conjugate signals is started by a state s_0 in cell 1 at time 0 and a state s_0' in cell n at time $n-1$. The $k+1$ states s_0' are set in cell n by the speed-1 signal sent by the general.

The number of cells not in state B when the frontlines of the grouping arrive on cell $\lceil \frac{a}{b}(n-1) \rceil$ and its right neighbour is

$$\left\lceil \frac{\lceil (n-1)\frac{a}{b} \rceil}{k} \right\rceil \quad \text{on the left, and} \quad \left\lceil \frac{n - \lceil (n-1)\frac{a}{b} \rceil}{k} \right\rceil \quad \text{on the right.}$$

Index of the first cell which is not in state B is

$$g = 1 + \left\lceil (n-1)\frac{a}{b} \right\rceil - \left\lceil \frac{\lceil (n-1)\frac{a}{b} \rceil}{k} \right\rceil$$

and of the last one is

$$d = \left\lceil (n-1)\frac{a}{b} \right\rceil + \left\lceil \frac{n - \lceil (n-1)\frac{a}{b} \rceil}{k} \right\rceil .$$

For the $k+1$ couples of conjugate signals we choose the same values $b = k$ and $a_0 = 0$, ... , $a_i = i$, ... , $a_k = k$. Then

$$d_i = 1 + \left\lceil (n-1)\frac{i}{k} \right\rceil + \left\lceil \frac{n - \lceil (n-1)\frac{i}{k} \rceil}{k} \right\rceil$$

and

$$g_{i+1} = 1 + \left\lceil (n-1)\frac{i+1}{k} \right\rceil - \left\lceil \frac{1 + \lceil (n-1)\frac{i+1}{k} \rceil}{k} \right\rceil .$$

Similar calculations as before lead to

$$g_{i+1} - d_i \leq 1 - \lfloor \frac{1}{k} - \frac{n-1}{k} \rfloor \leq 1.$$

As before this guarantees that any cell will receive a grouped state from one of the $(k+1)$ k-groupings.

Here the k-groupings, without or with simultaneous computing, will be stopped at time $2n-2$ and switched on the weak accelerated c.a's by a minimal time synchronizer started at time 0.

So at last we have the

Theorem 7.4.2 *A synchronizer in time $T_{1e}(n)$ $(> 2n-2)$ can be speeded up into*

$$2n - 2 + \left\lceil \frac{T_{1e}(n) - (2n-2)}{k} \right\rceil .$$

In particular, any synchronizer in linear time $a.n$ can be speeded up into a synchronizer in time $\lceil 2n - 2 + \varepsilon n \rceil$, for any positive rational ε.

7.5 Speeding up by a constant for synchronizers

Let us come back to the weak accelerated c.a of a recognizer or a synchronizer : its states and k-grouped input enable it to compute in one time-step what the original c.a computed in k time-steps. But nothing forbids it of doing less, of doing only what the original c.a did, or only a little less, what the original c.a did in h time-steps, $h < k$. What is needed for this is only that all the grouped cells be advised, at a determined instant, that they must work faster, or slower. All changes of speed must be commanded by auxiliary synchronizations.

Unluckily, synchronizations are heavy and unpractical tools : they need two borders, or states that can play the part of borders, and then they take the time they can and not the time we want !

The idea we have of using the fast speed during one time-step only and then to come back to the original speed will be very easy to implement with a synchronizer (or a parallel recognizer), because their accelerated c.a's have a minimal time synchronizer as a component. The fire state of this auxiliary synchronizer will start one only step of fast computing, by which we gain $(k-1)$ time-steps, as k time-steps are replaced by one.

Let us beware that, if from the auxiliary synchronization up, synchronization (or recognition) occurs after a time t comprised between 1 and k, this time t will be replaced by 1 time-step, and if synchronization (or recognition) takes minimal time, then there is no change whatever in process nor time. We keep these cases into account mostly to obtain a general formula.

If we group without computing at the same time, the result is not nice and simple, consequently not very interesting. So we consider only the case when computing starts with the grouping. Moreover, to express the result smoothly, we had better, here as we have already done previously, use the delay $D(n)$ between the minimal time ($2n-2$ for a synchronizer, $n-1$ for a 2e-synchronizer, and $n+1$ for a parallel recognizer) and the actual time, rather than the time itself.

We note $h = k - 1$.

The new synchronization (or parallel recognition) time is

$$D'(n) = \begin{cases} 0 & \text{if } D(n) = 0 \\ 1 & \text{if } 1 \leq D(n) \leq h+1 \\ D(n) - h & \text{if } D(n) \geq h+1 \end{cases}$$

If we want a formula for all cases we must write

$$D'(n) = min(D(n), max(1, D(n) - h)).$$

We can gain h time-steps only if $D(n)$ is strictly greater than h. This according to mere common sense.

Let us notice at the end of this chapter, because it will be of use later, that all we have just done by grouping cells by k can a fortiori be done by grouping them by $k' > k$.

Chapter 8

Synchronization times

8.1 General considerations

8.1.1 Justification

The problem of synchronization is not only a nice cellular automata problem.

If one needs to change, at some determined instant, the working of some c.a, to stop some process, or to freeze it, to release it, to start some new process ... cells must be warned all together at this moment that transition rules change. But cells, except the first one, are not accessible, at least in sequential c.a's.

(As for parallel c.a's, if we should consider them, we should also wonder why inputs used at time 0 could not be used at any other time if needed ! So that no synchronization would ever be needed. But we once more insist that these c.a's are mere study auxiliaries).

So all we can do is add to the c.a some built-in synchronizer, or establish rules by which inner events, that is certain states at certain places, start an auxiliary synchronizer. The fire state of this synchronizer will warn the cells. We have already used such synchronizers, to build a sequential c.a from a parallel one in 3.3.3, to stop the grouping process and start the fast computation in 7.3.

Up to now, we have built only 2 synchronizers, Minsky's and Mazoyer's, but this is enough to understand that synchronizing needs work, takes at least $2n-2$ transitions, and above all a time that we cannot choose at our convenience. For this reason, we should like to know beforehand what synchronizing times may be realized.

8.1.2 Which synchronizers ?

In the preceding chapters we have encountered different sorts of synchronizers, those with one general at one end (1e-), those with two generals at the two ends (2e-) and those with one general anywhere in the line. The two most important families are the 1end- and 2ends-synchronizers. Their two families of synchronizing times are two families of functions (respectively $\mathbb{N}^*$ and $\mathbb{N}^* \setminus \{1\} \mapsto$

$\mathbb{N}$) which are naturally different, as common sense makes us suspect and as is highlighted by considering their minimal elements

$$T_{1min}(n) = 2n - 2$$

$$T_{2min}(n) = n - 1.$$

If these two families do not contain the same functions, we nevertheless expect them to have analogous closure properties.

8.1.3 Synchronization delays

All synchronizing times, for 1e- and 2e-synchronizers, may be written

$$T_{1e}(n) = T_{1min}(n) + D_1(n)$$

$$T_{2e}(n) = T_{2min}(n) + D_2(n)$$

where D_1 and D_2 are positive functions, which we call synchronization delays. It is equivalent to study the families of synchronizing times or the families formed by these delays , but the latter are both families of positive functions, containing the null function, so they appear far more pleasant to study. We shall denote these two families $\mathcal{SD}_1$ and $\mathcal{SD}_2$.

A new question springs up to our mind : couldn't these two families, which have two important features in common, be the same one ? This question will become more stressing as we shall find in the sequel that the two families have quite a lot of functions in common.

8.2 Summary of results already established

8.2.1 Starting points

Till now we have actually built two 1-end synchronizers. Synchronizing time of the first, Minsky's, has a complicated expression (1.3.5), and it would be very surprising if we should ever need to end or start some process in a c.a at such a time ! Minimal synchronizing time $T_{1min}(n) = 2n - 2$, time of Mazoyer's c.a or its famous predecessors, seems bond to be more useful for later constructions.

From this one-end synchronizer, we have then built a 2ends-synchronizer, in minimal time $T_{2min}(n) = n - 1$. The construction, by symmetry, may be applied to any 1e-synchronizer.

Of course 1e-synchronizers have $n \geq 1$ cells, and 2e-synchronizers have $n \geq 2$ cells.

First functions in $\mathcal{SD}_1$ and $\mathcal{SD}_2$

The very first one is naturally the null function.

The second one is function $D_i(n) = n - 1$.

Indeed, a minimal time 1e-synchronizer can be made into a 2e-synchronizer by adding a second general which we shall let to sleep. For this 2e-synchronizer

$$D_2(n) = n - 1.$$

A minimal time 1e-synchronizer, once it is in fire state, can start a minimal time two ends synchronization. The resulting 1e-synchronizer has delay

$$D_1(n) = n - 1.$$

Change of variable

We shall choose to express synchronization delays not with variable n, but with variable

$$\nu = n - 1$$

which is length of lines minus one, ≥ 0 for 1e-synchronizers, and ≥ 1 for 2e-synchronizers, only because formulas will be neater. We shall then denote synchronization times and delays $t_i(\nu)$ and $d_i(\nu)$, and we shall keep notation $\mathcal{SD}_i$ for the families of delays, there will be no risk of confusing with the help of the context.

8.2.2 Slowing down and speeding up

Slowing down by a constant

From section 6.7 we know that

$$\text{if} \quad d_i(\nu) \in \mathcal{SD}_i \quad \text{then, for all k } \in \mathbb{N} \quad d_i(\nu) + k \in \mathcal{SD}_i$$

Linear slowing down :

With the very simple formula $T' = kT$ of section 6.6, for the delays we obtain $d'(\nu) = (k-1)T_{min} + kd(\nu)$. To create new delays, formula $d'(\nu) = kd(\nu)$ would suit us better. We can obtain it if the slowing down counter starts only at time T_{min}, which is possible if we adjoin to our synchronizer (1e- or 2e-) a minimal time synchronizer. So

$$\text{if} \quad d_i(\nu) \in \mathcal{SD}_i \quad \text{then, for all k } \in \mathbb{N} \quad k.d_i(\nu) \in \mathcal{SD}_i$$

linear speeding up :

From paragraphs 7.4.3 (2e-) and 7.4.4 (1e-synchronizers) we retain, once more, the only simple formula

$$T'(n) = T_{min} + \left\lceil \frac{D(n)}{k} \right\rceil$$

which gives

$$\text{if} \quad d_i(\nu) \in \mathcal{SD}_i \quad \text{then, for all k} \ \in \mathbb{N} \quad \left\lceil \frac{d_i(\nu)}{k} \right\rceil \in \mathcal{SD}_i$$

Speeding up by a constant

result of section 7.5 is not too complicated if delay $d(\nu)$ is strictly positive : for $h \in \mathbb{N}$

$$d'_i(\nu) = max(\ \ 1, d_i(\nu) - h \ \ ) \in \mathcal{SD}_i.$$

If $d_i(\nu)$ should be 0 for some values of ν, $d'_i(\nu)$ would also be 0 for these values, and formula becomes more complicated :

$$d'_i(\nu) = min[\ \ d_i(\nu), max(1, d_i(\nu) - h) \ \ ] \in \mathcal{SD}_i.$$

8.2.3 Finite modifications

This operation will be very practical in the sequel for some convenient rectifications. We wish to show that

$$\text{if } d_i(\nu) \in \mathcal{SD}_i \text{ and}$$

$$d'_i \neq d_i \text{ for a finite number of values of } \nu,\ \nu_1 < \ldots < \nu_k$$

then

$$d'_i \in \mathcal{SD}_i$$

Let then S_2 be a 2e-synchronizer with delay $d_2(\nu)$. We complete this 2e-synchronizer with

- a minimal time synchronizer Σ_2 which returns to quiescent state after fire
- in each cell a clock which counts up to $\sup(\nu_k,\ d'(\nu_1), \ldots, d'(\nu_k))$ and has marked values $\nu_1, \ldots, \nu_k$ memorized.

Thus, states have three components.
The behavior of this new c.a is as follows

1/ if the clocks are not on a marked value when Σ_2 fires at time ν, then we know that $\nu \neq \nu_1, \ldots, \nu_k$. The new c.a then behaves as S_2 does, and fires when S_2 does, i.e. with delay $d_2(\nu)$

2/ if the clocks are on a marked time ν_i when Σ_2 fires at time ν, then they are reset to 0 and fire will occur after $d'_2(\nu_i)$ units of time.

If we deal with a 1e-synchronizer, the only difference is in the marked times of the clocks, which must be $2\nu_1, \ldots, 2\nu_k$. The clocks must of course count up to $\sup(2\nu_k, d'_1(\nu_1), \ldots, d'_1(\nu_k))$.

Let us not wait a second more to use this new possibility. We start by observing that it could be reasonable to admit that a synchronization delay should not have value 0 for arbitrarily great values of ν, in which case it could have value 0 only for a finite number of values, $\nu_1, \ldots, \nu_k$. By modifiying d for these k values, we have a new synchronization delay $d'(\nu_i)$ which is strictly positive. When speeding up by a constant, by first modifiying d_i, replacing the zero values by 1, we obtain

$$d"_i(\nu) = max(1, d'_i(\nu) - h) = max(1, d_i(\nu) - h)$$

and we can forget the restriction we had made concerning this formula.

8.3 Stretching the lines (spatial homothety)

Our first result will be better expressed with the synchronizing times and the full length n of the lines

Proposition 8.3.1 *if $T_i(n) \in \mathcal{ST}_i$ then, for all $k \in \mathbb{N}$, $T_i(kn) \in \mathcal{ST}_i$.*

where $\mathcal{ST}_i$ is the family of synchronizing times for ie-synchronizers, $i = 1$ or 2. We shall give the proof for $k = 2$, to avoid lenghty expressions. Let then S_i be a synchronizer in time $T_i(n)$, with set of states Q and transition δ. We build a new synchronizer with set of states $Q \times Q$ (Figure 1) and transition

$$\Delta((p_l, q_l), (p, q), (p_r, q_r)) = (\delta(q_l, p, q), \delta(p, q, p_r))$$

$$\Delta(\beta, (p, q), (p_r, q_r)) = (\delta(\beta, p, q), \delta(p, q, p_r))$$

$$\Delta((p_l, q_l), (p, q), \beta) = (\delta(q_l, p, q), \delta(p, q, \beta)).$$

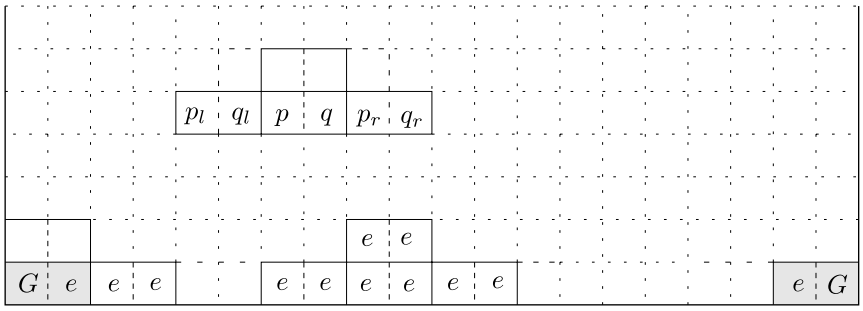


Fig.1

The quiescent state will be (e, e), the fire state $(*, *)$. For a 1e- synchronizer the general state will be (G, e). For a 2e-synchronizer we can identify states (G, e) and (e, G), (let $\overline{G}$ be the state identifying (G, e) and (e, G)), which may nevertheless be treated differently by transition Δ because one has the border state on its left and the other on its right

$$\Delta(\beta, (\overline{G}), (e, e)) = (\delta(\beta, G, e), \delta(G, e, e))$$

$$\Delta((e, e), (\overline{G}), \beta) = (\delta(e, e, G), \delta(e, G, \beta)).$$

It is clear that the diagram of the new c.a reproduces the diagram of the original one for a line twice longer. Hence the result, for $k = 2$. Proof is quite the same for $k > 2$.

In order to obtain a result for delays, we shall now make a more elaborate construction, separately for 1e- and 2e-synchronizers.

Starting with a 1e-synchronizer in time $T_1(n) = 2n - 2 + D_1(n)$, we first consider the synchronizer having as states k-tuples of states of the latter, so that a line of n cells reproduces a k-grouped line of kn cells. A line of kn cells should synchronize in $T_1(kn) = 2.kn - 2 + D_1(kn)$ time-steps.

To this synchronizer we add a minimal-time synchronizer, which can warn cells at time $2n$, 2 time-steps after its synchronization, that transition rules change.

If we let the first synchronizer compute k times faster till time $2n$, at this time it has reproduced $k.2n$ steps of the synchronization of the line of kn cells, so that the number of steps left to reach synchronization is

$$2.kn - 2 + D_1(kn) \ - \ k.2n = D_1(kn) - 2.$$

If from this time up, computation continues at normal speed, synchronization will appear at time

$$2n \ + \ D_1(kn) - 2$$

so that the delay is exactly $D_1(kn)$.

There is just a little correction for the cases when $D_1(kn) = 0$ or 1, because then, the accelerated synchronizer of the line of kn cells synchronizes at time

$$\left\lceil \frac{T_1(kn)}{k} \right\rceil = \left\lceil \frac{2kn - 2 + D_1(kn)}{k} \right\rceil = \begin{cases} 2n = (2n-2) + 2 & \text{if } k > 2 \\ \\ 2n = (2n-2) + 2 & \text{if } k = 2 \text{ and } D_1(2n) = 1 \\ 2n - 1 = (2n-2) + 1 & \text{if } k = 2 \text{ and } D_1(2n) = 0 \end{cases}$$

so that the delay is not $D_1(kn)$ but 2 (or 1) (of course this case is highly improbable !) Thus we have

Proposition 8.3.2 *if $D_1(n) \in \mathcal{SD}_1$, and $k > 2$, then $max(2, D_1(kn)) \in \mathcal{SD}_1$*

Proposition 8.3.3 *if $D_1(n) \in \mathcal{SD}_1$, then $D_1(2n) + \chi_{(D_1(2n) \leq 1)} \in \mathcal{SD}_1$*

where $\chi_{(D_1(2n) \leq 1)} = 1$ if $D_1(2n) = 0$ or 1, and $\chi_{(D_1(2n) \leq 1)} = 0$ otherwise.

For a 2e-synchronizer, in time $T_2(n) = n - 1 + D_2(n)$, the proof is similar. We consider the synchronizer having as states k-tuples of states reproducing a k-grouped line of kn cells, whose synchronizing time should be $T_2(kn) = kn - 1 + D_2(kn)$. This synchronizer is accelerated k times and completed by a minimal-time 2e-synchronizer, which warns cells at time n to continue at normal speed. The total time for synchronization is then

$$n \ + \ T_2(kn) - kn = n + (kn - 1 + D_2(kn)) - kn = n - 1 + D_2(kn)$$

and the delay is $D_2(kn)$.

In the, quite improbable, case when $D_2(kn) = 0$, the accelerated synchronizer is synchronized at time

$$\left\lceil \frac{T_2(n)}{k} \right\rceil = \left\lceil \frac{kn - 1 + D_1(kn)}{k} \right\rceil = n = (n-1) + 1$$

so the delay is not $D_2(kn) = 0$ but 1.

Proposition 8.3.4 *if $D_2(n) \in \mathcal{SD}_2$, then $max(1, D_2(kn)) \in \mathcal{SD}_2$*

8.4 Combining synchronization times

Up to now, if we do not take the auxiliary minimal-time synchronizer into account, we have used only one synchronizer. We shall now use two, which we suppose to be of the same type, 1e- or 2e-, and we moreover precise that if we deal with two 1e- synchronizers, the two generals are both on the left end, so the product is still 1e-, (we postpone passing from one type of synchronizer to the other till last section). Proofs are valid for the 2 types : beware that in this section indexes 1 and 2 do not refer to the type of synchronizers.

8.4.1 Min and max

Let

S_1 be a synchronizer with fire state $*_1$ and delay d_1,

S_2 a synchronizer with fire state $*_2$ and delay d_2.

We make the product of these 2 c.a's. If we merge all states having at least one fire component, than we have a synchronizer with delay $\min(d_1(\nu), d_2(\nu))$. If we decide that the fire states $*_1$ and $*_2$ are permanent, and fire state of the product is $(*_1, *_2)$ then we have a synchronizer with delay $\max(d_1(\nu), d_2(\nu))$.

8.4.2 Sum

We remind that we argue for both the 1e-/2e-cases (Figure 2 illustrating the 2e-case). Let

S_1 be a synchronizer with fire state $*_1$, and delay d_1,

S_2 a synchronizer with fire state $*_2$, and delay d_2,

and S a minimal time synchronizer with fire state $*$ (delay 0).

We make the product of these 3 c.a's, and let it work as follows :

- $*$ freezes S_2 on configuration $C_2(2\nu)/C_2(\nu)$, (while S stops on state $*$)

- $*_1$ unfreezes S_2, (while S_1 stops on state $*_1$)

- fire state of the product will be $(*, *_1, *_2)$.

Delay of the product is then $d_1 + d_2$.

Let us notice that:

- if $d_1 = 0$, S_2 is not blocked

- if $d_2 = 0$, $C_2(2\nu)/C_2(\nu)$ is in state $*_2$

so that in all cases the triple fire appears at time $d_1 + d_2$.

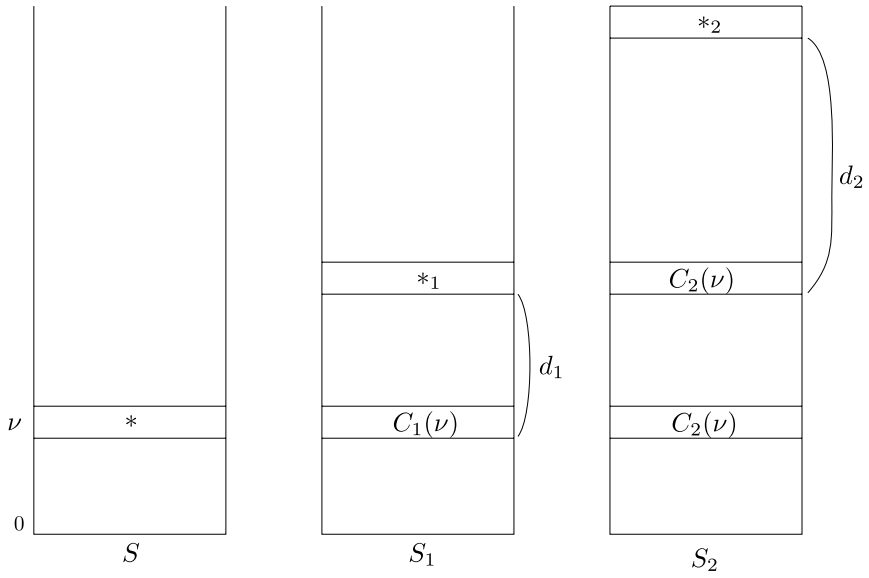

$*_2$
d_2
$*_1$
$C_2(\nu)$
d_1
ν
$*$
$C_1(\nu)$
$C_2(\nu)$
0
S
S_1
S_2

Fig. 2

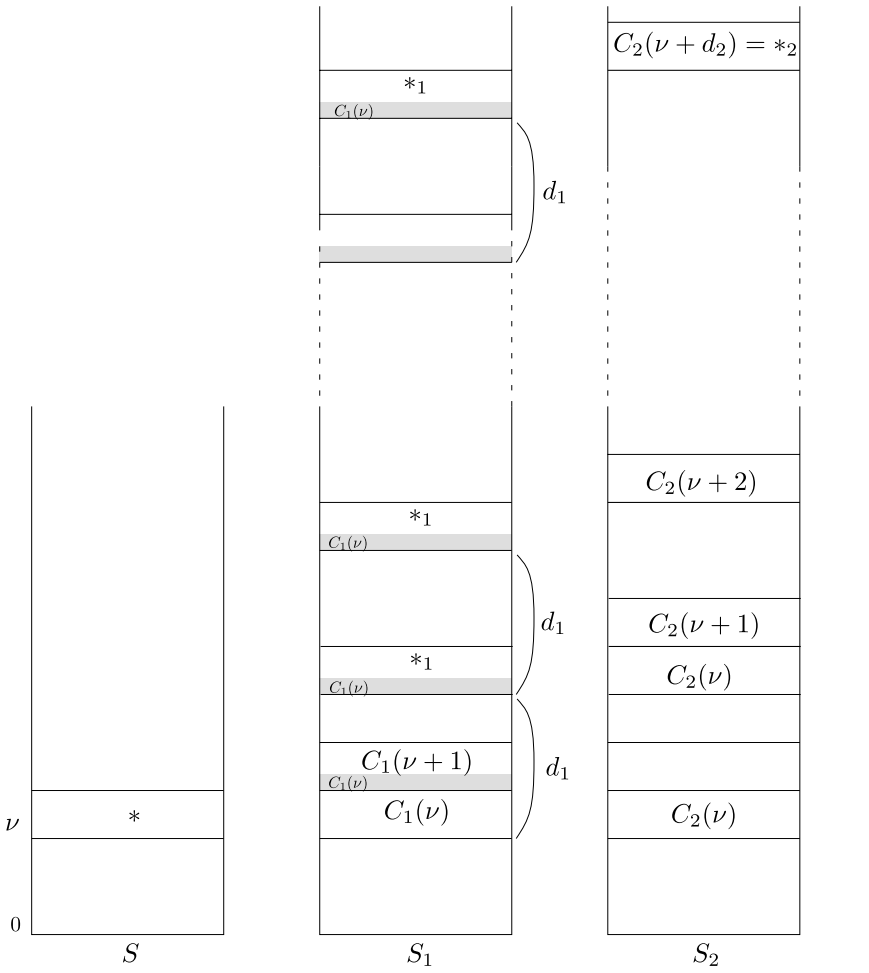

$C_2(\nu + d_2) = *_2$
$*_1$
$C_1(\nu)$
d_1
$C_2(\nu + 2)$
$*_1$
$C_1(\nu)$
d_1
$C_2(\nu + 1)$
$*_1$
$C_2(\nu)$
$C_1(\nu)$
d_1
$C_1(\nu + 1)$
$C_1(\nu)$
ν
$*$
$C_1(\nu)$
$C_2(\nu)$
0
S
S_1
S_2

Fig. 3

8.4.3 Product

Now we let the same product work as follows (Figure 3), when d_1 and d_2 are not 0

- $*$ blocks S_2 (on configuration $C_2(2\nu)/C_2(\nu)$) and definitively records configuration $C_1(2\nu)/C_1(\nu)$ in some part of cells of S_1, (the lower part in Figure 3)

- $*_1$ unblocks S_2 for one time-step. It also warns S_1 to start again its synchronization process from the recorded configuration up

- fire state of the product is $*_2$ on S_2.

It appears after delay

$$d_1.d_2 + 1.$$

In order for this formula to keep valid if $d_1 = 0$ or $d_2 = 0$ we agree that

- $*$ and $*_2$ appearing simultaneously produce the fire state of the product at next time-step

- as well as $*$ and the first $*_1$, (recognizable because there is no memorization yet).

It is worth-while complicating a little the preceding c.a in order to obtain product $d_1.d_2$, for what we must gain one time-step. Modifications are the following

- state $*$ gives rise to a new state $**$ before disappearing

- if $*$ and $*_2$ appear together ($d_2 = 0$) we have fire state

- if $*$ and the first $*_1$ appear together ($d_1 = 0$) we have fire state

- if $**$ and $*_2$ appear together ($d_2 = 1$) fire will be $*_1$

- if $**$ and $*_1$ appear together ($d_1 = 1$) fire will be $*_2$

- for the general case, state $**$ replaces the memorized configuration $C_1(2\nu)/C_1(\nu)$ by configuration $C_1(2\nu + 1)/C_1(\nu + 1)$, so that one time-step is gained before the second appearance of state $*_1$.

8.4.4 Semi-differences

Let p be any integer greater than 1.We shall build a c.a synchronizing with delay

$$\delta_p(\nu) = \begin{cases} d_1(\nu) - d_2(\nu) & \text{if} \quad \left\lceil \frac{d_1(\nu)}{p+1} \right\rceil > \left\lceil \frac{d_2(\nu)}{p} \right\rceil \\ \left\lceil \frac{d_2(\nu)}{p} \right\rceil & \text{if} \quad \left\lceil \frac{d_1(\nu)}{p+1} \right\rceil \leq \left\lceil \frac{d_2(\nu)}{p} \right\rceil \end{cases}$$

The first condition implies $d_1 > d_2$, it is even a little more demanding, and if $d_1 - d_2 < 0$ we are in the second case.

This result looks a little complicated at first, but we can explain it a little. When $d_1 > d_2$, let us observe that $d_1 - d_2$ may be very small, smaller than d_1 and d_2, even much smaller. So the idea we have to build $d_1 - d_2$ is to use accelerated synchronizing delays, $\lceil d_1/p \rceil$, $\lceil d_2/p \rceil$, with p large so that at least $\lceil d_2/p \rceil$ slips under $d_1 - d_2$. Now

$$\lceil \frac{d_2}{p} \rceil \leq d_1 - d_2 \Leftrightarrow \frac{d_2}{p} \leq \frac{d_1}{p+1}.$$

Thus, if our condition to capture the difference is not exactly that $\lceil d_2/p \rceil$ slips under $d_1 - d_2$, it is just a little more.

With large values for p we shall have more chances of realizing our condition, which means that we shall have as synchronizing time the difference $d_1 - d_2$, and not $\lceil d_2/p \rceil$, for more (lengthes of) lines.

In section 7.4 we have seen how to build, from an original synchronizer S_0 (1e- or 2e-), a synchronizer accelerated k times, which included :

- a minimal time synchronizer Σ
- and $k+1$ many k-grouper-computers,

and we did not fail to point out that, once grouping is achieved, we are not forced to use all the power it allows, we can accelerate less than k times, we can even compute at the original speed.

Here we make such a construction twice, with $k = p+1$ for S_1 and $k = p$ for S_2, with a common Σ.
The c.a we obtain will work as follows :

1/ it groups and computes normally up to time $2\nu/\nu$ when Σ synchronizes

2/ from this time up

the grouped c.a's from S_1 are accelerated $p+1$ times
the grouped c.a's from S_2 are accelerated p times

so they shall fire at respective delays

$$\left\lceil \frac{d_1}{p+1} \right\rceil \quad \text{and} \quad \left\lceil \frac{d_2}{p} \right\rceil$$

3/ if the accelerated S_1 has already synchronized or is just synchronizing when S_2 fires, then our c.a fires. The condition is therefore

$$\left\lceil \frac{d_1}{p+1} \right\rceil \leq \left\lceil \frac{d_2}{p} \right\rceil$$

and the delay is then

$$\left\lceil \frac{d_2}{p} \right\rceil$$

4/ before we go on, let us remind how the halting times of a recognizer or the fire of an accelerated synchronizer have been defined (7.1.2) :

the accelerated S_2 enters fire state if preceding configuration has lead to the fire of S_2 in p' time-steps, with $1 \leq p' \leq p$: $d_2 = p(\left\lceil \frac{d_2}{p} \right\rceil - 1) + p'$. As we establish the transition function of the accelerated S_2, it is possible to complete the fire state by the value of p', or as well the value of

$$p - p' = p \left\lceil \frac{d_2}{p} \right\rceil - d_2.$$

So each cell of the accelerated S_2 knows this number from the time it fires, and can memorize it in a blocked counter for later use.

5/ Suppose now that

$$\left\lceil \frac{d_1}{p+1} \right\rceil > \left\lceil \frac{d_2}{p} \right\rceil$$

that is, the accelerated S_1 has not yet synchronized when the accelerated S_2 fires : in this case, it continues at normal speed.

When the accelerated S_2 fires, at delay $\lceil d_2/p \rceil$, the accelerated S_1 has emulated $(p+1)\lceil d_2/p \rceil$ steps towards synchronization, so the number of steps still left to obtain synchronization is

$$d_1 - (p+1) \left\lceil \frac{d_2}{p} \right\rceil$$

6/ finally the fire state of the accelerated S_1 unblocks the counters and fire state will occur when counters mark 0. Synchronization delay is then

$$\left\lceil \frac{d_2}{p} \right\rceil \quad + \quad (d_1 - (p+1) \left\lceil \frac{d_2}{p} \right\rceil) \quad + \quad (p \left\lceil \frac{d_2}{p} \right\rceil - d_2) = d_1 - d_2$$

The c.a we have just built thus synchronizes at the announced delay.

Complicating notably this basic construction, we may obtain the more pleasant

Proposition 8.4.1 *if d_1 and d_2 are two delays in $\mathcal{SD}_i$ and $d_1 \geq (1+\varepsilon)d_2$, then the difference $d_1 - d_2$ is also in $\mathcal{SD}_i$.*

proof : a sufficient condition for δ_p to equal the difference $d_1 - d_2$ is

$$\frac{d_1}{p+1} \geq \frac{d_2}{p} + 1$$

which may be written

$$d_1 - d_2 \geq \frac{d_2}{p} + (p+1).$$

Let us choose $p \geq 2/\varepsilon$. Hypothesis of the proposition then implies

$$d_1 - d_2 \geq \frac{2}{p} d_2 = \frac{d_2}{p} + \frac{d_2}{p}.$$

If $\frac{d_2}{p} \geq p+1$, we are assured that $\delta_p(d_1, d_2) = d_1 - d_2$, so the δ_p-c.a suits us. But if $\frac{d_2}{p} < p+1$, we must obtain $d_1 - d_2$ with some other c.a. The two c.a's will work simultaneously (a product), and only one of the two will be observed as soon as the case we are in is determined.

The second construction is certainly cumbersome, but here we do not spare the states. It will comprise :

- the minimal-time synchronizer Σ (it can be the same as the one in the δ_p-c.a)
- S_1
- 2 sets of $p(p+1)+1 \quad p(p+1)$-grouper-computers for S_1
- 1 set of $p(p+1)+1 \quad p(p+1)$-grouper-computers for S_2.

They all work normally up to minimal time, $2\nu/\nu$.

We shall now carefully list all the cases that may present themselves and what synchronization time is adopted in each.

- if at minimal time we have the two synchronizations, $*_1$ and $*_2$, this is declared fire state
- if at minimal time we have only synchronization $*_2$, fire will be the fire of S_1.
- for all following cases $d_2 \geq 1$, so also $d_1 - d_2 \geq 1$ (because $d_1 - d_2 \geq \varepsilon d_2$).

we now observe result of the next time-step

- if the $p(p+1)$-accelerated S_2 is synchronized, then we know that $d_2 \leq p(p+1)$, and we may even check the exact value of d_2
- if $d_2 = p(p+1)$, or if the $p(p+1)$-accelerated S_2 is not synchronized, then $d_2 \geq p(p+1)$, the synchronizing state will be the fire of the δ_p-c.a
- if the $p(p+1)$-accelerated S_2 is synchronized and $d_2 < p(p+1)$, then $d_2 + 1 \leq p(p+1)$, we observe the first $p(p+1)$-accelerated S_1
- if it is also synchronized, we can check the value of d_1, and if $d_1 = d_2 + 1$ we declare this is a fire state
- in all remaining cases $d_1 - d_2 \geq 2$

- during this first time-step after minimal time, if the second $p(p+1)$-accelerated S_1 has worked at normal speed, the number of time-steps left for its synchronization is $d_1 - 1$

- for the next time-step, this second $p(p+1)$-accelerated S_1 is accelerated, but only $d_2 + 1$ times

if ever it synchronizes now, we know that $d_1 - 1 \leq d_2 + 1$, so $d_1 - d_2 = 2$, we declare this is a synchronization state

- if this is not the case, the second $p(p+1)$-accelerated S_1 resumes normal speed. The number of time-steps left for synchronization being

$$d_1 - 1 - (d_2 + 1) = d_1 - d_2 + 2$$

the total time for its synchronization is $d_1 - d_2$.

In all cases, $d_1 = d_2 = 0$, $d_2 = 0$, $d_1 - d_2 = 1$, $d_1 - d_2 = 2$, $d_1 - d_2 > 2$, the synchronizing time is indeed $d_1 - d_2$.

8.5 The families of synchronization delays

In the preceding sections we have already proved that both families $\mathcal{SD}_1$ and $\mathcal{SD}_2$ contain

- the zero function $d_i(\nu) = 0$
- the identity function $d_i(\nu) = \nu$

and are closed for operations which, from one function d_i, or two functions d_i and d'_i, of the family, and k any positive integer, give :

- $d_i + k$
- $\max(1, d_i - k)$
- $k.d_i$
- $\lceil d_i/k \rceil$
- d_i modified for a finite number of ν (in particular $\max(1, d_i)$)
- $\min(d_i, d'_i)$, $\max(d_i, d'_i)$
- $d_i + d'_i$
- $d_i.d'_i$
- $\delta_p(d_i, d'_i)$ for $p \geq 2$

We shall now combine these operations.

Common sense leads us to think that synchronization delays should be increasing functions. With pseudo-difference we can nevertheless obtain functions which are not increasing, but quite delicate constructions are involved, we must admit !

Thus we get

Proposition 8.5.1 *for any polynomial $Q \in \mathbb{Q}^+[X]$, if $d_i \in \mathcal{SD}_i$, then $\lceil Q(d_i) \rceil \in \mathcal{SD}_i$. Otherwise said : the two families of synchronization delays are closed for composition with polynomials having positive rational coefficients.*

indeed, there exists an integer a such that $P = aQ \in \mathbb{N}[X]$

- by product : $d_i^k \in \mathcal{SD}_i$
- by slowing down : $a_k d_i^k \in \mathcal{SD}_i$
- by finite summing : $P = a_p d_i^p + \ldots + a_1 d_i + a_0 \in \mathcal{SD}_i$
- by linear speeding up : $\lceil Q(\nu) \rceil = \lceil P(\nu)/a \rceil \in \mathcal{SD}_i$

as a corollary, since $\nu \in \mathcal{SD}_i$

Corollary 8.5.2 *for any polynomial $Q \in \mathbb{Q}^+[X]$, $\lceil Q(\nu) \rceil \in \mathcal{SD}_i$.*

We shall now consider polynomials with positive or negative coefficients, but only those which have positive values when the variable is positive ($x \in \mathbb{R}^+$). To make it a little shorter we shall only say "with positive values".

Proposition 8.5.3 *for any polynomial $Q \in \mathbb{Q}[X]$ having positive values, $\lceil Q(\nu) \rceil \in \mathcal{SD}_i$*

Proof : first, there exists a positive integer a such that $P = aQ \in \mathbb{N}[X]$. P has positive values. In P we can separate terms with positive and negative coefficients, so the polynomial writes

$$P(X) = P_1(X) - P_2(X)$$

with

$$P_1 \quad \text{and} \quad P_2 \quad \in \mathbb{N}^+[X]$$

$$\forall x \in \mathbb{R}^+ \qquad P_1(x) \geq P_2(x)$$

$$degree(P_1) > degree(P_2).$$

By preceding corollary we have

$$P_1(\nu) \quad \text{and} \quad P_2(\nu) \quad \in \quad \mathcal{SD}_i,$$

by pseudo-difference, for any integer p, 2 for example,

$$\delta_2(P_1(\nu), P_2(\nu)) \in \mathcal{SD}.$$

This delay is equal to $P_1(\nu) - P_2(\nu)$ for all ν such that

$$\left\lceil \frac{P_1(\nu)}{3} \right\rceil > \left\lceil \frac{P_2(\nu)}{2} \right\rceil$$

so also if

$$\frac{P_1(\nu)}{3} \geq \frac{P_2(\nu)}{2} + 1.$$

But, as $degree(P_1) > degree(P_2)$ this is the case for all values of ν except a finite number of $\nu_i, i \in I$. Then, by finite modification we can replace all $\delta_2(\nu_i)$ by $P(\nu_i)$ and we have obtained delay $P(\nu)$. Delay $\lceil Q(\nu) \rceil$ is then obtained by linear speeding up.

Proposition 8.5.4 *for any polynomial* $Q \in \mathbb{Q}[X]$ $\max(0, Q(\nu)) \in \mathcal{SD}_i$

Here again we consider $P = aQ \in \mathbb{N}^+[X]$, where we separate positive and negative coefficients

$$P(X) = P_1(X) - P_2(X),$$

and we consider two cases. If $degree(P_2) > degree(P_1)$, P has positive values for a finite number of values of ν, so we obtain delay $P(\nu)$ by finite modification of delay 0.

If $degree(P_1) > degree(P_2)$, the proof is the same as before, the semi-difference $\delta_2(\nu_i)$ is equal to $P_1 - P_2$ except for a finite number of values of ν, and for these values we modify $\delta_2(\nu_i)$ by $\max(0, P(\nu_i))$. We end by a linear speeding up to go from P to Q.

All these results quite satisfy our common sense.

8.6 Relations between the two families of synchronization delays

Proposition 8.6.1 *if* $d_1(\nu) \in \mathcal{SD}_1$ *then* $\nu + d_1(\nu) \in \mathcal{SD}_2$

Indeed, if S_1 is a 1e-synchronizer with delay $d_1(\nu)$, we define the 2e-synchronizer with a sleeping right general, that is, all rules with no general or only the left one are the rules of S_1, and transitions involving the right general have the result of rules where this general is replaced by the quiescent state. The synchronizing-time is thus $2\nu + d_1(\nu)$ and the delay is $\nu + d_1(\nu)$.

Proposition 8.6.2 *if* $d_2(\nu) \in \mathcal{SD}_2$ *then* $\nu + d_2(\nu) \in \mathcal{SD}_1$

Let S_2 be a 2e-synchronizer with delay $d_2(\nu)$. We build a 1e-synchronizer as follows : its general sends two signals, a speed 1-signal with quick reflection on the border, and a signal of slope 3 (Figure 4), as in Minsky's synchronizer. When they meet, indicating the middle, the second signal stops and is replaced by a speed 1- signal. The two speed 1-signals reach the end cells at time $2n-2$. These immediately play the role of two generals, with the rules of S_2. The total synchronizing-time is then $2\nu + \nu + d_2(\nu)$, hence the result.

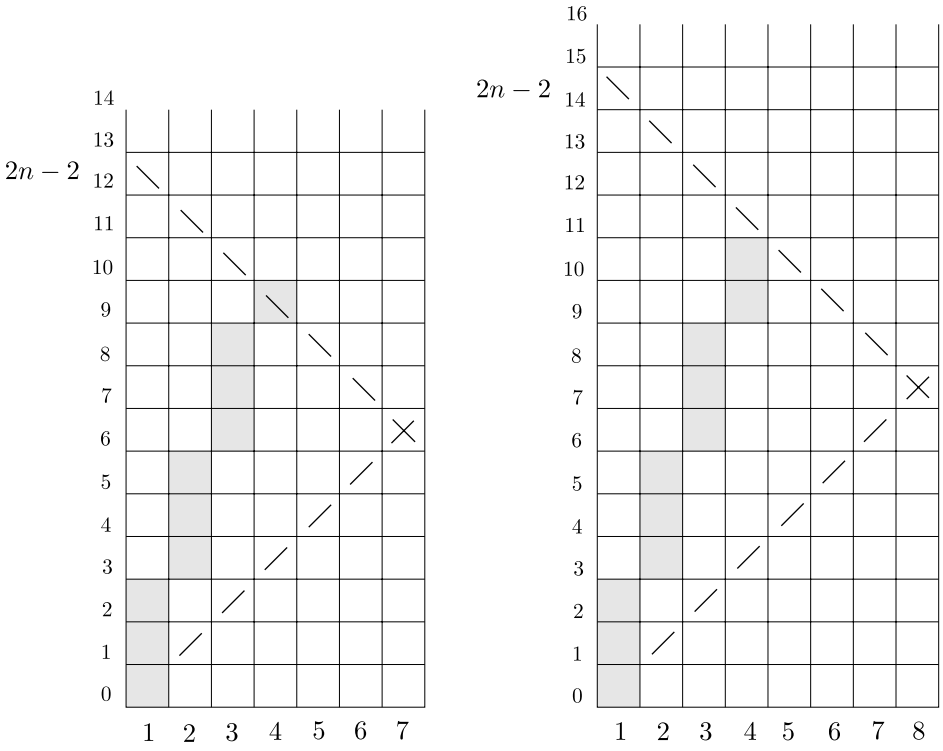


Fig.4

In the two preceding propositions, the result is that, if $d_i(\nu) \in \mathcal{SD}_i$ then $\nu + d_i(\nu) \in \mathcal{SD}_j$, where $\mathcal{SD}_j$ is the other family. But, as $\mathcal{SD}_j$ also contains function ν, we are much tempted to operate the difference

$$(\nu + d_i) - \nu$$

to conclude that d_i belongs to $\mathcal{SD}_j$

Alas, we can only do semi-differences δ_p, with p an arbitrary integer

$$\delta_p(\nu) = \begin{cases} d_i(\nu) & \text{if} \quad \left\lceil \frac{\nu + d_i}{p+1} \right\rceil > \left\lceil \frac{\nu}{p} \right\rceil \\ \left\lceil \frac{\nu}{p} \right\rceil & \text{otherwise} \end{cases}$$

Condition for $\delta_p(\nu)$ to coincide with $d_i(\nu)$ can be written

$$d_i > p\lceil \frac{\nu}{p} \rceil - \nu + \lceil \frac{\nu}{p} \rceil,$$

thus we may at least assert that any function of one of the two families of delays satisfying, for some integer p,

$$d_i > p + \lceil \frac{\nu}{p} \rceil$$

also belongs to the other family.

Corollary 8.6.3 *if $d_i(\nu) \in \mathcal{SD}_i$ and $d_i(\nu) > p + \lceil \frac{\nu}{p} \rceil$, then $d_i(\nu)$ also belongs to $\mathcal{SD}_j$ $(j = 3 - i)$.*

We shall give another, perhaps more usual and pleasant form, to this result :

Proposition 8.6.4 *if $D_i(n) \in \mathcal{SD}_i$ and satisfies some condition $D_i(n) > \varepsilon n$, then $D_i(n) \in \mathcal{SD}_j$ $(j = 3 - i)$.*

Indeed, let us first choose integer p greater than $2/\varepsilon$, so that $D_i(n) > \frac{n}{p/2}$. Then let

$$D'_i(n) = \begin{cases} D_i(n) & \text{if} \quad n > p^2 + p - 1 \\ \lceil \frac{n-1}{p} \rceil + p + 1 & \text{if} \quad n \leq p^2 + p - 1 \end{cases}$$

For the values of n smaller than $p^2 + p - 1$, which are in finite number, D'_i satisfies condition of preceding corollary.

For the values of n greater than $p^2 + p - 1$, we have

$$\begin{aligned} D'_i(n) = D_i(n) > \tfrac{n}{\frac{p}{2}} = p + \tfrac{n - \frac{p^2}{2}}{\frac{p}{2}} &= p + \tfrac{2n - p^2}{\frac{p}{2}} \\ &> p + \tfrac{n + p^2 - p - 1 - p^2}{p} = p + \tfrac{n-1}{p} + 1 \\ &> p + \lceil \tfrac{n-1}{p} \rceil \end{aligned}$$

D'_i, satisfying condition of the corollary for all n, thus belongs to $\mathcal{SD}_j$. Finally, by finite modification, we may conclude that $D_i(n)$ also belongs to $\mathcal{SD}_j$.

But in one case we have a better result :

Proposition 8.6.5 $\mathcal{SD}_1 \subseteq \mathcal{SD}_2$

Indeed, let S_1 be a 1e-synchronizer with delay $D_1(n)$. For any $n \geq 2$ we build a n-cells 2e-synchronizer having as states couples of states of S_1, and initial configuration

$$(G, e), (e, e), \ldots, (e, e), (e, G),$$

(where (G, e) and (e, G) are identified as in section 8.3), and working as S_1 and a symmetrical c.a, but 2 times faster as is possible because states are grouped by two (Figure 5). We may suppose that at time $\lfloor n/2 \rfloor$ the middle axis is located,

so that we find ourselves with two symmetrical S_1-synchronizers of length n. As they work 2 times faster, at time $n-1$ they are both in configuration of time $2n-2$ of $S_1(n)$, with $D_1(n)$ time-steps left to synchronization. Now we complete our c.a with an auxiliary minimal-time 2e-synchronizer, which warns cells to slacken pace and return to normal speed. So that $D_1(n)$ time-steps later, our two symmetrical S_1-synchronizers enter fire state $*$, the fire state of our 2e-synchronizer being $(*,*)$.

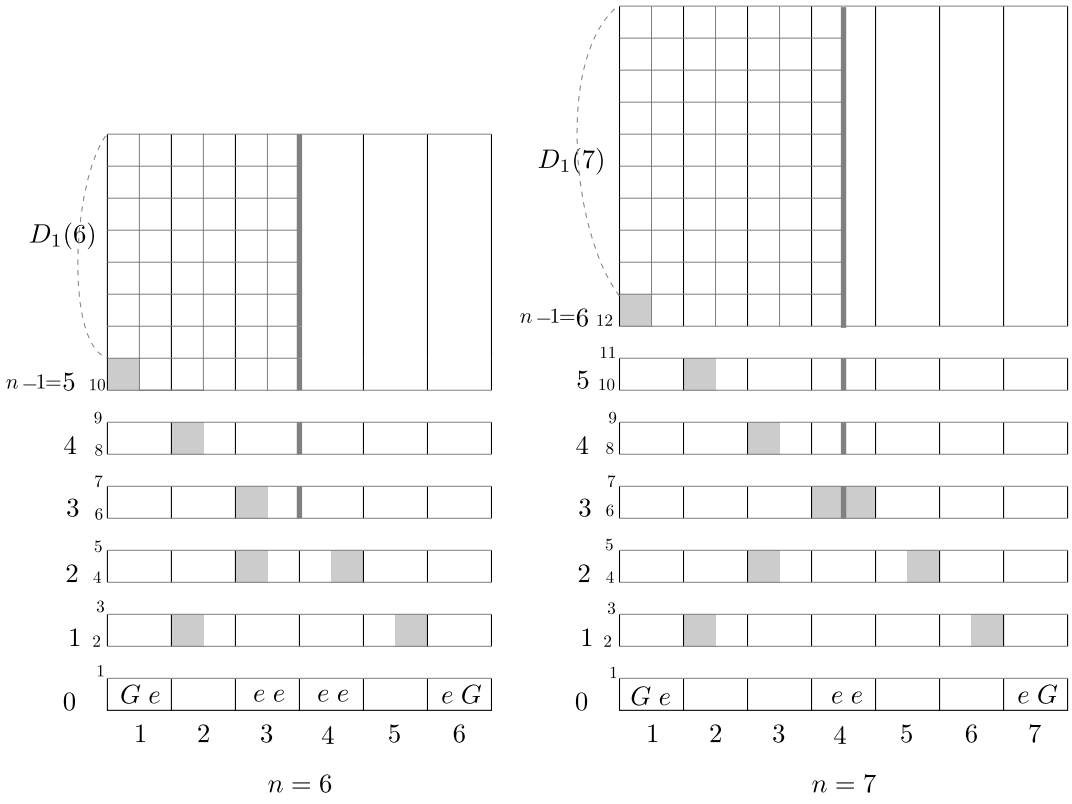


Fig.5

Chapter 9

Grouper c.a's

The basic idea of grouper c.a's is due to Olivier HEEN [23]. In this chapter we develop a formalization of O.Heen's notions which leads to an analysis of grouper c.a's as compositions of an ordinary c.a $\mathcal{A}$ together with a pair $\mathcal{R} = (\mathcal{B}, \mathcal{E})$, where $\mathcal{B}$ is another ordinary c.a and $\mathcal{E}$ is essentially an embedding of the states of $\mathcal{B}$ as areas in the plane.

All automata transformations done in chapter 7, in view of speeding up, can be interpreted as operations

$$\mathcal{A} \longmapsto \mathcal{R} \otimes \mathcal{A}$$

where $\mathcal{R}$ is some adequate, excessively simple, grouper c.a.

9.1 Paradigmatic examples

But first, to arouse the curiosity of the reader, and to let him have some intuitive perception, we start with a little movie sequence of two particular examples (which are to be studied in detail at the end of the chapter). The first figure of each sequence is an ordinary s.t.d, the second and third figures are two different presentations of the s.t.d of some geometrical c.a, which has no relation whatsoever with the preceding ordinary c.a. The fourth and fifth figures result from a marriage of the ordinary c.a and the geometrical c.a, used to a new reading of the s.t.d of the first c.a.

9.1.1 First exemple : the horizontal 3-grouper

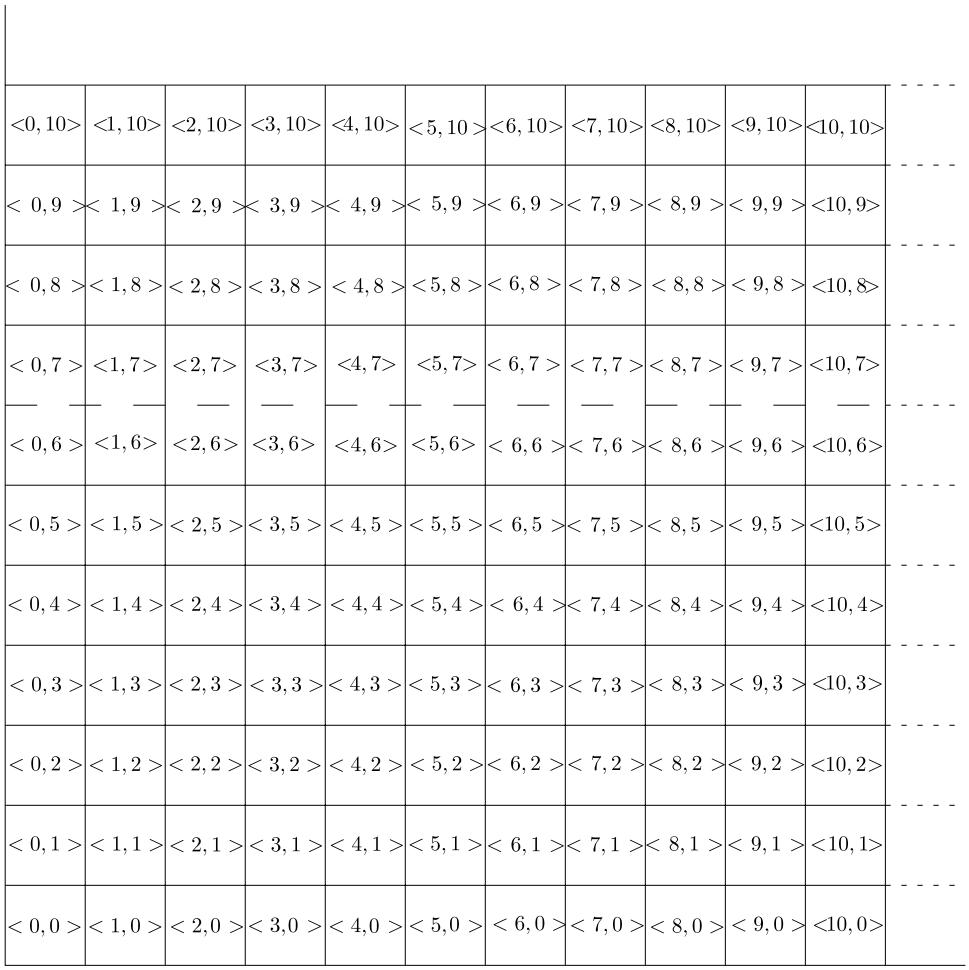


Fig A1 : s.t.d of any ordinary c.a

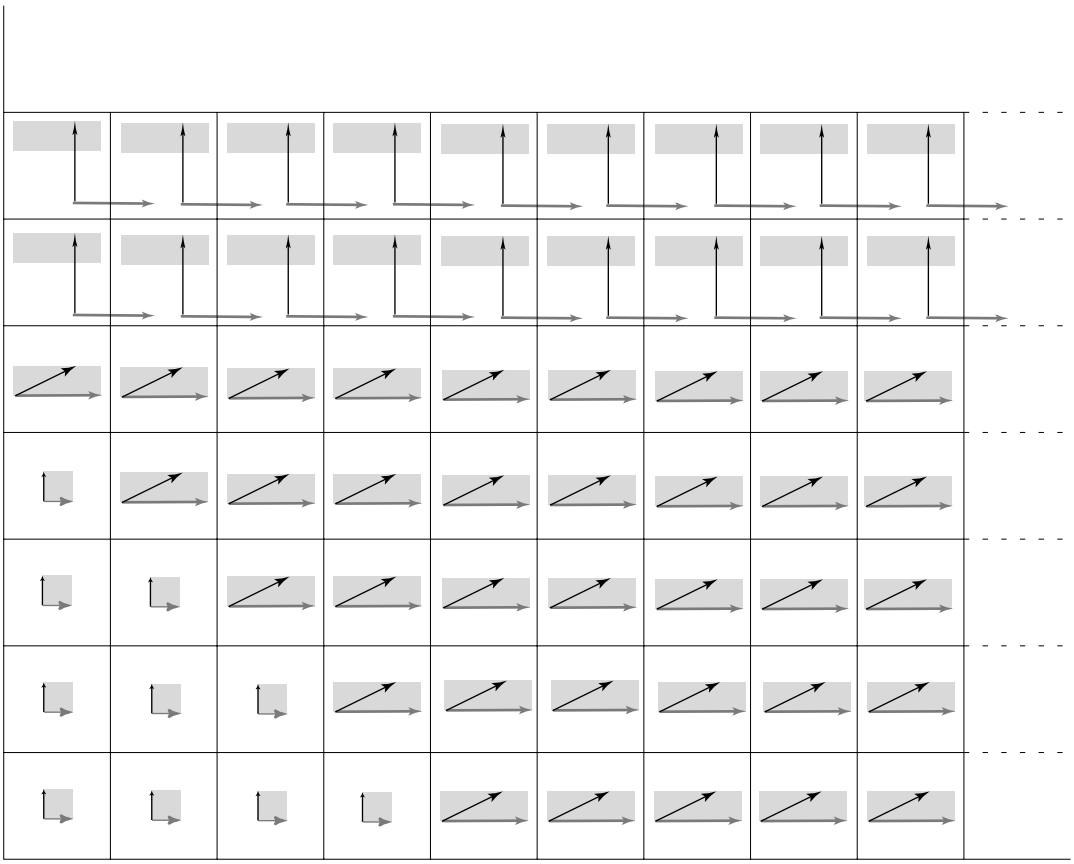


Fig A2 : s.t.d of a particular geometrical c.a (horizontal grouper)

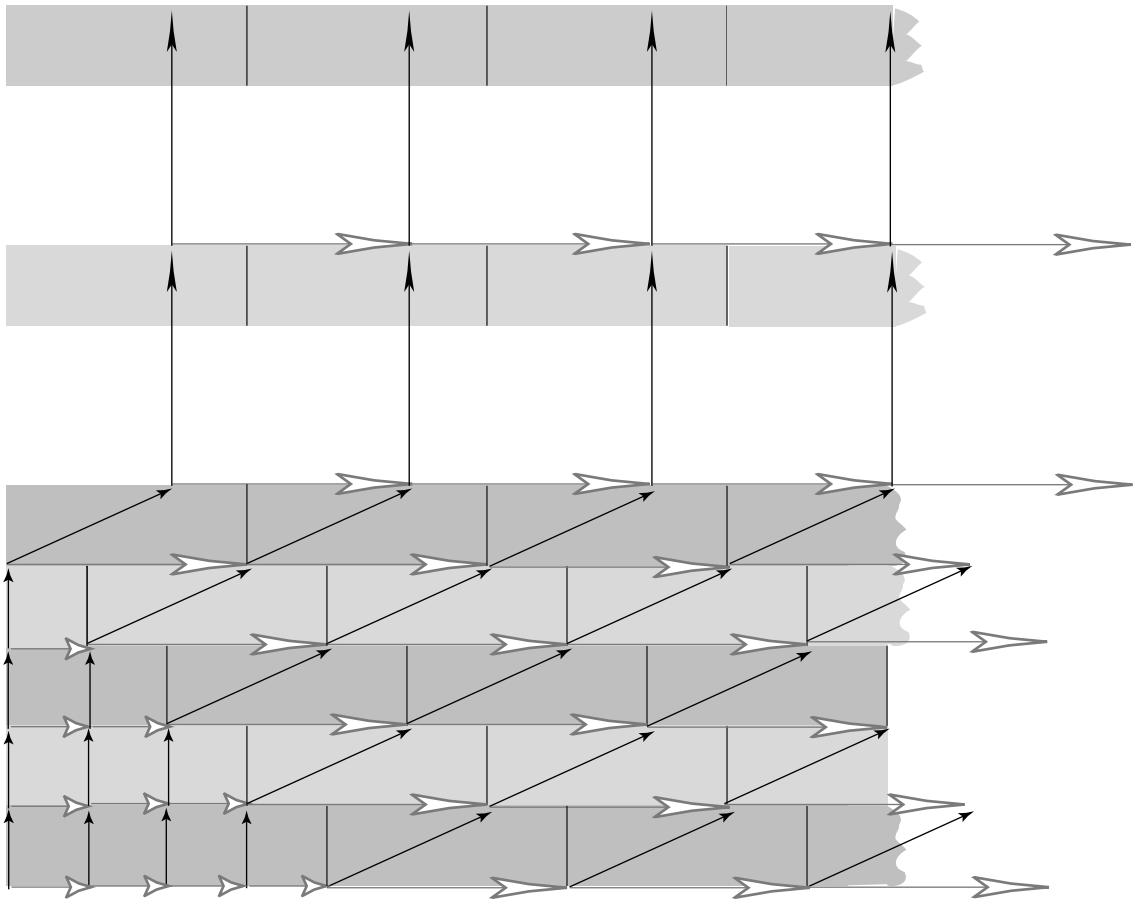


Fig A3 : $\mathbb{R}^2$-embedding of the s.t.d of the same geometrical c.a
(states are positioned by space and time vectors)

- 2 different greys are used to distinguish successive configurations, but they represent the same geometrical states
- white areas do not belong to any configuration

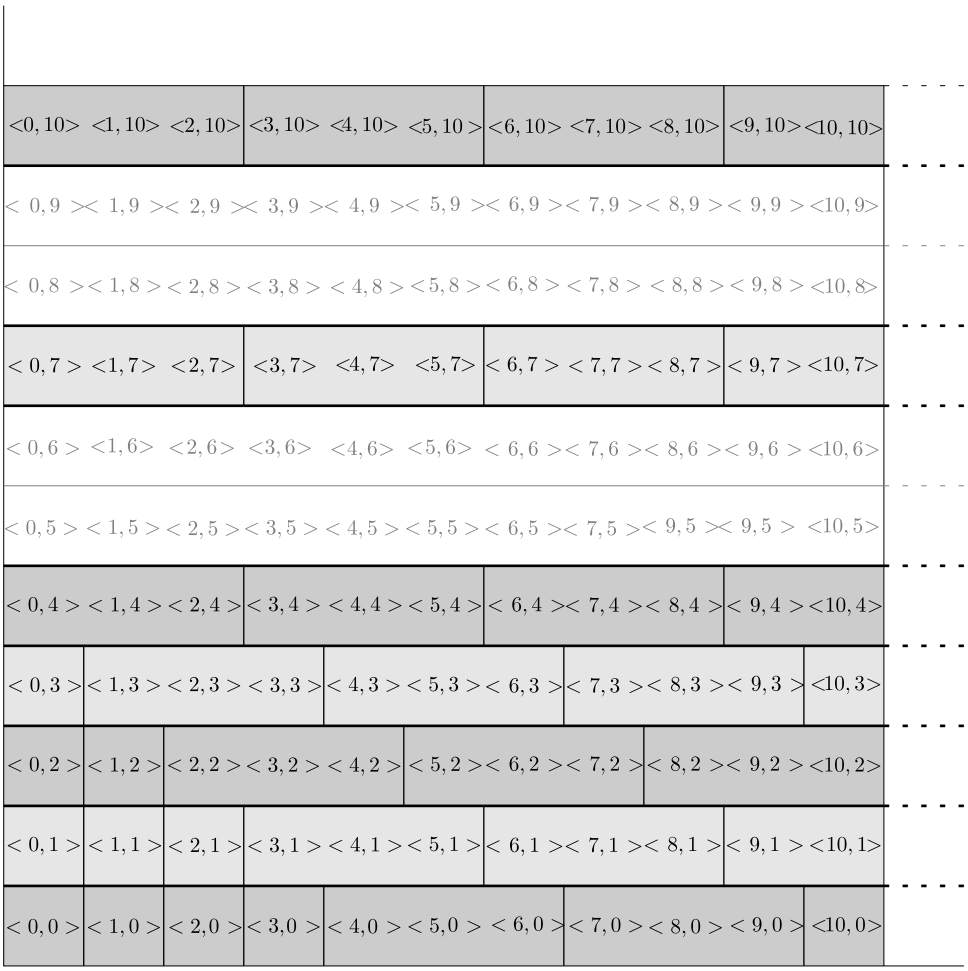


Fig A4 : original s.t.d read through the geometrical c.a (tensor product)

<2, 10> <1, 10> <0, 10>	< 5, 10 > <4, 10> <3, 10>	<8, 10> <7, 10> <6, 10>	<10, 10> <9, 10>					
<2, 7> <1, 7> < 0, 7 >	<5, 7> <4, 7> <3, 7>	< 8, 7 > < 7, 7 > < 6, 7 >	<10, 7> < 9, 7 >					
< 2, 4 > < 1, 4 > < 0, 4 >	< 5, 4 > < 4, 4 > < 3, 4 >	< 8, 4 > < 7, 4 > < 6, 4 >	<10, 4> < 9, 4 >					
< 0, 3 >	< 3, 3 > < 2, 3 > < 1, 3 >	< 6, 3 > < 5, 3 > < 4, 3 >	< 9, 3 > < 8, 3 > < 7, 3 >	<10, 3>				
< 0, 2 >	< 1, 2 >	< 4, 2 > < 3, 2 > < 2, 2 >	< 7, 2 > < 6, 2 > < 5, 2 >	<10, 2> < 9, 2 > < 8, 2 >				
< 0, 1 >	< 1, 1 >	< 2, 1 >	< 5, 1 > < 4, 1 > < 3, 1 >	< 8, 1 > < 7, 1 > < 6, 1 >	<10, 1> < 9, 1 >			
< 0, 0 >	< 1, 0 >	< 2,0 >	< 3,0 >	< 6, 0 > < 5,0 > < 4,0 >	< 9, 0 > < 8, 0 > < 7, 0 >	<10, 0>		

Fig A5 : how sites of the original s.t.d have been grouped

9.1.2 Second exemple : the square diagonal grouper

(with branches of length 2)

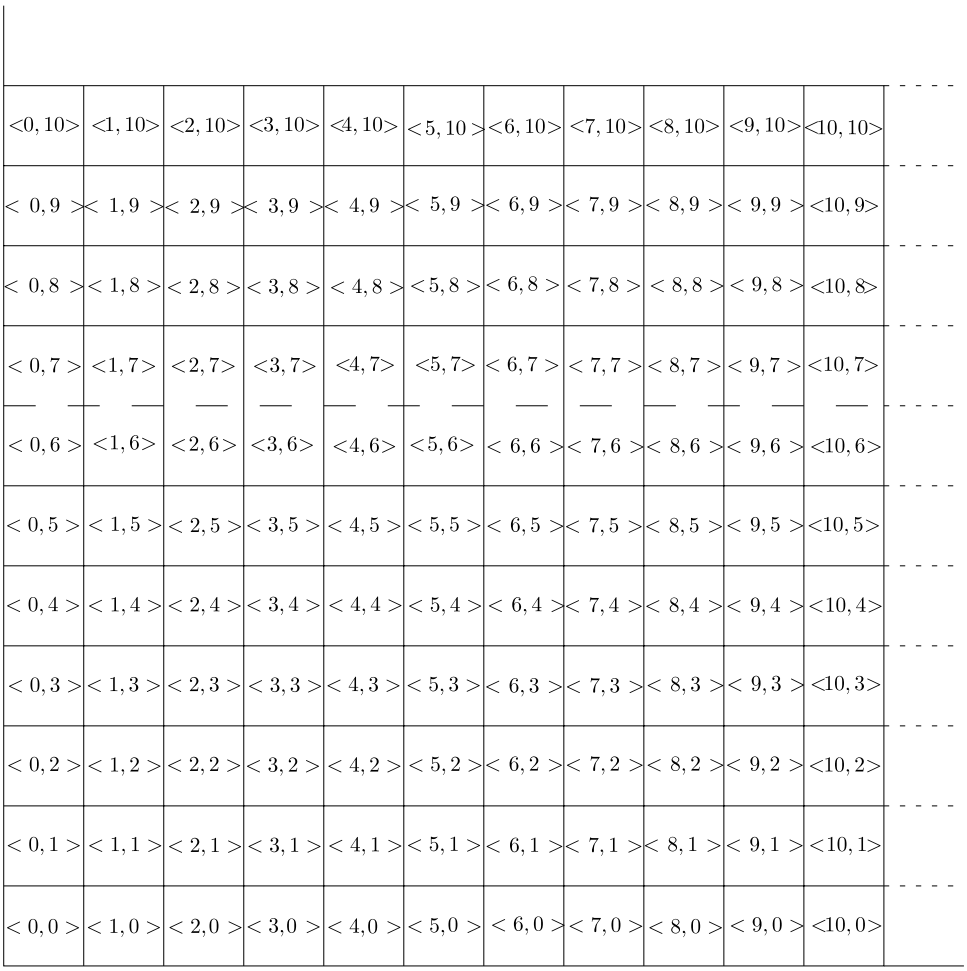


Fig B1 : s.t.d of any ordinary c.a

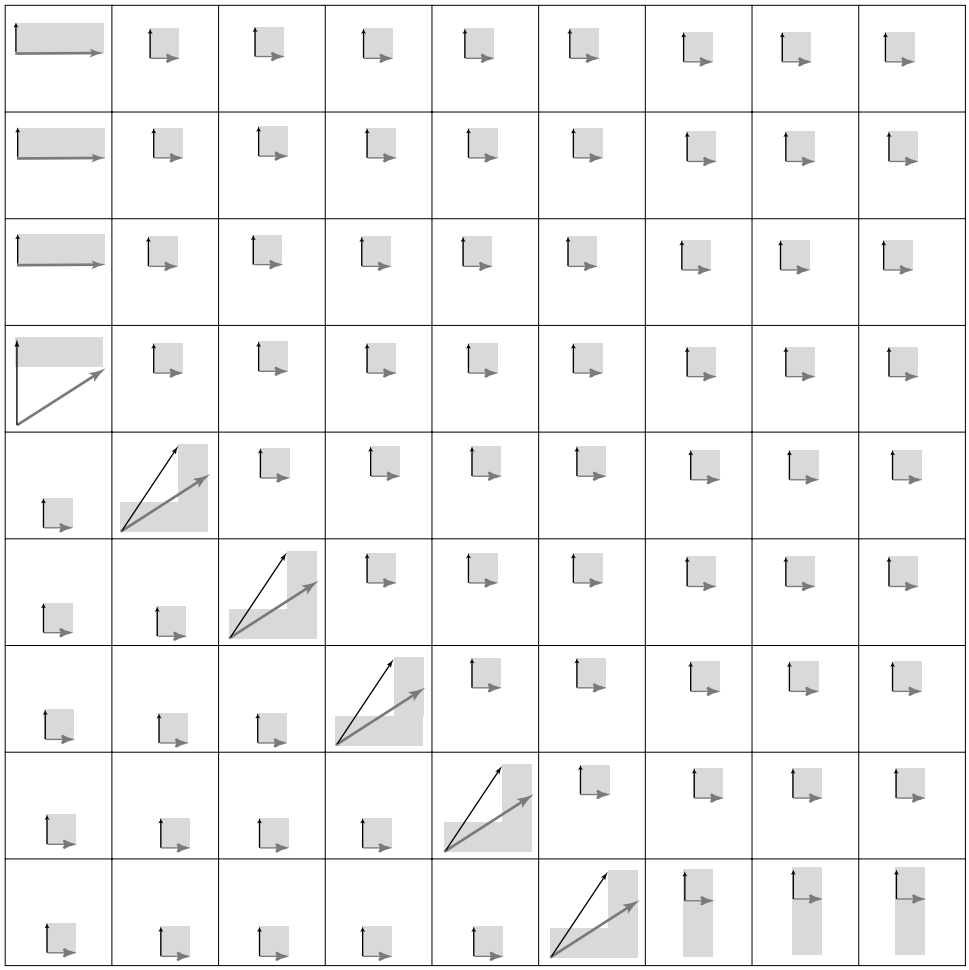


Fig B2 : s.t.d of a particular geometrical c.a (sqare diagonal)

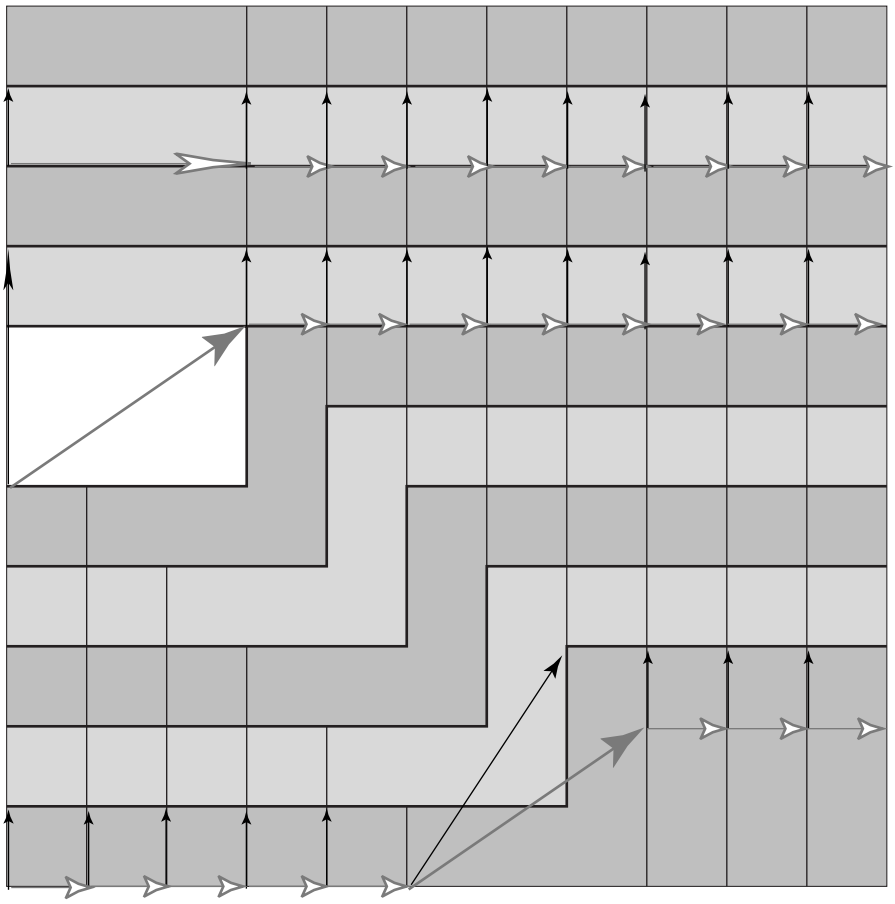


Fig B3 : $\mathbb{R}^2$-embedding of the s.t.d of the same geometrical c.a
(states are positioned by space and time vectors)

- 2 different greys are used to distinguish successive configurations, but they represent the same geometrical states
- white areas do not belong to any configuration

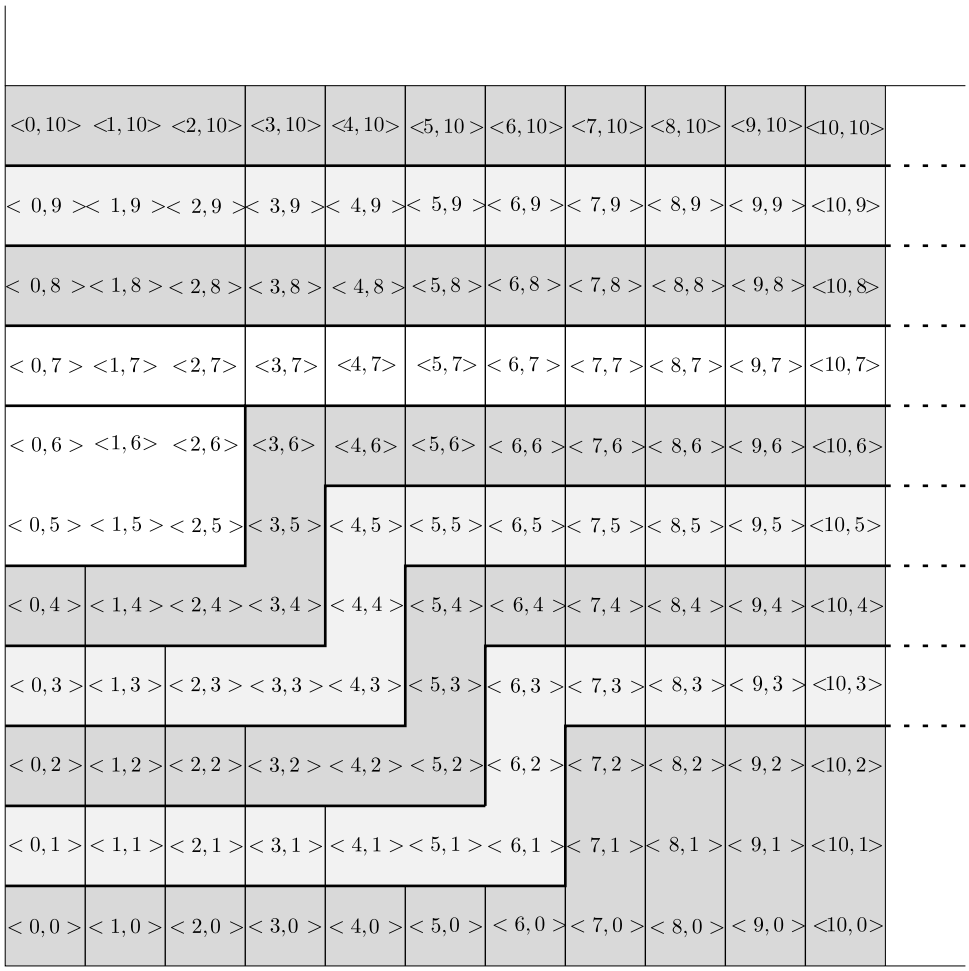


Fig B4 : original s.t.d read through the geometrical c.a (tensor product)

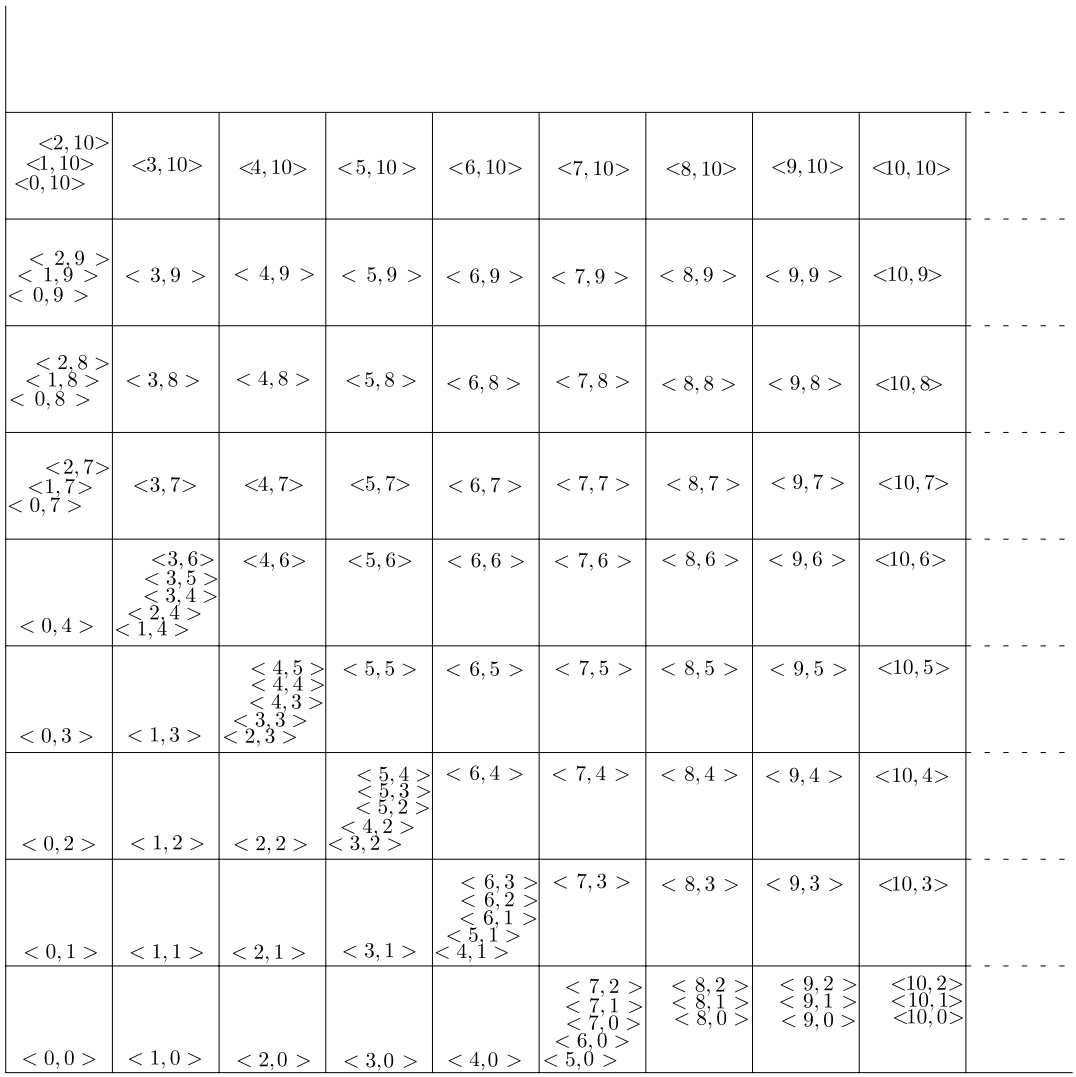


Fig B5 : how sites of the original s.t.d have been grouped

In the first example the grouping process is spatial, in the second it involves both space and time.

9.2 Definition of grouper c.a's, first formal approach

We shall now try to guide the reader from a close observation of figures to the progressive construction of the concept of geometrical and grouper c.a's as it has asserted itself as necessary to us.

Let us observe anew the first figure where we operated simultaneous grouping and computing, namely Figure 8b of chapter 7, which becomes Figure 0 of this new chapter. (This example is similar to the first part of 9.1.1). We draw next figure by repositioning all the sites of this figure at their original place, and putting a frame around the ones that are grouped. This gives Figure 1, which shows how the grouping-computing process has grouped sites of the original s.t.d. To sum up we could say : in Figure 0 sites of the original computing are to be seen in sites of the grouping-computing diagram, while in Figure 1 sites of the grouping-computing appear on those of the original computing diagram. Through this observation the *grouping-computing appears as a new way of reading the original s.t.d.* Sites of the grouping-computing form some kind of tiling which develops in a regular and mechanical manner. Therefrom the idea of c.a's producing geometrical configurations, made of little squares representing sites of an original regular tiling .

7	$< 2,7 >$ $< 1,7 >$ $< 0,7 >$	$< 5,7 >$ $< 4,7 >$ $< 3,7 >$	$< 8,7 >$ $< 7,7 >$ $< 6,7 >$	$< 11,7 >$ $< 10,7 >$ $< 9,7 >$	eee	eee	eee	eee
6	$< 0,6 >$	$< 3,6 >$ $< 2,6 >$ $< 1,6 >$	$< 6,6 >$ $< 5,6 >$ $< 4,6 >$	$< 9,6 >$ $< 8,6 >$ $< 7,6 >$	e e $< 10,6 >$	eee	eee	eee
5	$< 0,5 >$	$< 1,5 >$	$< 4,5 >$ $< 3,5 >$ $< 2,5 >$	$< 7,5 >$ $< 6,5 >$ $< 5,5 >$	e $< 9,5 >$ $< 8,5 >$	eee	eee	eee
4	$< 0,4 >$	$< 1,4 >$	$< 2,4 >$	$< 5,4 >$ $< 4,4 >$ $< 3,4 >$	$< 8,4 >$ $< 7,4 >$ $< 6,4 >$	eee	eee	eee
3	$< 0,3 >$	$< 1,3 >$	$< 2,3 >$	$< 3,3 >$	$< 6,3 >$ $< 5,3 >$ $< 4,3 >$	e e $< 7,3 >$	eee	eee
2	$< 0,2 >$	$< 1,2 >$	$< 2,2 >$	$< 3,2 >$	$< 4,2 >$	e $< 6,2 >$ $< 5,2 >$	eee	eee
1	$< 0,1 >$	$< 1,1 >$	$< 2,1 >$	$< 3,1 >$	$< 4,1 >$	$< 5,1 >$	eee	eee
0	$< 0,0 >$ $=$ q_0	E	E	E	E	E	E	E
	$\Uparrow$ x_0	$\Uparrow$ x_1	$\Uparrow$ x_2	$\Uparrow$ x_3	$\Uparrow$ x_4	$\Uparrow$ x_5		

Fig. 0

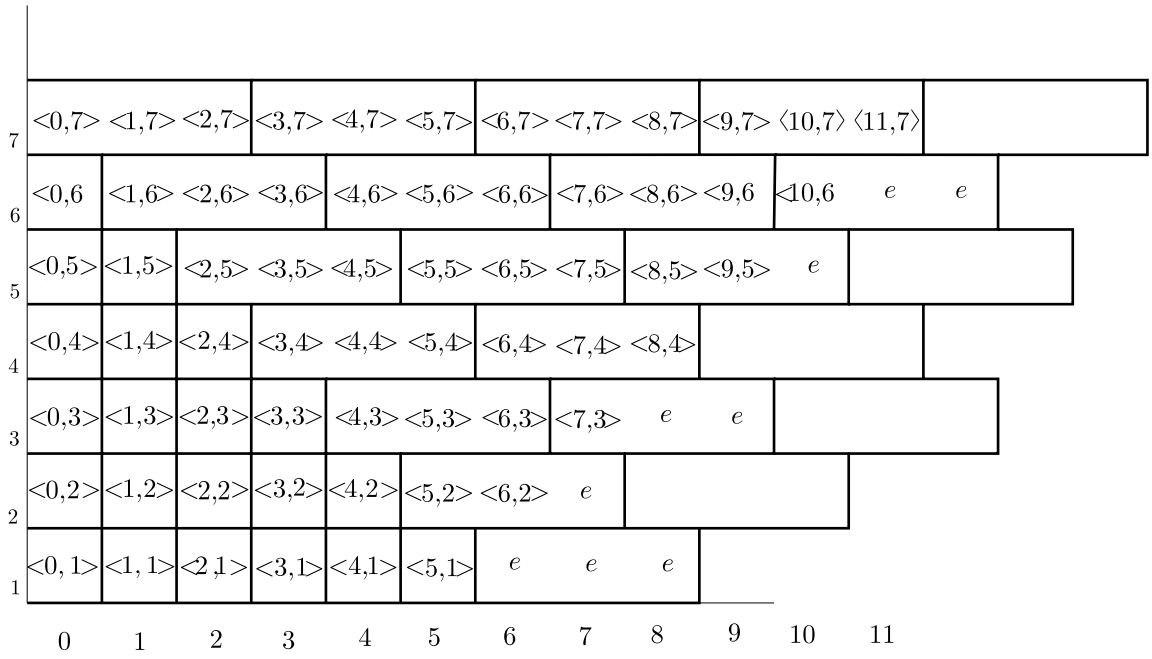


Fig.1

9.2.1 States of geometrical c.a's

These states will be geometrical pieces consisting of unit squares (satisfying a connexity condition that we shall express later on).

As these pieces must be aligned next to one another, we supply each one with an *origin point* and a *spacing vector* $\overrightarrow{V_s}$, at the end of which we may then place the origin of the next piece on the right.(We could have privileged spacing leftwards, but as we must choose, the usual reading direction is certainly the most comfortable). Thus, given a sequence of pieces, there is one and only one way to abutt each piece to the preceding one so as to form a "line". A configuration of pieces hangs on a string of vectors, like beads on a necklace.

Now, each piece with a left and a right neighbouring piece will produce at next time-step a new piece, which will take place above the (central) piece at the extremity of a second vector, the *timing-vector* $\overrightarrow{V_t}$.

In practice, the origin will most frequently be one of the vertices among the unit squares of which the piece is made of, but it may happen to be a vertex of an external unit square.

Of course, the two spacing and timing vectors have integer coordinates (unit naturally being the edge of the unit square).

In Figure 2 we give a few examples of pieces and the configuration they form if we align them.

The set of piece-states of the geometrical c.a will be denoted P.

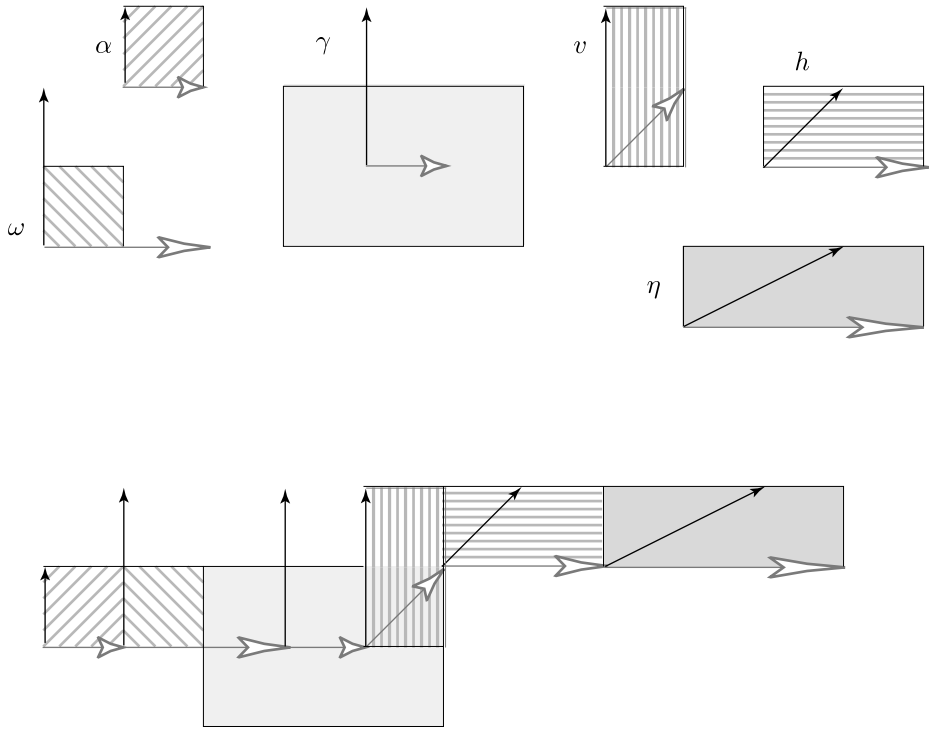


Fig.2 : configuration $\alpha\omega\gamma vh\eta$.

9.2.2 Transition rules

As in every c.a, the transition function is a mapping

$$d : P^3 \longrightarrow P.$$

Let us give a few simple examples :

1/ with α as only piece and transition $d(\alpha, \alpha, \alpha) = \alpha$ we obtain as s.t.d the basic tiling.

2/ with ω as only piece and transition $d(\omega, \omega, \omega) = \omega$ we obtain a s.t.d of spaced squares.

3/ with pieces α and η and transition

$$d(\alpha, \alpha, \alpha) = \alpha \qquad d(\alpha, \eta, \eta) = \eta$$

$$d(\alpha, \alpha, \eta) = \eta \qquad d(\eta, \eta, \eta) = \eta$$

and at time 0 configuration

$$\alpha\ \alpha\ \alpha\ \alpha\ \alpha\ \alpha\ \eta$$

we obtain as s.t.d the tiling of Figure 1.

9.2.3 Continuity condition

In Figure 3, we have represented the result of applying transition

$$d(\alpha, \alpha, \eta) = \alpha \qquad d(\alpha, \eta, \eta) = \eta$$

on configuration $\alpha\ \alpha\ \eta\ \eta$.

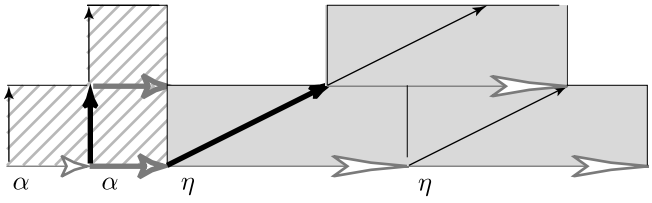


Fig.3

The pieces resulting from the two central pieces α and η are not correctly aligned, the one resulting from the η-piece being too far away to be a right neighbour for the one resulting from the α-piece. The defect is that the diagram formed by the following vectors

- spacing vector of piece α + timing vector of right neighbour η
- timing vector of piece α + spacing vector of piece produced at next time-step (which is also α)

is not commutative. There is a hole in the net formed by the spacing and timing vectors.

In order that this situation should never occur, transition must satisfy the following condition

Condition 9.2.1 *for any triple π_1, π_2, π_3 of pieces of P*

$$\overrightarrow{V_t}(\pi_2) + \overrightarrow{V_s}(d(\pi_1, \pi_2, \pi_3)) = \overrightarrow{V_s}(\pi_2) + \overrightarrow{V_t}(\pi_3).$$

With this condition we are guaranteed that as time goes on, a sequence of correctly aligned geometrical configurations develops.

9.2.4 Computability condition

Next conditions are fundamental to compose geometrical c.a's with usual c.a's. With these conditions, the *geometrical c.a* we have just defined becomes what we shall call a *grouper c.a.*

Indeed, we must keep in sight what use we intend to make of the preceding geometrical c.a : the sequence of geometrical configurations that it develops is meant to help us to a new reading of the s.t.d of some ordinary c.a, represented in the basic tiling.

In Figure 4 we represent transition

$$d(\alpha, \alpha, v) = v$$

and a portion of s.t.d read according to the grid formed by these pieces : what we notice here is that the states of the original basic sites lying in the piece resulting from transition d cannot be computed (using transition δ of the original c.a) from the sites of the three pieces to which d applies. This defect withdraws any interest the grouper c.a could have, so we mean to prevent it.

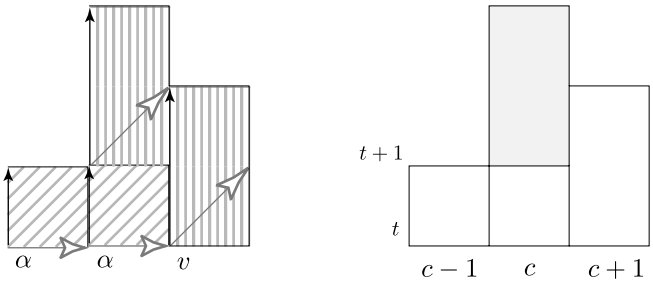


Fig.4

For this we shall introduce a definition.

If S is a set of sites of some ordinary s.t.d, we shall denote $csq(S)$ (csq stands for consequence), the set of sites resulting from sites of S through one transition exactly, and CSQ the transitive closure of S by this mapping :

$$CSQ(S) = \cup_{i \geq 0} (csq)^i(S) = S \cup csq(S) \cup \ldots$$

(two examples are shown in Figure 5).

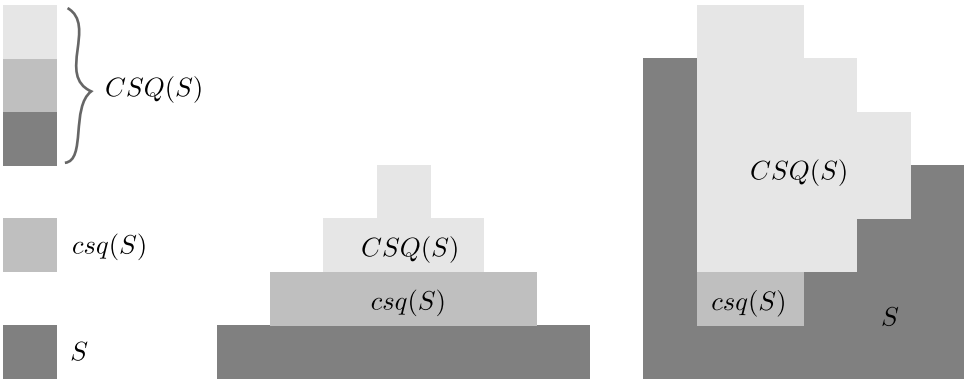


Fig.5

Our request concerning d will be the following

Condition 9.2.2 (computability) *for any triple of pieces* (π_1, π_2, π_3) *of* P

$$d(\pi_1, \pi_2, \pi_3) \subseteq CSQ(\pi_1 \cup \pi_2 \cup \pi_3).$$

Care : the above inclusion is a bit speedy. The notation is abusive, it uses an implicit positioning of pieces in the $\mathbb{R}^2$ plane, formally it should be written

$$\overrightarrow{V_t}(\pi_2) + d(\pi_1, \pi_2, \pi_3) \subseteq CSQ((-\overrightarrow{V_s}(\pi_1) + \pi_1) \cup \pi_2 \cup (\overrightarrow{V_s}(\pi_2) + \pi_3),$$

where π is identified with the subset of $\mathbb{R}^2$ obtained by positioning π in $\mathbb{R}^2$ so that its origin coincides with $(0, 0)$, and $\overrightarrow{V} + \pi$ is obtained via $\overrightarrow{V}$-translation.

9.2.5 Covering condition

At first we have the idea that all sites of the s.t.d of the original c.a should be covered by the tiling developed by the grouper c.a. Or perhaps rather that each cell of the original c.a should be present in each configuration of the grouper c.a. In fact our request will be neither of these two : the first one is not necessary (see Figures A3, A4), and the second is not sufficient (see middle example of Figure 6). It will be a third one, which is that no configuration can have a hole in it : indeed, because of the computability condition, this hole would grow larger and larger with time (Figure 6). Let us notice that, on the contrary, overlappings are no problem.

But as soon as we have worded this condition, it appears clearly to us that we have implicitly supposed that the pieces themselves had no holes, as such a hole would indeed be a hole in the configuration containing this piece !

We can express our condition as follows :

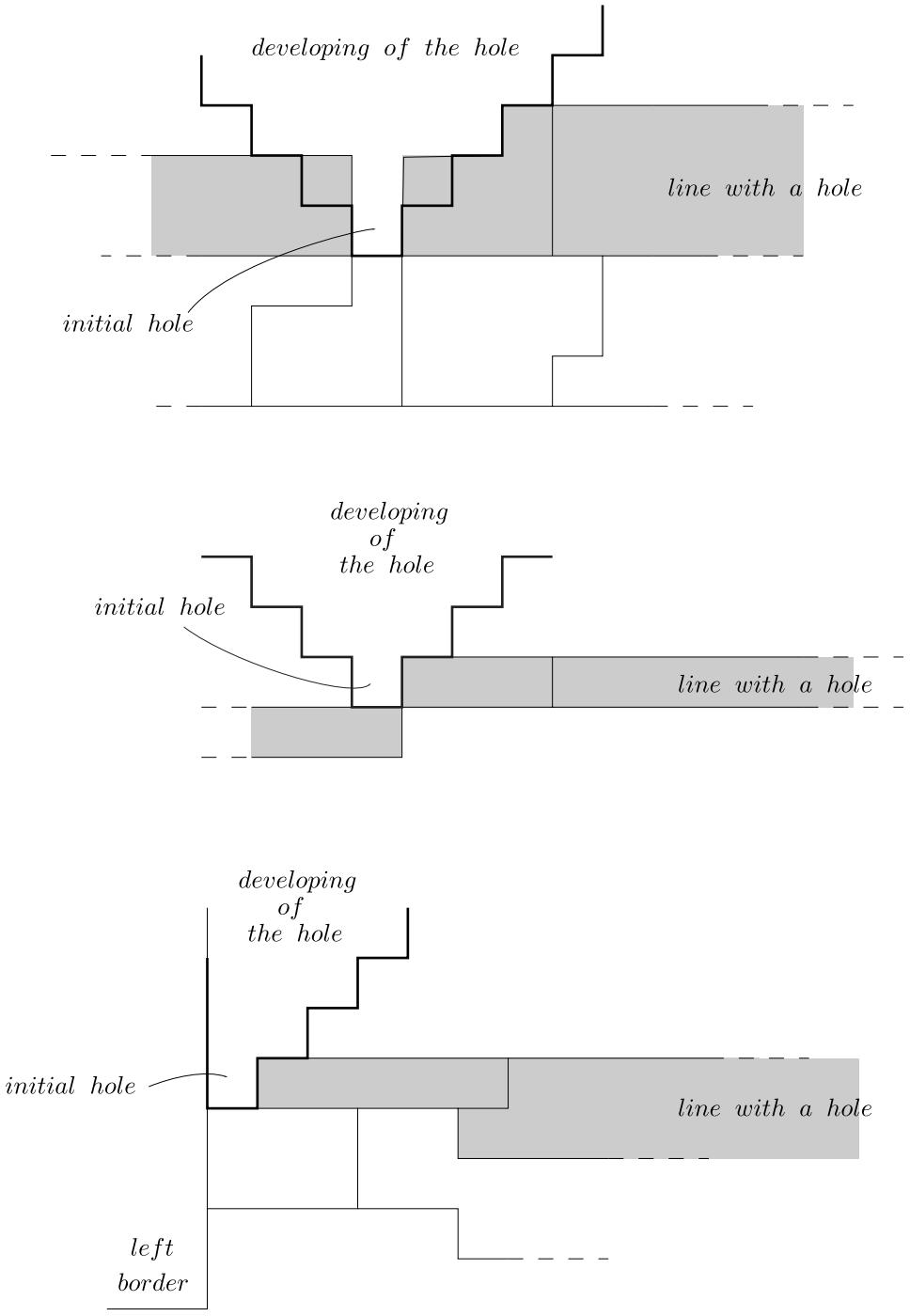

developing of the hole
line with a hole
initial hole
developing
of
the hole
initial hole
line with a hole
developing
of
the hole
initial hole
line with a hole
left
border

Fig.6.

Condition 9.2.3 (contact-connexity) *1)contact-connexity of each configuration : two neighbouring pieces must lean against one another by at least one unit side, be it vertical or horizontal, unless they overlap.*

2)contact-connexity of each piece : for any two basic squares s, s' of a piece, there exists a finite sequence $s = s_1, \ldots, s_n = s'$ of basic squares of the piece such that for all i, s_i and s_{i+1} must lean against one another by one side

2) is the connexity condition that we had evoked in defining the pieces.

Condition 9.2.4 (covering) *any configuration is contact-connex*

In all the examples that we shall present the configurations are very simple, with very few pieces, they reproduce themselves with time, and this condition is evidently satisfied.

If we want to formalize preservation of contact-connexity of configurations through transition, it could be written

$\overrightarrow{V_t}(\pi_2) + d(\pi_1, \pi_2, \pi_3)$ and $\overrightarrow{V_s}(\pi_2) + \overrightarrow{V_t}(\pi_3) + d(\pi_2, \pi_3, \pi_4)$ are contact-connex

this for all quadruples $(\pi_1, \pi_2, \pi_3, \pi_4)$ to be found in the configurations.

9.2.6 Borders

If our grouper c.a should apply to some s.t.d limited by one or two borders, it is clear that there is no use for it to develop its geometry beyond these borders. Accordingly, we shall define two special piece-states that are not real ones (as the border states were no real states), LB and RB. These pieces will be in the shape of quadrants of space (see figures 6, 8a, 9a ...) for which we do not precise either origin nor vectors, and they are constant pieces.

There is no continuity condition for LB and RB. The computability condition writes normally

$$d(LB, \pi, \pi_r) \subset CSQ(LB \cup \pi \cup \pi_r)$$

$$d(\pi_l, \pi, RB) \subset CSQ(\pi_l \cup \pi \cup RB).$$

Covering condition only requests that pieces neighbouring a border must lean against this border by one unit side at least, unless they overlap the border.

9.2.7 Inputs

We delay to 9.5 the study of inputs with grouper c.a's, as it could be a little confusing. We only mention now that we foresee only parallel input.

As for now we look at the functioning of geometrical c.a's starting from an initial configuration. To be coherent with section 3.3 where we introduced parallel input, which was meant to set up an initial configuration, our initial configuration for grouper c.a's will be in place at time 1, at the same time as it was set up for ordinary c.a's.

9.2.8 Product $\mathcal{R} \otimes \mathcal{A}$ of a grouper c.a and a c.a

Let $\mathcal{R}(P, d)$ be a grouper c.a and $\mathcal{A}(Q, \delta)$ an ordinary c.a. We define the tensor product $\mathcal{R} \otimes \mathcal{A}$ (which is to be distinguished it from the ordinary product of ordinary c.a's) very naturally as follows :

- its states are the pieces of $\mathcal{R}$ with on each of their unit squares a state of $\mathcal{A}$. To each piece formed of k squares corresponds at most $|Q|^k$ states, so the number of states of the product is a polynomial in $|Q|$, the degree of which is the maximum area of a piece of $\mathcal{R}$.

  We shall denote such a state by listing, after the name of the geometrical piece, the states appearing on it, in the order of usual reading (left to right and topdown)

$$\pi(q_1, q_2, \dots, q_k)$$

  and if all the states are the same state q, we shall abbreviate by $\pi(q)$.

- its transition Δ is the transition for the pieces, completed by the computation (by the transition δ of $\mathcal{A}$) of all the states of the resulting piece. This computation is always possible if the computability condition 9.2.2 is satisfied.

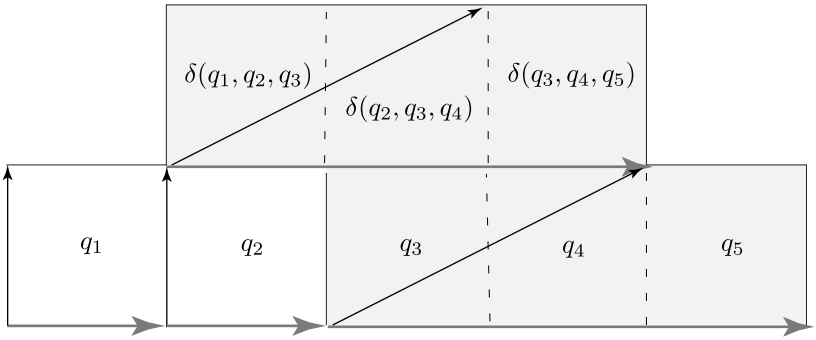


Fig.7.

- its initial configuration is the initial configuration of $\mathcal{R}$ with, on each unit square of the piece-states of this configuration, the corresponding state of the s.t.d of $\mathcal{A}$.

We have just formally described the rules of the tensor product. But we shall never write them. In practice, nothing is easier than obtaining the s.t.d of the tensor product, we just superimpose the s.t.d of the grouper on the s.t.d of the c.a !

9.3 Examples and applications

9.3.1 The horizontal k-grouper

- **Functioning** :

  pieces : α , η (see Figure 8a, where $k = 3$)

  rules :

$$\begin{array}{cccccccccccccc} LB & \alpha & \alpha & & \alpha & \alpha & \alpha & & \alpha & \alpha & \eta & & \alpha & \eta & \eta & & \eta & \eta & \eta \\ & \alpha & & & & \alpha & & & & \eta & & & & \eta & & & & \eta & \end{array}$$

  evolution of configuration : $LB\ \alpha\ \ldots\ \alpha\ \eta\ \eta\ \ldots$

$$\begin{array}{ccccccccc} LB & \alpha & \alpha & \alpha & \alpha & \eta & \eta & \ldots \\ LB & \alpha & \alpha & \alpha & \eta & \eta & \eta & \ldots \\ LB & \alpha & \alpha & \eta & \eta & \eta & \eta & \ldots \\ LB & \alpha & \eta & \eta & \eta & \eta & \eta & \ldots \\ LB & \eta & \eta & \eta & \eta & \eta & \eta & \ldots \end{array}$$

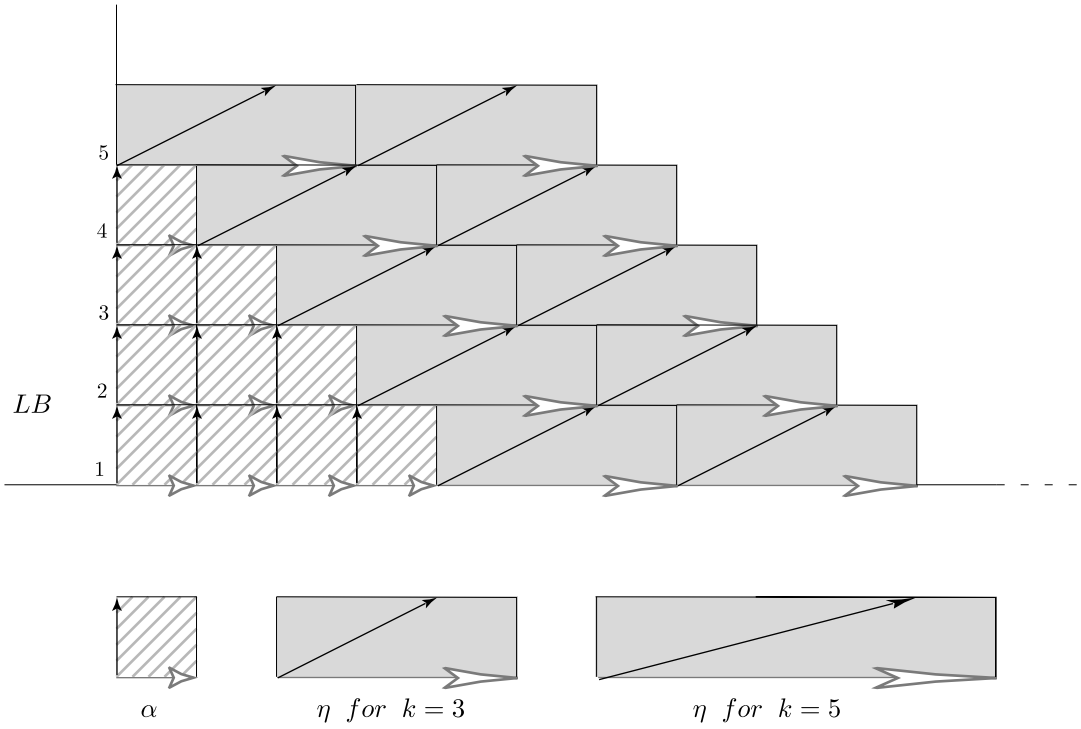


Fig.8a.

- **Application** :

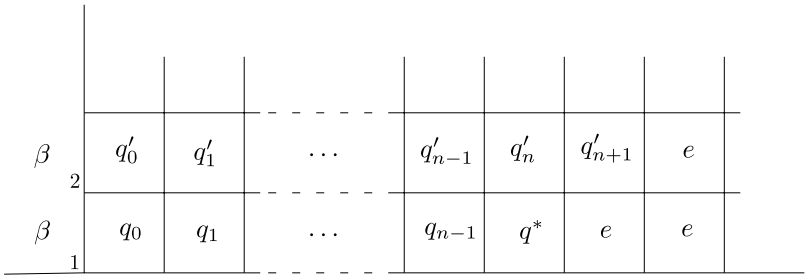


Fig.8b.

We shall now do the product of preceding grouper $\mathcal{R}$, with some c.a $\mathcal{A}$, whose s.t.d is represented in Figure 8b. If the initial configuration of c.a $\mathcal{A}$ has n states q_i and an end state $q*$, the grouper will have an initial configuration with n states α.

In Figure 8c we find the s.t.d of the tensor-product : with the rules as we have defined them, it is clear that this s.t.d may as well be obtained by simply superposing the geometrical s.t.d to the s.t.d of $\mathcal{A}$.

The η-pieces will reach the left border at time $n+1$. If at this moment we could suddenly change transition of the grouper for

$$d'(LB,\eta,\eta) = H \quad d'(\eta,\eta,\eta) = H \quad d'(H,H,H) = H$$

(Figure 8c) the product would start doing k-accelerated computation.

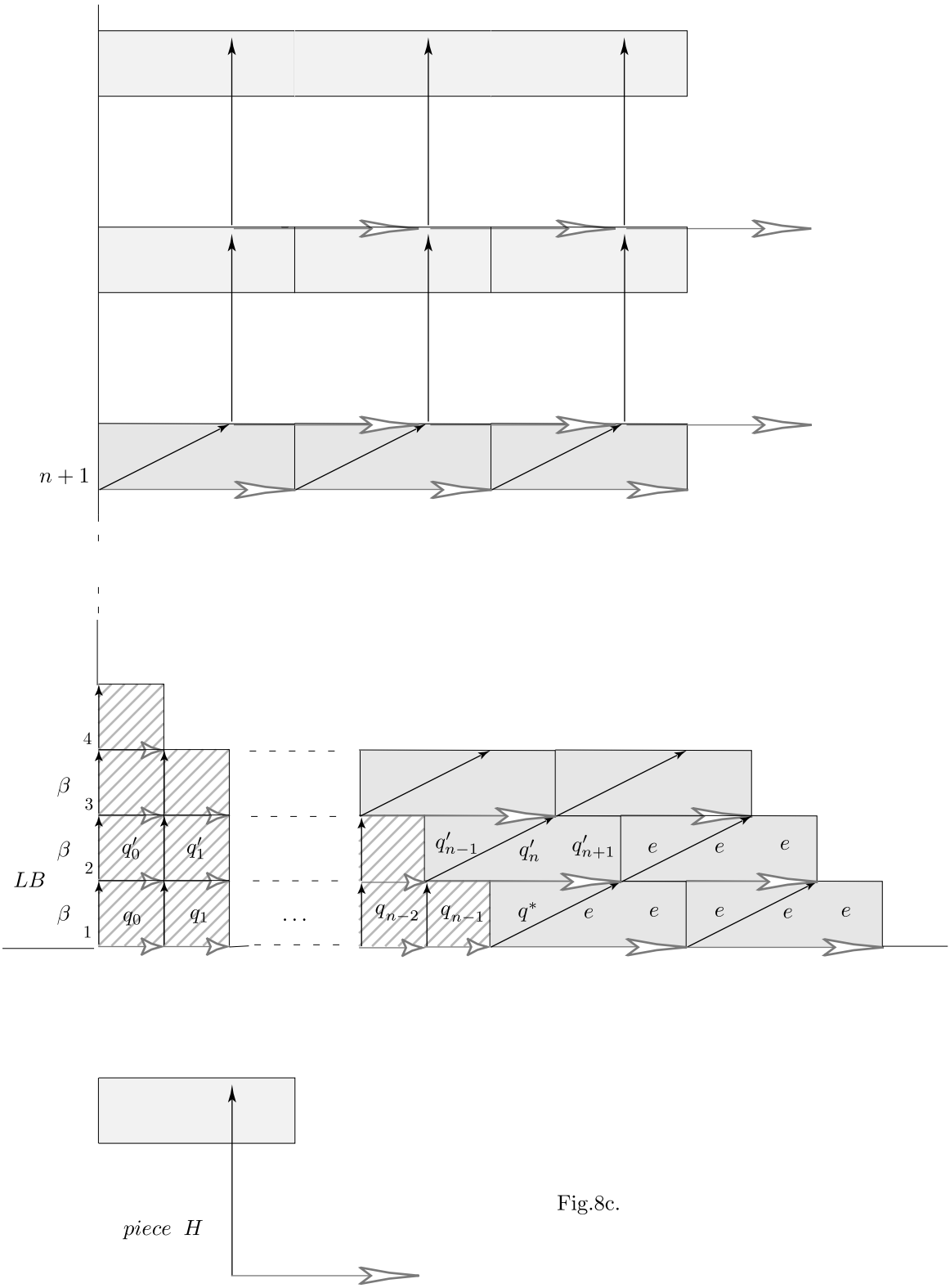


Fig.8c.

Well, if we complete the c.a $\mathcal{A}$ with a minimal time 2e-synchronizer starting at time 2, we can decide that the product $\mathcal{R}\otimes\mathcal{A}$ starts the preceding geometrical transition when fire state appears on its pieces, something that the grouper c.a by itself could not do. Then we have obtained, presented differently, in a geometrical manner, the strong speeding up of the parallel c.a $\mathcal{A}$ of chapter 7 (Figure 8c).

The number of states of $\mathcal{R}\otimes\mathcal{A}$ is easy to count : if s is the number of states of $\mathcal{A}$, it is $2s^k + s$, at most, because all states are not necessarily used.

Note 9.3.1 *we may consider that here we have done two successive tensor products, with two different groupers. We may also introduce some non-determinism in the groupers, with the choice between rules determined by the sites of the ordinary c.a. A much more complicated and cumbersome solution would be to incorporate states of the minimal-time synchronizer in the pieces, as colors put on them, but this would destroy the remarkable simplicity of the groupers that we have considered all along.*

9.3.2 The square diagonal grouper

We shall present the simplest case, when the two branches of the square have length 1. The general case is an easy adaptation

- **Functioning** :

  pieces : α , τ , a , c , λ (see Figure 9a)

  rules :

$$\begin{array}{cccc} LB & \alpha & \alpha & \\ & \alpha & & \end{array} \qquad \begin{array}{cccc} & \alpha & \alpha & \alpha \\ & & \alpha & \end{array} \qquad \begin{array}{cccc} & a & a & a \\ & & \alpha & \end{array} \qquad \begin{array}{cccc} & \alpha & \alpha & \tau \\ & & \tau & \end{array}$$

$$\begin{array}{cccc} \alpha & \tau & a & \\ & \alpha & & \end{array} \qquad \begin{array}{cccc} & \alpha & \tau & \alpha \\ & & \alpha & \end{array} \qquad \begin{array}{cccc} & \tau & a & a \\ & & \alpha & \end{array} \qquad \begin{array}{cccc} & \tau & \alpha & \alpha \\ & & \alpha & \end{array}$$

$$\begin{array}{cccc} LB & \alpha & \tau & \\ & c & & \end{array} \qquad \begin{array}{cccc} LB & c & \alpha & \\ & \lambda & & \end{array} \qquad \begin{array}{cccc} LB & \lambda & \alpha & \\ & \lambda & & \end{array}$$

  evolution of a configuration : $LB\ \alpha\ \alpha\ \alpha\ \alpha\ \tau\ a\ a\ \ldots$ (Figure 9a)

$$\begin{array}{cccccccc} LB & \alpha & \alpha & \alpha & \alpha & \tau & a & \ldots \\ LB & \alpha & \alpha & \alpha & \tau & \alpha & \alpha & \ldots \\ LB & \alpha & \alpha & \tau & \alpha & \alpha & \alpha & \ldots \\ LB & \alpha & \tau & \alpha & \alpha & \alpha & \alpha & \ldots \\ LB & c & \alpha & \alpha & \alpha & \alpha & \alpha & \ldots \\ LB & \lambda & \alpha & \alpha & \alpha & \alpha & \alpha & \ldots \\ LB & \lambda & \alpha & \alpha & \alpha & \alpha & \alpha & \ldots \end{array}$$

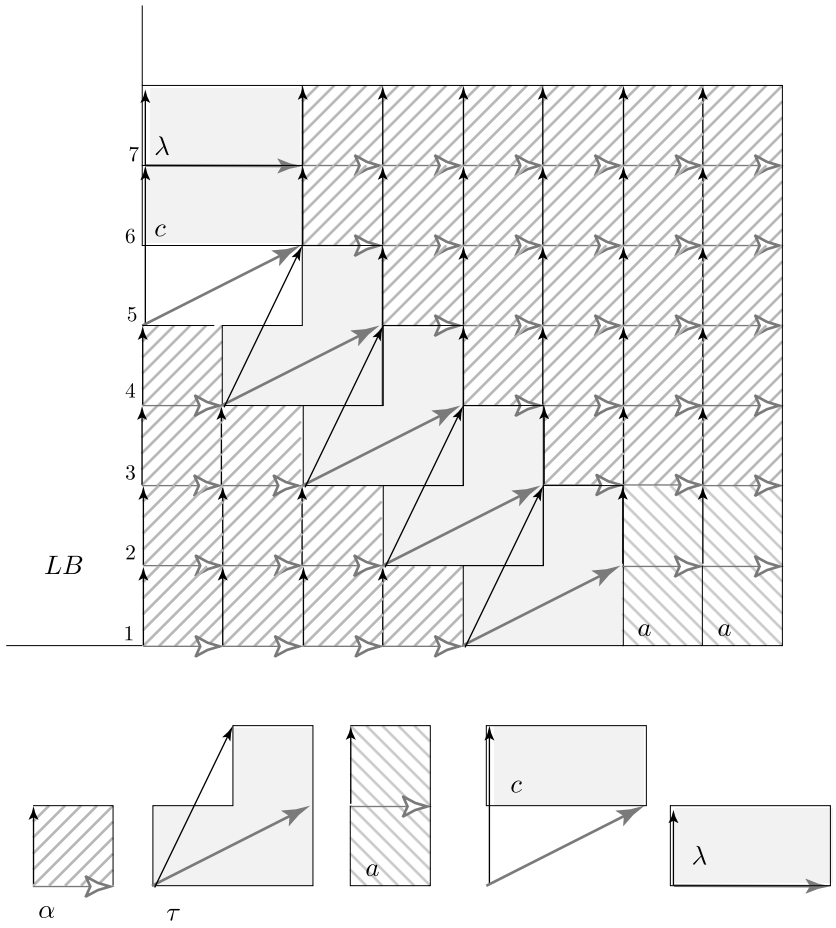


Fig.9a.

- **Application** : acceleration of one time-step of a parallel recognizer (figures 9b and c).

  To obtain the s.t.d of the tensor-product $\mathcal{P}$ of this grouper with a c.a $\mathcal{A}$, as we did in previous example, we superimpose the geometrical s.t.d on the s.t.d of $\mathcal{A}$, represented in Figure 9b (for $n = 4$). We are left to define the halting states of c.a $\mathcal{P}$: they will be the pieces λ (or c) which carry as first state a halting state of $\mathcal{A}$.

  If the recognizing time for an input word of length n by $\mathcal{A}$ was $T(n) = (n+1) + D(n)$ with $D(n) \geq 1$, the recognizing time by $\mathcal{P}$ will be $T'(n) = (n+1) + D(n) - 1$. If $T(n) = n+1$, then $T'(n) = n+1$ (we must add to $\mathcal{A}$ a "post-halting" state with which we shall define a halting c-state). So the general formula for $D(n) \geq 0$ is

$$T'(n) = (n+1) + max(0, D(n) - 1).$$

If s is the number of states of $\mathcal{A}$, the number of states of $\mathcal{B}$ is $s^3 + 3s^2 + s$.

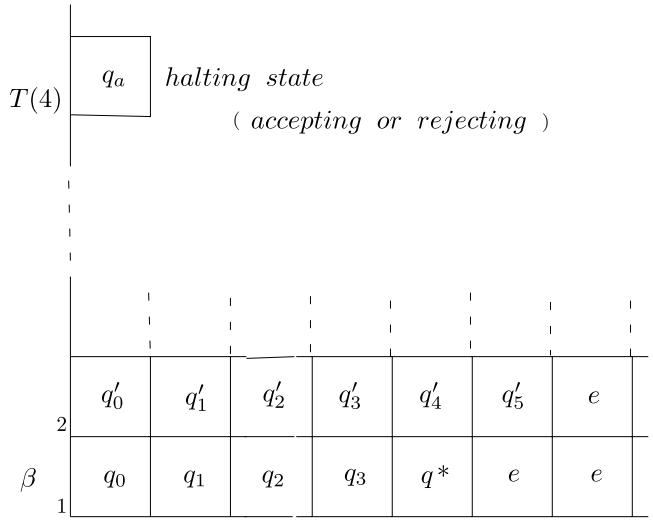


Fig.9b.

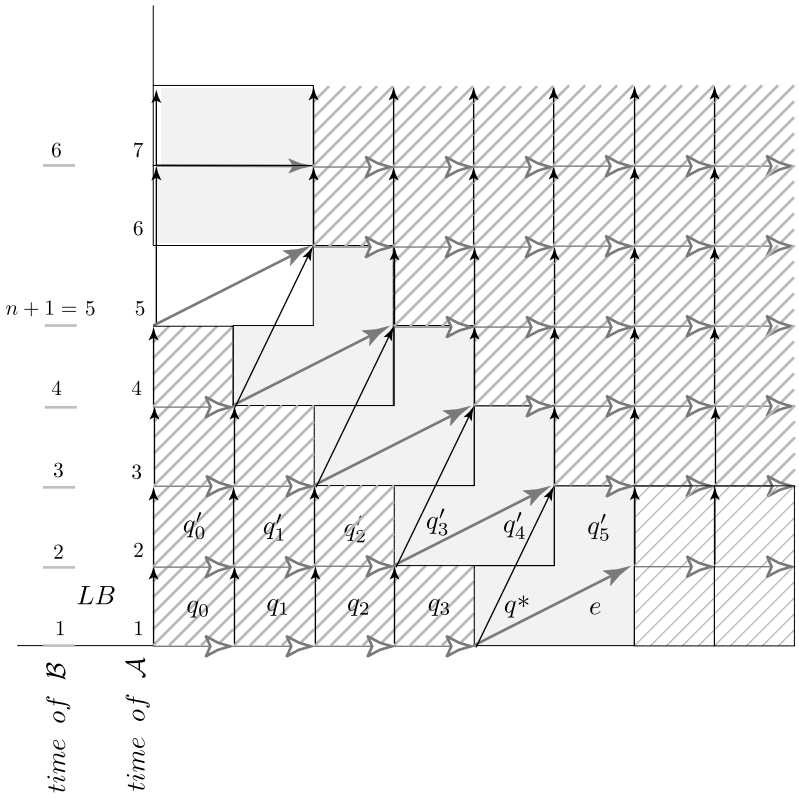


Fig.9c. The s.t.d of $\mathcal{R} \otimes \mathcal{A}$ represented in the space-time of $\mathcal{A}$

- **Generalisation** : accelerating of h time-steps of a parallel recognizer.

The two branches of the square now have length $h \geq 1$. Pieces τ, c and λ all have length $h+1$, and the spacing and timing vectors are as represented in Figure 10.

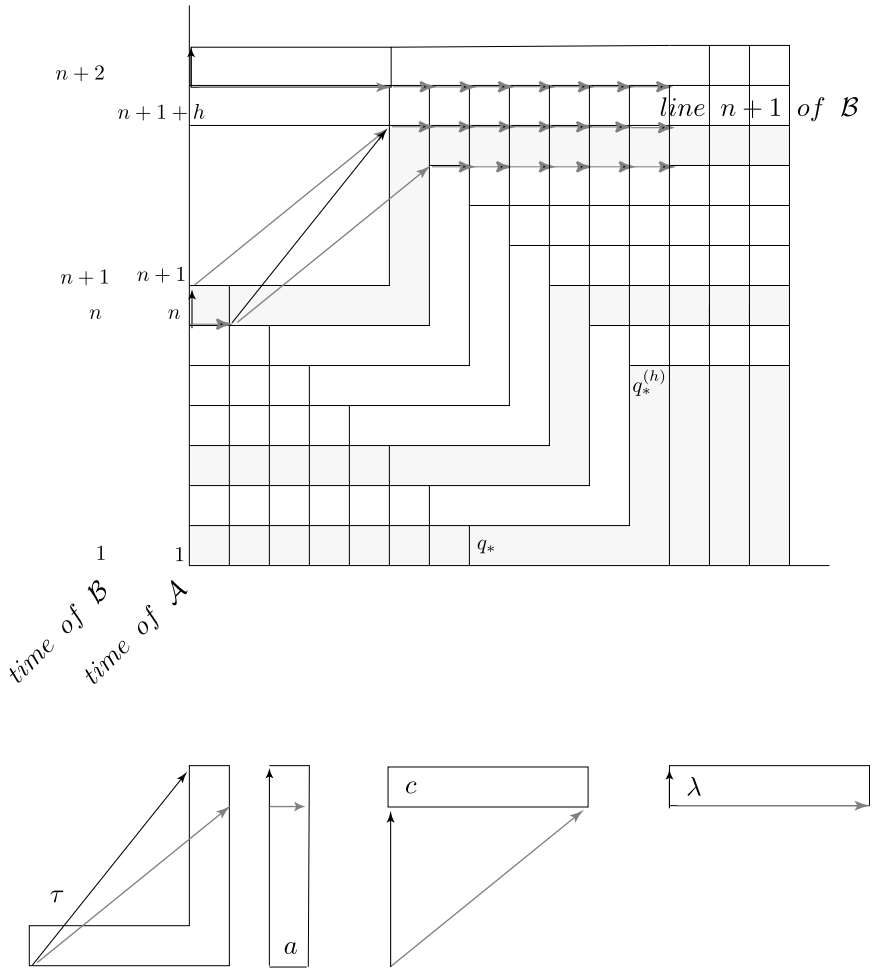


Fig.10.

Recognizing time will be :

$$T'(n) = n + 1 + max(0, D(n) - h)$$

(If necessary s.t.d of $\mathcal{A}$ is extended a little after halting).

We can notice here a slight improvement of the formula of chapter 7 (7.5), because of cases when $1 \leq D(n) \leq h$.

But the main thing, really worth noticing and admiring, is that **no auxiliary synchronizer has been used**. Speeding up is then much simpler, with much lesser states : their number is $s^{2h+1} + 3s^{h+1} + s$, at most.

9.3.3 The broken sticks diagonal grouper

- **Functioning** :

  pieces : α, η, λ, u, v, c (Figure 11, where k=3)

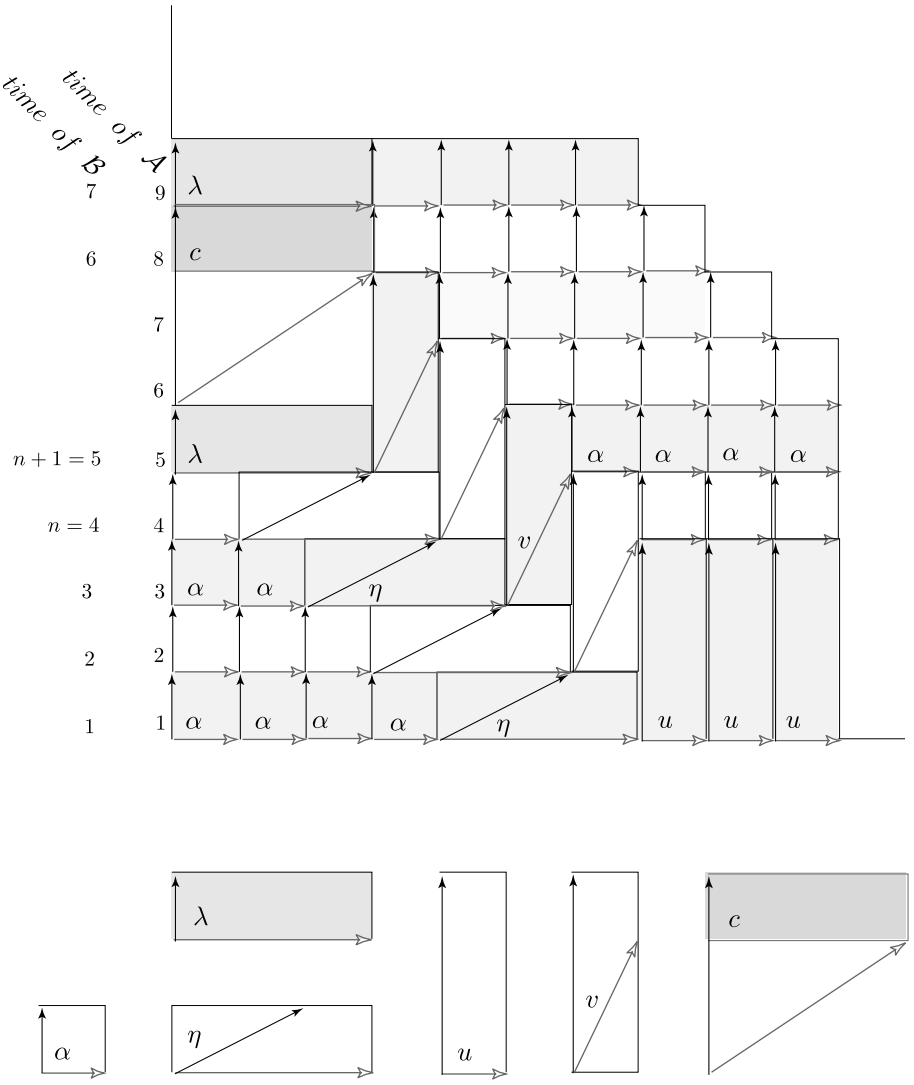


Fig.11.

rules:

$$\begin{array}{cccccccccccc} LB & \alpha & \alpha & \alpha & \alpha & \alpha & u & u & u & v & \alpha & \alpha & c & \alpha & \alpha \\ & \alpha & & & \alpha & & & \alpha & & & \alpha & & & \alpha & \end{array}$$

$$\begin{array}{cccccccccccc} \alpha & \alpha & \eta & \alpha & \eta & u & \alpha & \eta & v & \eta & u & u & \eta & v & \alpha \\ & \eta & & & v & & & v & & & \alpha & & & \alpha & \end{array}$$

$$\begin{array}{cccccccccccc} \lambda & v & \alpha & LB & \alpha & \eta & LB & \lambda & v & LB & c & \alpha & LB & \lambda & \alpha \\ & \alpha & & & \lambda & & & c & & & \lambda & & & \lambda & \end{array}$$

evolution of a configuration $LB\ \alpha\ \alpha\ \alpha\ \eta\ u\ u\ u$

$$\begin{array}{cccccccc} LB & \alpha & \alpha & \alpha & \eta & u & u & u & \dots \\ LB & \alpha & \alpha & \eta & v & \alpha & \alpha & \alpha & \dots \\ LB & \alpha & \eta & v & \alpha & \alpha & \alpha & \alpha & \dots \\ LB & \lambda & v & \alpha & \alpha & \alpha & \alpha & \alpha & \dots \\ LB & c & \alpha & \alpha & \alpha & \alpha & \alpha & \alpha & \dots \\ LB & \lambda & \alpha & \alpha & \alpha & \alpha & \alpha & \alpha & \dots \end{array}$$

- **Application** : speeding up a parallel recognizer by h time-steps.

  We want to do the product of the preceding grouper having pieces of length $k = h + 1$ and recognizer $\mathcal{A}$. We proceed always in the same way.

  Recognizing time by $\mathcal{P}$ is the same as before

  $$T'(n) = n + 1 + max(0, D(n) - h).$$

  As preceding accelerated c.a, this one uses **no auxiliary synchronizer**. It is a better one still, because it is simpler, it has smaller pieces, and so a number of states of lesser degree : $5s^{h+1} + s$

9.3.4 The broken sticks grouper

The s.t.d of this new product is represented in Figure 12 (where $k = 3$ and two shades of grey have been used to distinguish successive configurations).

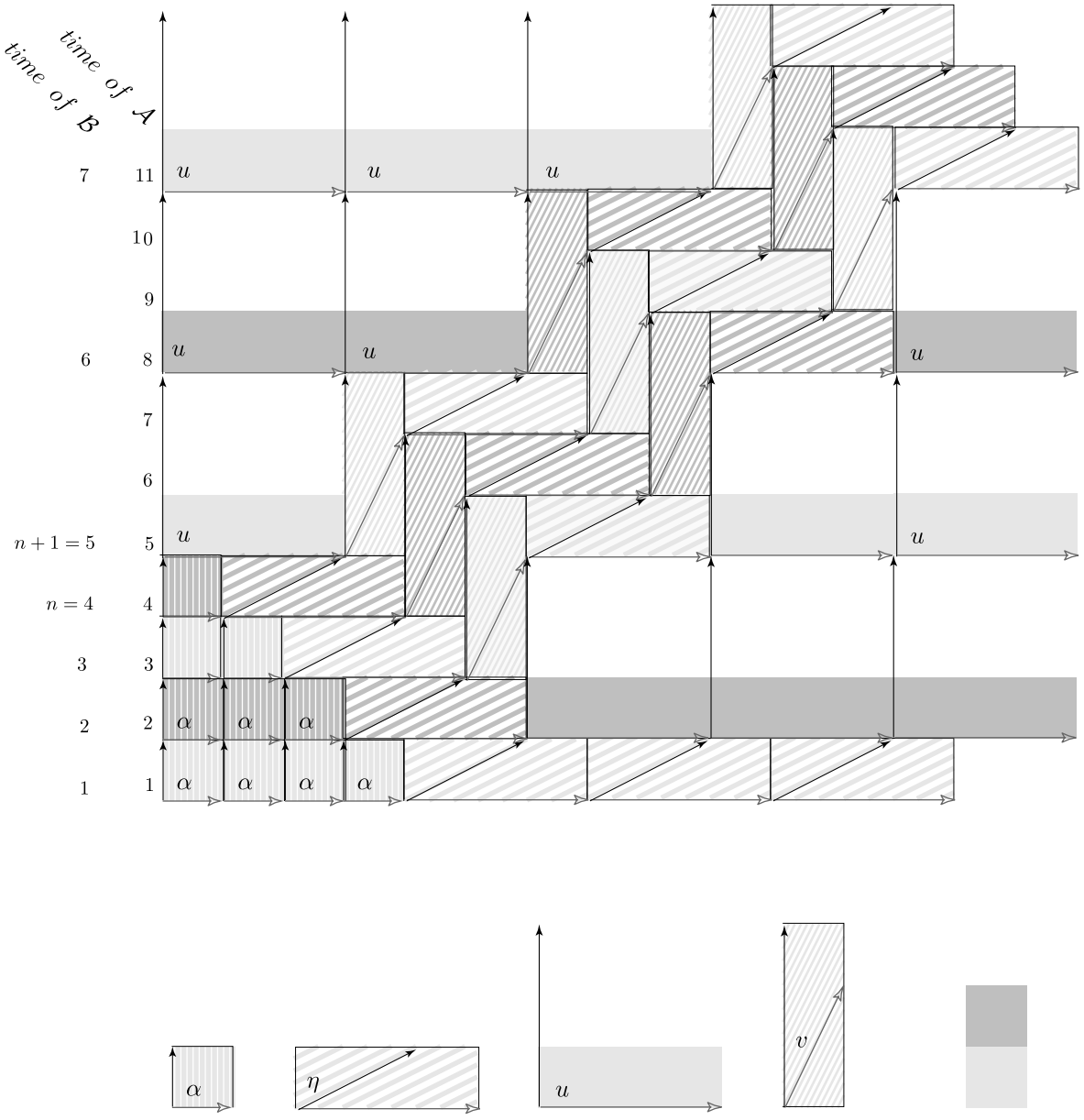


Fig.12.

- **Functioning** :
  Here pieces are : α,η,v,u

and all the rules may be found in the following evolution of configuration $LB\ \alpha\ \alpha\ \alpha\ \alpha\ \eta\ \eta\ \eta$

$$\begin{array}{cccccccc} LB & \alpha & \alpha & \alpha & \alpha & \eta & \eta & \eta & \dots \\ LB & \alpha & \alpha & \alpha & \eta & u & u & u & \dots \\ LB & \alpha & \alpha & \eta & v & \eta & u & u & \dots \\ LB & \alpha & \eta & v & \eta & v & \eta & u & \dots \\ LB & u & v & \eta & v & \eta & v & \eta & \dots \\ LB & u & u & v & \eta & v & \eta & v & \dots \\ LB & u & u & u & v & \eta & v & \eta & \dots \end{array}$$

- **Application** : linear speeding up, by a factor k.

  The way to do the product is always the same, and is shown in Figure 12.

  The new recogizing time will clearly be :

$$T'(n) = n + 1 + \left\lceil \frac{D(n)}{k} \right\rceil .$$

  The number of states of the accelerated c.a is here, at most, $3s^k + s$.

  Indeed the strong speeding up is here realized in a marvelously simple manner, without auxiliary synchronization, with a remarkably small number of states : here is a beautiful feat of strength of the grouper c.a's.

9.4 An afterthought on our definition, second formal approach

When we have defined finite and semi-infinite linear cellular automata, the states were always abstract formal symbols, and Q the set of states had no structure. But the states of our geometrical c.a's are taken out of the set of geometrical figures in $\mathbb{R}^2$, with moreover an origin and a couple of two vectors ! So that it would be more honest to separate in our definition

- the underlying c.a B where the states are the formal symbols for the pieces, i.e their names

- a mapping E from the set of these states in the set of geometrical pieces, the "geometrical interpretation" of the states.

This definition is more general than definition of section 9.2 : indeed, with the first formal definition, three pieces π_1, π_2, π_3 could have only one image $d(\pi_1, \pi_2, \pi_3)$. With our second formal definition different states may have the same geometrical interpretation, so that triple (π_1, π_2, π_3) may correspond to different triples of states having different images. Otherwise said, this amounts to putting colors, (equivalent to giving names), to pieces.

All the conditions (continuity, computability, covering) are conditions on this interpretation, which are necessary for the resulting grouper c.a to be an applicable tool.

Thus a grouper c.a becomes a couple consisting of an ordinary c.a and a geometric interpretation mapping : $(\mathcal{B}, \mathcal{E})$. Therefore we cannot flatly present it as an ordinary c.a.

It must be stressed that all the grouper c.a's in our examples, though they are tools for building elaborate c.a's, are of a disarming simplicity !

Let us recall here the possibility of introducing non-determinism for grouper c.a's, which may allow (see example 9.3.1) elegant descriptions for some tensor products.

9.5 Parallel input to set up an initial configuration

In all the examples that we have presented, groupers were applied to speed up recognizers. The initial configuration of these recognizers is not exactly the word to be recognized, but an image of this word by transition function of time 0, $\delta^0(e, e, e, .)$. It has the same length as the input word, with states corresponding to the input letters. The initial configurations of the grouper c.a's for different words were nearly the same, differing only by the length of their first section, in which the image of the input word took place. So we have the idea that such simple configurations could also be set up by the input word entering some equivalent of a quiescent configuration of an ordinary c.a, a line made of some abstract neutral state ε, bound to receive the inputs and disappear once and for all immediately after, with transition function of time 0

$$d^0 : \{LB, \varepsilon\} \times \{\varepsilon\} \times \{\varepsilon, RB\} \times X \longmapsto P,$$

where X is the input alphabet, such that

$$d^0(LB \text{ or } \varepsilon, \varepsilon, \varepsilon, x) = \pi_1$$

$$d^0(\varepsilon, \varepsilon, \varepsilon, *) = \pi_2$$

$$d^0(LB \text{ or } \varepsilon, \varepsilon, \varepsilon, -) = \pi_3$$

where $*$ is a marker at the end of the input word.

Is it possible to extend the operation of the tensor product to time 0 ? For this we want to give shape to the abstract configuration of time 0 of the grouper, so that it will superimpose itself on the initial quiescent configuration of the ordinary c.a $\mathcal{A}$. It must then be a line of consecutive squares and rectangles, whose width is not precised and will depend on what space the pieces produced by the entering inputs need to expand. Origin of ε, spacing and timing vectors are then as shown in Figure 13, where a configuration $LB, \varepsilon, \varepsilon, \ldots$ is evocated.

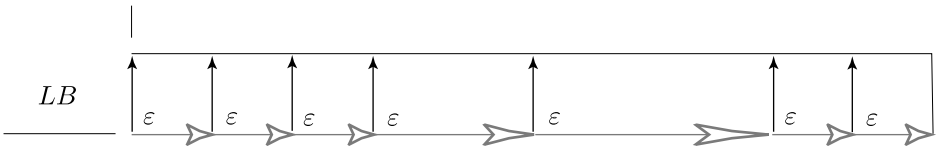


Fig.13.

An illustration for the square diagonal grouper is given in Figures 14 and 15.

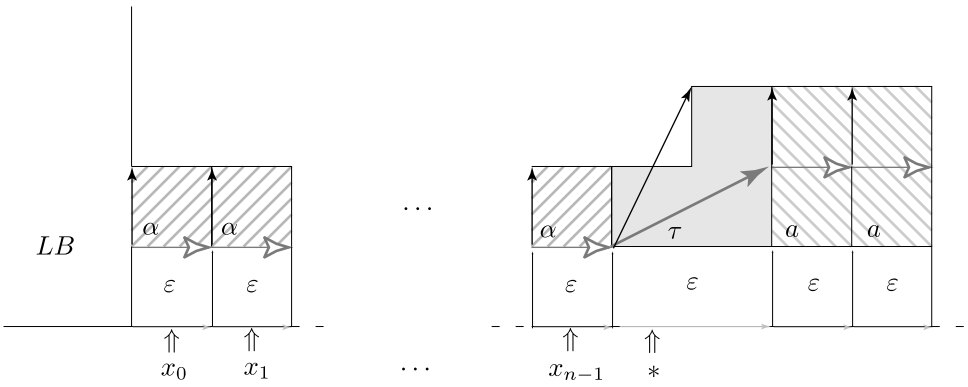


Fig.14.

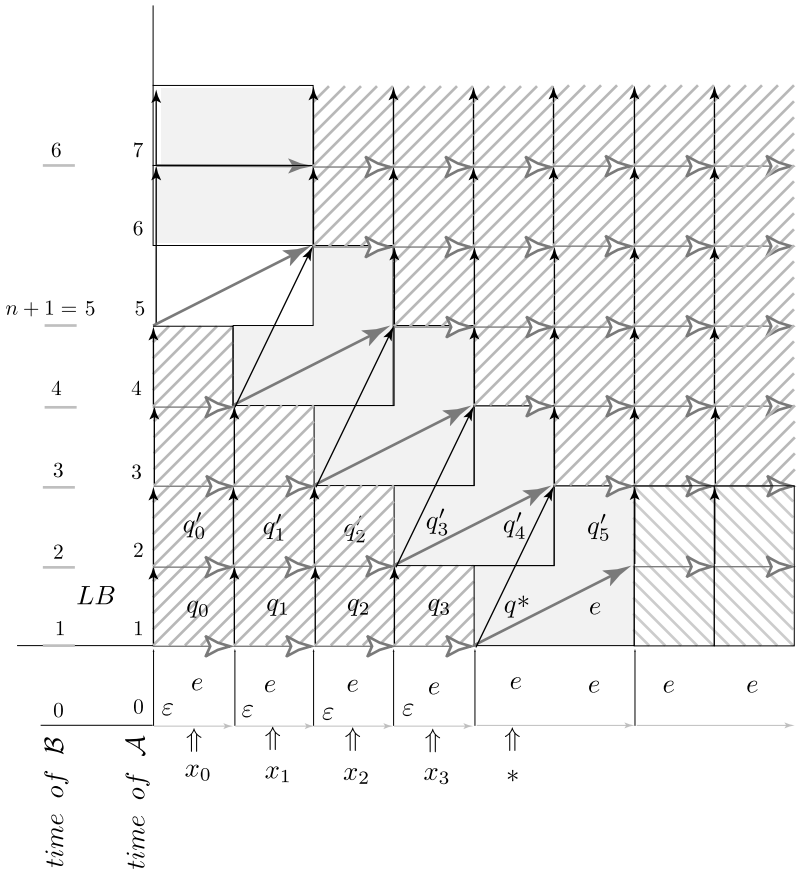


Fig.15. The s.t.d of $\mathcal{R} \otimes \mathcal{A}$ represented in the space-time of $\mathcal{A}$

Chapter 10

n-dimensional c.a's with arbitrary neighbourhoods

This chapter is the sole incursion of this book into dimensions higher than 1 for nets of automata.

One c.a of dimension 2 is famous : John Conway's game of Life [43], whose remarkably simple rules and fascinating properties called forth hords of fanatics in the seventies.

If not for fun, at least for completness, we must certainly explore n-dimensional c.a's, should it only be to compare them with one-dimensional c.a's, and investigate what extra properties they may share. Indeed we naturally foresee that with greater dimension, as more interconnections are possible, c.a's must gain in efficiency and speed.

A very simple observation ([3]) immediately confirms this : if synchronizing n cells with a one-dimensional c.a requires time $2n-2$, with a 2-dimensional c.a, synchronization of a square net of $n = r^2$ cells can be done in time $4r-4 = 4\sqrt{n}-4$ (using one-dimensional synchronization) . Indeed, suppose that our n cells are cells

$$(1,1)\dots(1,r)\dots(r,1)\dots(r,r).$$

In time $2r-2$ we can synchronize cells

$$(1,1)\dots(1,r)$$

If fire state on each $(1,i)$ starts a synchronization of cells

$$(1,i)\dots(r,i)$$

in $2r-2$ extra time-steps, all r^2 cells are synchronized at time $4r-4$. (The neighbours used for each cell (x,y) are $(x-1,y),(x,y),(x+1,y)$ in the first stage, and $(x,y-1),(x,y),(x,y+1)$ in the second stage).

Alas, study of n-dimensional c.a's is also bound to be of increasing complexity.

10.1 Description

10.1.1 Definition

Definition of these c.a's will be copied on that of one-dimensional c.a's.

Cells are now regularly disposed in an n-dimensional space. Whatever be their shape, we imagine that they are centered on the points of $\mathbb{Z}^n$, and we denote them by the coordinates of their center

$$\bar{z} = (z_1, \ldots, z_n)$$

in particular, central cell is

$$\bar{0} = (0, \ldots, 0).$$

Let us remind that $\mathbb{Z}^n$ is a group for addition

$$\bar{z} + \bar{y} = (z_1 + y_1, \ldots, z_n + y_n)$$

and a module on $\mathbb{Z}$ with scalar multiplication

$$k\bar{z} = (kz_1, \ldots, kz_n).$$

Let us point out here that a group morphism of $\mathbb{Z}^n$ to $\mathbb{Z}^p$ is also automatically a linear mapping from the module $\mathbb{Z}^n$ to the module $\mathbb{Z}^p$. This is because the scalar multiplication can be built from addition :

$$k\bar{z} = \underbrace{\bar{z} + \ldots + \bar{z}}_{k \ \ times}$$

$$(-1)\bar{z} = -\bar{z}.$$

The neighbourhood of a cell is the (finite) set of cells under which influence it lays, (among which the cell itself is generally, but not necessarily, to be found). If V denotes the neighbourhood of cell $\bar{0}$, the neighbourhood of any cell $\bar{z}$ is

$$\bar{z} + V = \{\bar{z} + \bar{v} \mid \bar{v} \in V\}.$$

V is the "stencil" of all the neighbourhoods, and we shall often simply call it the stencil of the c.a.

We must underline here that in the case of one-dimensional c.a's, in all the preceding chapters, we considered only neighbourhoods of first neighbours. So that the following study, with the case $n = 1$, comprises a generalization of our preceding study to one-dimensional c.a's with arbitrary neighbourhoods. We shall point results for dimension 1 later in this chapter.

Taking up Cole's notations, we denote

$$|\bar{z}| = \sum_{i=1}^{n} |z_i| \quad (\textit{ standard notation would be } \|\bar{z}\|_1)$$

$$\|\bar{z}\| = \max_{i=1,\dots,n} |z_i| \quad (\textit{ standard notation would be } \|\bar{z}\|_\infty)$$

the usual norms on $\mathbb{Z}^n$, the most important neighbourhoods are the most regular ones, that is :

$$J_1 = \{\bar{z} \mid |\bar{z}| \leq 1\} \qquad J_k = \{\bar{z} \mid |\bar{z}| \leq k\}$$

$$H_1 = \{\bar{z} \mid \|\bar{z}\| \leq 1\} \qquad H_k = \{\bar{z} \mid \|\bar{z}\| \leq k\}$$

which are Von Neumann's and Moore's neighbourhoods (Figure 1). (If dimension is 1 they are identical).

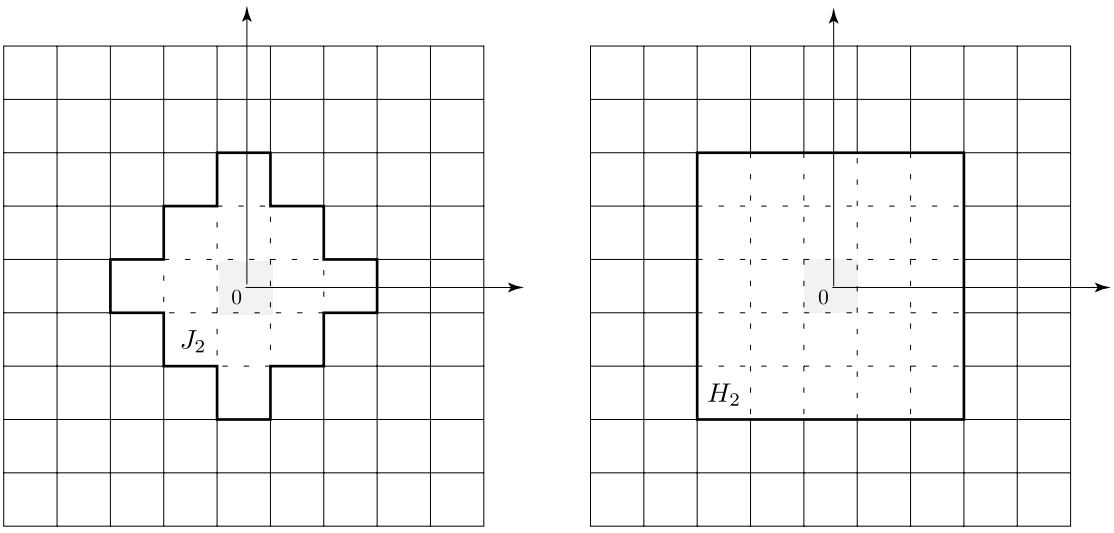


Fig.1 : Von Neumann and Moore's neighbourhoods in $\mathbb{Z}^2$

The set of states will still be denoted Q and it always contains a quiescent state e. (See below for the meaning of "quiescent")

Configurations A V-configuration of states is a mapping $V \to Q$, it represents the global state of a neighbourhood.

A $\mathbb{Z}^n$-configuration is a mapping $\mathbb{Z}^n \to Q$, we shall also call it a global configuration or simply a state of the c.a. We shall never consider but almost quiescent configurations, that is configurations that are quiescent except on a finite number of cells.

The transition function is a mapping

$$\delta : Q^V \to Q.$$

It gives the state of a cell at time $t+1$ from the states of the neighbouring cells at time t.

For a quiescent V-configuration, we ask that

$$\delta(e^V) = e.$$

The initial configuration will normally be an "impulse" configuration, where only cell $\bar{0}$ is not quiescent.

The evolution of the c.a with time is the succession of the $\mathbb{Z}^n$-configurations resulting from the repeated application of the global transition to the initial state. It seems still possible to represent these successive configurations for $\mathbb{Z}^2$-c.a's, in successive diagrams (see 10.5.6), but totally impossible for greater dimensions.

Inputs and outputs enter and exit through cell $\bar{0}$. Input alphabet is always denoted X.

Inputs and outputs are sequential : the discussion we had in section 3.1.1 leads us here, not differently than in dimension 1, to restrict the definition to sequential inputs and outputs. A supplementary argument in dimension 2 and more, is that there is no natural shape on which such a parallel input should be entered.

As for one-dimensional c.a's, we must introduce here the transition function for cell $\bar{0}$

$$\delta_0 : X \times Q^V \longrightarrow Q$$

which takes into account, not only the states of the neighbours of cell $\bar{0}$ at time t, but also the input at time t.

Let us remind that in the case of recognizers the output alphabet is reduced to the set of possible characterizations of the states that are : non halting, halting and accepting, halting and rejecting, or else $\{0, 1\}$.

10.1.2 $\mathbb{Z}^n$ versus $\mathbb{N}^n$

In the case of linear c.a's, we chose studying the half-lines, indexed on $\mathbb{N}$, instead of the lines, indexed on $\mathbb{Z}$. In this chapter we shall prefer studying c.a's on $\mathbb{Z}^n$. So our n-dimensional c.a's are $\mathbb{Z}^n$-c.a's.

In any case, as in dimension 1, while any c.a on $\mathbb{N}^n$ might be considered a c.a on $\mathbb{Z}^n$, any $\mathbb{Z}^n$-c.a can reduce to a $\mathbb{N}^n$-c.a : for this it suffices that each cell $\bar{z} = (z_1, \ldots, z_n)$ of $\mathbb{N}^n$, after n foldings, should represent the 2^n cells

$$(\varepsilon_1 z_1, \ldots, \varepsilon_n z_n) \qquad \varepsilon_i = \pm 1$$

so the set of states will be Q^{2^n}. The neighbourhood stencil will be

$$V' = \{(z_1, \ldots, z_n)\ \epsilon\ \mathbb{N}^n \ \mid\ \exists\, \varepsilon_1, \ldots, \varepsilon_n \quad (\varepsilon_1 z_1, \ldots\ldots, \varepsilon_n z_n) \in V\}$$

In particular, if V is symmetrical in all directions, then $V' = V$.

10.1.3 Neighbourhoods and dimension

Let us start by examining a few examples :

example 1 a c.a on $\mathbb{Z}^2$ with neighbourhood stencil $V = \{(-1,0),(0,0),(1,0)\}$.

The lines of cells having a given ordinate $z_2 = c$ have independant evolutions, each constitutes a linear c.a in itself. If the initial state is an impulse, only line $z_2 = 0$ is active. In this case we are reluctant to speak of a 2-dimensional c.a. The only interesting c.a, the one containing cell $\bar{0}$, is really a linear c.a on $\mathbb{Z}$ with stencil $\{-1,0,1\}$.

This first example shows that if for n-dimensional c.a's we allow any stencil whatever, we shall not always have a real n-dimensional c.a.

example 2 a c.a on $\mathbb{Z}$ with stencil $\{-2,0,2\}$ (See Figure 2).

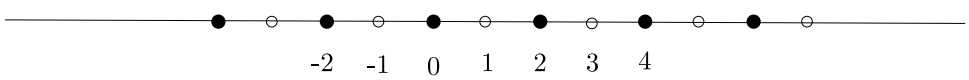


Fig.2

Here it is odd and even cells that have independent evolutions : this c.a is formed of 2 imbricated linear c.a's which work independently. If the initial state is an impulse, the second one does not function at all ! As for the interesting c.a, the one containing cell $\bar{0}$, which is $2\mathbb{Z}$, we shall change indexes of its cells by taking as new basis for $2\mathbb{Z}$ element 2. Our c.a then becomes a $\mathbb{Z}$-c.a with stencil $\{-1,0,1\}$.

This second example points out that we must beware that the stencil of the neighbourhoods is not artificially complicated.

example 3 the seemingly $\mathbb{Z}^2$-c.a with stencil $V = \{(-2,-1),(0,0),(2,1)\}$ is really only a $\mathbb{Z}$-c.a with stencil $\{-1,0,1\}$ (see Figure 3).

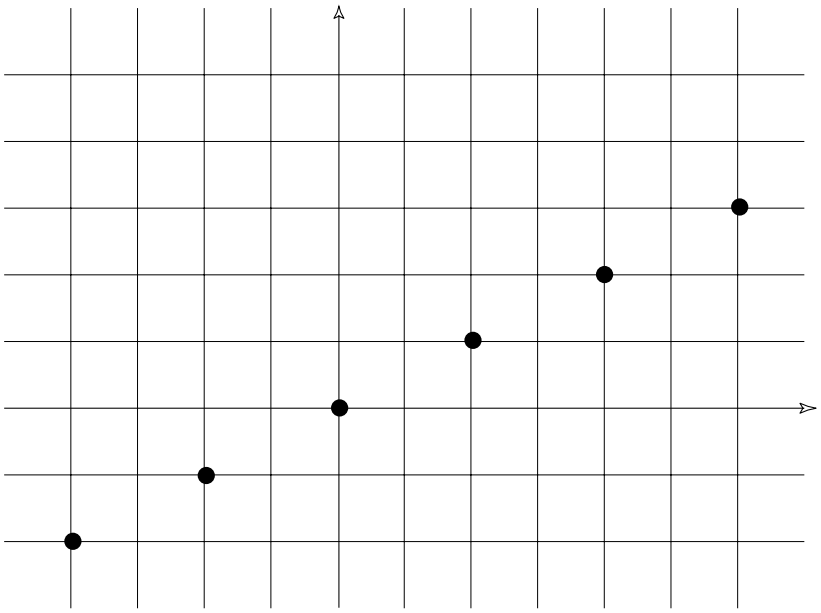


Fig.3

In this example we have the two problems met separately in examples 1 and 2. It is also the case in next example 4, with figures a bit more difficult to draw and to read because dimension is one more.

example 4 the seemingly $\mathbb{Z}^3$-c.a with stencil $V = \{(0,1,1),(0,2,2),(2,0,2)\}$ is really only a $\mathbb{Z}^2$-c.a with stencil $V = \{(1,0),(0,1),(0,2)\}$ (see Figure 4).

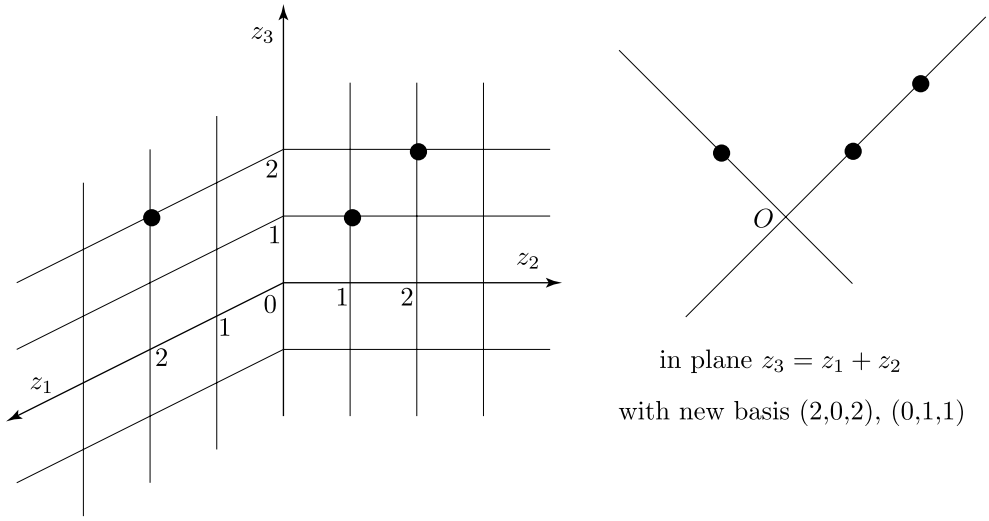


Fig.4

Through all these examples it appears clearly that what matters really is the net formed by the "active" cells, those which have a chance of getting out of the quiescent state sooner or later, and which are determined by the neighbourhood V.

Let E_1 be the set of cells liable to be influenced by cell $\bar{0}$ at time 1. Clearly, $\bar{z}$ is in E_1 if and only if the neighbourhood $\bar{z} + V$ of $\bar{z}$ contains $\bar{0}$, i.e. $z \,\epsilon - V$. A simple induction on i gives that the set E_i of cells liable to be influenced by cell $\bar{0}$ at time i is

$$E_i = -iV = \underbrace{(-V) + \ldots + (-V)}_{i \ times} \text{ where } -V = \{-\bar{v} \mid \bar{v} \in V\}.$$

Note : take care that

$$iV = \underbrace{V + \ldots + V}_{i \ times} \text{ is not } \{i\bar{v} \mid \bar{v}\epsilon V\} \text{ but } \{v_1 + \ldots + v_i \mid v_1, \ldots, v_i \,\epsilon\, V\}$$

and

$$-iV \text{ is not } \{-i\bar{v} \mid \bar{v}\epsilon V\} \text{ but } -(iV).$$

Thus the active cells, in the case of an impulse initial state are all those in

$$\bigcup_{i=0}^{\infty} -iV.$$

But even if the initial state is not an impulse state, with only cell $\bar{0}$ non quiescent, we separate "independent" sets of cells. Indeed, two cells $\bar{z}$ and $\bar{z}'$ have a chance

of exercising some influence on one another sooner or later only if

$$\text{either} \quad \bar{z'} \ \epsilon \ \bar{z} + \bigcup_{i=0}^{\infty} -iV \quad \text{or} \quad \bar{z} \ \epsilon \ \bar{z'} + \bigcup_{i=0}^{\infty} -iV$$

i.e.

$$\bar{z} - \bar{z'} \ \epsilon \ \bigcup_{-\infty}^{\infty} iV = sg(V)$$

where $sg(V)$ is the subgroup of $\mathbb{Z}^n$ generated by V. This relation between $\bar{z}$ and $\bar{z}'$ is an equivalence relation, the "independent" sets of cells are the classes for this equivalence, and the set of cells constituting the c.a with which we are concerned is the one containing cell $\bar{0}$, namely $sg(V)$.

Let us look now at the conditions we wish for our neighbourhoods. Condition needed so as to have a real n-dimensional c.a is that the subgroup $sg(V)$, which is also a submodule of the free module $\mathbb{Z}^n$, has dimension n. (For results about $\mathbb{Z}$-modules see textbooks, e.g [34]). Nevertheless, this submodule needs not be all of $\mathbb{Z}^n$. However, it has n generators , and if we take these as new basis, then we have exactly

$$\boxed{sg(V) = \mathbb{Z}^n} \ .$$

This is the condition that we require for the stencil V of neighbourhoods of any $\mathbb{Z}^n$-c.a. It will be part of our definition of $\mathbb{Z}^n$-c.a's.

10.1.4 Real time

We shall not copy here section 2.2.3 which does not need the slightest change, but we mention it because it is not long before we focus on languages recognized by $\mathbb{Z}^n$-c.a's in strictly real time.

10.2 Cole's general theorem

The idea of accelerating a c.a by grouping the cells comes naturally to anybody's mind, but its most early, exhaustive, and precise formulation is Cole's [6]. We shall present it now, very slightly modified in its presentation.

We insist that here cells will be grouped in a perfectly regular manner , the groups having a determined shape (the shape of K) and being regularly disposed (by morphism ψ).

And we consider only recognizers.

10.2.1 Here acceleration is weak

We want to speed up k times an automaton $\mathcal{A}$ on $\mathbb{Z}^n$ having stencil V : in one time-step the accelerated automaton $\mathcal{A}'$ will have to do what $\mathcal{A}$ does in k

time-steps. As a first consequence, in one time-step it must deal with k inputs of X :

at time 0 : $x_0, x_1, \ldots, x_{k-1}$ where x_i is the input at time i

...

at time t : $x_{tk}, \ldots, x_{tk+k-1}$

.... .

Thus, the input alphabet has to be the set of k-tuples of X, augmented with incomplete k-tuples, as we have already encountered in the one-dimensional case. The new input is what we have called in chapter 7 the k-grouping of the original one.

10.2.2 Characteristic elements of the accelerated c.a

Suppose $\mathcal{A}'$ is some p-dimensional c.a which emulates the n-dimensional c.a $\mathcal{A}$, but k times faster.

If $\mathcal{A}$ recognized some input in time $T = kt$, $\mathcal{A}'$ must recognize the corresponding k-grouped input in time t : that is, if at time kt cell $\bar{0}$ of $\mathcal{A}$ entered a halting (accepting or rejecting) state, cell $\bar{0}$ of $\mathcal{A}'$ must enter a halting state at time t.

Same thing if $\mathcal{A}$ entered a halting state at some time T not a multiple of k :

$$k(t-1) < T < kt.$$

Let us denote by W the stencil of $\mathcal{A}'$ (such that $sg(W) = \mathbb{Z}^p$). Cells of $\mathcal{A}'$ code finite sets of cells of $\mathcal{A}$, of a determined pattern and these sets are regularly disposed in $\mathcal{A}$: their common stencil is $K \subset \mathbb{Z}^n$, and their centers are distributed in $\mathbb{Z}^n$ by the group morphism (or linear mapping)

$$\psi : \mathbb{Z}^p \longrightarrow \mathbb{Z}^n.$$

so that cell $\bar{l}$ of $\mathbb{Z}^p$ codes the set of cells of $\mathcal{A}$

$$\psi(\bar{l}) + K.$$

Although the context is sufficient to distinguish between the neutral elements of $\mathbb{Z}^p$ and $\mathbb{Z}^n$, we shall give them indexes p or n. Accordingly we shall write that cell $\bar{0}_p$ of $\mathcal{A}'$ codes

$$\psi(\bar{0}_p) + K = \bar{0}_n + K = K.$$

As we wish that the inputs and outputs, which go in and out through cell $\bar{0}_n$ of $\mathcal{A}$, should go in and out through cell $\bar{0}_p$ of $\mathcal{A}'$, we must have

$$\boxed{\text{condition (0) : } \bar{0}_n \in K}.$$

10.2.3 A technical lemma

Here we want to state precisely on which elements the state of a cell $\bar{z}$ of $\mathcal{A}$ at time $T+k$ depends, going back down to time T. This state is determined by

1/ states at time $T+k-1$ of cells $\bar{z}+V$, and only these states, unless $\bar{z}=\bar{0}_n$ in which case it also depends on input x_{T+k-1}

2/ these states in turn are determined by states of cells of $\bar{z}+V+V = \bar{z}+2V$ and only on these, unless $\bar{0}_n \in \bar{z}+V$ in which case they also depend on x_{T+k-2}

- ...

i/ states of cells of $\bar{z}+iV$, and only these, unless $\bar{0}_n \in \bar{z}+(i-1)V$, in which case they also depend on x_{T+k-i}

- ...

k/ states of cells of $\bar{z}+kV$, and only on these, unless $\bar{0}_n \in \bar{z}+(k-1)V$ in which case they also depend on x_T

Finally we conclude that

- if $\bar{0}_n$ belongs neither to $\{\bar{z}\}$, nor to $\bar{z}+V$, ... nor to $\bar{z}+(k-1)V$, the state of $\bar{z}$ depends only on states on $\bar{z}+kV$ at time T

- if $\bar{0}_n \in \bar{z}+iV$ for some i in $\{0,\dots,k-1\}$ (where $0V=\bar{0}_n$), this state also depends on some of the inputs x_T, x_{T+1}, ... , x_{T+k-1} (the $i+1$ last ones).

Namely, for $T=kt$ we can state

Technical lemma 10.2.1 *the state of a cell $\bar{z}$ of the n-c.a $\mathcal{A}$ at time $k(t+1)$ is completely determined by*

- *states on $\bar{z}+kV$ at time kt*

- *and, if $\bar{0}_n \in \bar{z}+\bigcup_{i=0}^{k-1} iV$, inputs $(x_{tk}, x_{tk+1}, \dots, x_{tk+k-1})$ (at least some of them).*

In the latter sequence we recognize the future input for $\mathcal{A}'$ at time t.

10.2.4 Necessary conditions

Let us come back to this desired $\mathcal{A}'$. We wish, for any of its cell $\bar{l}$ and any t, that

- state of $\bar{l}$ at t be the configuration of states of $\psi(\bar{l})+K$ at kt

- state of $\bar{l}$ at $t+1$ be the configuration of states of $\psi(\bar{l})+K$ at $kt+k$

From preceding lemma :

the configuration of states of $\psi(\bar{l}) + K$ at $kt + k$ depends on states at time kt on

$$\psi(\bar{l}) + K + kV \tag{10.1}$$

and only on those states, unless

$$\bar{0}_n \in \psi(\bar{l}) + K + \bigcup_{i=0}^{k-1} iV \tag{10.2}$$

in which case the configuration also depends on (at least some elements of)

$$(x_{tk}, x_{tk+1}, \ldots, x_{tk+k-1}).$$

But, if W is the stencil for $\mathcal{A}'$,

> the state of any cell $\bar{l}$ at $t+1$ must depend only on input at t and states at t on $\bar{l} + W$, state of a cell $\bar{l} \neq \bar{0}_p$ depending only on the latter.

In view of (10.1), it so appears necessary that states of $\mathcal{A}'$ at t on $\bar{l} + W$, which are states of $\mathcal{A}$ at kt on $\psi(\bar{l}) + \psi(W) + K$, should contain states of $\mathcal{A}$ at kt on $\psi(\bar{l}) + K + kV$, which leads to condition

$$\psi(\bar{l}) + K + kV \subset \psi(\bar{l}) + \psi(W) + K$$

that is

$$\boxed{\text{condition (1) : } K + kV \subset \psi(W) + K}$$

and because of (10.2) necessary also that

$$\boxed{\text{condition (2) : if } \bar{l} \neq \bar{0}_p \text{ then } \bar{0}_n \notin \psi(\bar{l}) + K + \textstyle\bigcup_{i=0}^{k-1} iV}$$

Condition (1) expresses that stencil W is large enough for neighbourhoods of $\mathcal{A}'$ to contain the k-iterated neighbourhoods of sets K in $\mathcal{A}$.

Condition (2) expresses that, if $\bar{l}$ is not $\bar{0}_p$, then it is not influenced by inputs in cell $\bar{0}_n$ of $\mathcal{A}$ between time 0 and time $k-1$.

condition 1 implies $p \geq n$

Indeed this condition implies

$$K + V \subseteq \psi(\mathbb{Z}^p) + K \quad \text{because } W \subset \mathbb{Z}^p \tag{10.3}$$

and

$$V \subseteq \psi(\mathbb{Z}^p) + K \quad \text{because } \bar{0}_n \epsilon K \tag{10.4}$$

Let us suppose that $p < n$.

Any element of $\psi(\mathbb{Z}^p)$ is a linear combination of

$$s_1 = \psi(1, 0, \ldots, 0), \ldots, s_p = \psi(0, 0, \ldots, 1)$$

and thus belongs to the $\mathbb{Z}$-submodule $\mathbb{Z}s_1 + \ldots + \mathbb{Z}s_p$.

If every element of V were to belong to the subspace $\mathbb{R}s_1 + \ldots + \mathbb{R}s_p$, then $\mathbb{Z}^n = sg(V)$ would be included in this subspace, and this would imply that $\mathbb{R}^n$ in turn would be included in $\mathbb{R}s_1 + \ldots + \mathbb{R}s_p$, which is impossible because $p < n$. Thus there exists some v in V which does not belong to the $\mathbb{R}$-subspace of $\mathbb{R}^n$ generated by s_1, ... , s_p.

Because of (10.2), $v \in \psi(\mathbb{Z}^p) + K$, so there exists d_1 in $\psi(\mathbb{Z}^p)$ and k_1 in K such that

$$v = d_1 + k_1 \quad \text{otherwise written} \quad k_1 = v - d_1$$

Because of (10.1), $k_1 + v \in \psi(\mathbb{Z}^p) + K$, so there exists d_2 in $\psi(\mathbb{Z}^p)$ and k_2 in K such that

$$k_1 + v = d_2 + k_2 \quad \text{otherwise written} \quad k_2 = 2v - d_1 - d_2$$

And so on, for all i we find in K some

$$k_i = iv - d_1 - d_2 - \ldots - d_i.$$

Elements k_i are all distinct, because if some k_i were equal to some k_j, with $i > j$, we would have

$$(i - j)v = d_{j+1} + \ldots + d_i$$

so that v would belong to the subspace generated by s_1, ... , s_p. Thus K is proved to be infinite, which is contrary to our hypotheses. Thus $p < n$ is impossible.

condition 2 implies $p \leq n$

Indeed, if ψ were not injective, there would exist $\bar{l} \neq \bar{0}_p$ such that $\psi(\bar{l}) = \psi(\bar{0}_p) = \bar{0}_n$, so that condition 2 could not be satisfied. Thus ψ must be an injective morphism, which implies that p cannot be greater than n.

conclusion is that $p = n$

If an accelerated c.a exists, it has the same dimension as the original c.a.

From now on we shall no more distinguish $\bar{0}_p$ and $\bar{0}_n$. Cell $\bar{0}$ of $\mathcal{A}'$ codes K, while cell $\bar{0}$ of $\mathcal{A}$ is only $\{\bar{0}\}$.

10.2.5 The theorem

Theorem 10.2.2 *Let $\mathcal{A}$ be an n-c.a (set of states Q) of stencil V , and*

$\psi : \mathbb{Z}^n \longrightarrow \mathbb{Z}^n$ *an injective morphism*

W *a finite subset of* $\mathbb{Z}^n$ *such that* $sg(W) = \mathbb{Z}^n$

K *a finite subset of* $\mathbb{Z}^n$ *containing* $\bar{0}$

satisfying, for some integer k, conditions

$$\begin{array}{ll} (1) & K + kV \subset \psi(W) + K \\ (2) & if \quad \bar{l} \neq \bar{0} \quad then \quad \bar{0} \notin \psi(\bar{l}) + K + \bigcup_{i=0}^{k-1} iV \end{array}$$

Then there exists a $\mathbb{Z}^n$-c.a $\mathcal{A}'$ of stencil W such that, if $\mathcal{A}$ recognizes (/rejects) an input word in time T, then $\mathcal{A}'$ recognizes (/rejects) the k-grouped word in time $\lceil T/k \rceil$.

Moreover, the number of states of $\mathcal{A}'$ is at most $|Q|^{|K|}$, where $|Q|$ and $|K|$ denote the number of states of $\mathcal{A}$ and the number of cells in K.

This we shall now verify, after all we need has been previously collected in preceding sections.

Cells $\bar{l}$ of $\mathcal{A}'$ are the $\psi(\bar{l}) + K$ and states of $\mathcal{A}'$ are the K-configurations of states of Q (hence the upper bound for the number of states of $\mathcal{A}'$).

Transition for a cell $\bar{l}$ (with its W-neighbourhood around) is obtained by repeting k times transition of $\mathcal{A}$ to cells of $\psi(\bar{l})+K$ (each with its V-neighbourhood around).

For any c.a $\mathcal{A}$ we must distinguish two transitions : one for ordinary cells, δ, and one for cell $\bar{0}$, δ_0, which takes the input into account. It will be the same for $\mathcal{A}'$. If we consider a cell $\bar{l} \neq \bar{0}$, the k successive applications of δ to cells of $\psi(\bar{l}) + K$ will concern cells among which, by condition (2), $\bar{0}$ never appears. On the other hand, condition (1) guarantees that $\bar{l} + W$ contains all the cells whose states contribute when we apply δ k times.

As for cell $\bar{0}$, application of transition of $\mathcal{A}$ to K comprises one application of δ_0 to cell $\bar{0}$ of K and to an input x_{tk+k-1}, while next applications all comprise an application of δ_0 to cell $\bar{0}$ and successively to inputs x_{tk+k-2}, ..., x_{tk+1}, x_{tk}. With condition (1) and input $(x_{tk}, x_{tk+1}, \ldots, x_{tk+k-1})$ we have all the information to compute δ'_0.

With this definition it is clear that state of $\bar{l}$ at time t will be the K-configuration of states of $\psi(\bar{l}) + K$ at time kt.

Let us now examine halting and halting states : if a halting state a, accepting or rejecting, appears on cell $\bar{0}$ of $\mathcal{A}$ at time kt, at time t cell $\bar{0}$ of $\mathcal{A}'$ which is set K of cells of $\mathcal{A}$, has its cell $\bar{0}$ in state a. We shall naturally agree that any K-configuration having state a on cell $\bar{0}$ is a halting state of $\mathcal{A}'$, of the same sort, accepting or rejecting, as a. If however state a should appear at some time τ

$$k(t-1) < \tau < kt$$

computing of δ' cannot be completed up to time t, unless we complete δ' by deciding that for a state of $\mathcal{A}'$ and a W-configuration leading to such a situation this transition leads to state $\{a\}^K$ of $\mathcal{A}'$, (which is, by the same convention as we made a few lines before , of the same sort, accepting or rejecting, as a).

$\mathcal{A}'$ with set of states and transition so completed for each halting state a of $\mathcal{A}$ will recognize the same input words that $\mathcal{A}$ recognized in time T, but k-grouped, in time $\lceil T/k \rceil$.

Conclusion about this theorem : we could not hope for more general a result, and we shall now apply it to various situations. The theorem tells us what materials we need, and then, not only guarantees the existence of $\mathcal{A}'$ but tells us effectively how to build it.

Now the materials must be found : no problem, Cole has found everything for us !

10.3 Application to weak speeding up

Here we want to prove that it is possible to speed up a $\mathbb{Z}^n$-c.a by any (integer) factor k.

10.3.1 The weak speeding up theorem of Cole

Theorem 10.3.1 *for any c.a $\mathcal{A}$ on $\mathbb{Z}^n$ (with stencil V and input alphabet X) which recognizes some language L in time $T(m)$ and for any $k \geq 2$,*

there exists a c.a $\mathcal{A}'$ on $\mathbb{Z}^n$, having stencil H_1, input alphabet X^k which recognizes in time $\lceil T(m)/k \rceil$ the words of L k-grouped.

Note that if we usually denote by n the length of the words to be recognized, or not, we must presently change our habits because at the moment n is the dimension of the c.a.

Let $j = max\{ \|v\| \mid \bar{v} \epsilon V \}$ and $K = H_{(k-1)j} = \{\bar{z} \mid \|\bar{z}\| \leq (k-1)j\}$ and let ψ be defined by $\psi(\bar{z}) = [2(k-1)j+1]\bar{z}$. Clearly, K is a finite subset of $\mathbb{Z}^n$ which contains $\bar{0}$. We check that the two conditions in Cole's theorem are satisfied.

Let us examine condition (1) : we have $V \subseteq H_j$, so that

$$kV \subseteq kH_j = \underbrace{H_j + \ldots + H_j}_{k \ times} = H_{kj}$$

$$K + kV \subseteq H_{(k-1)j} + H_{kj} = H_{(2k-1)j}.$$

We also have

$$\psi(H_1) = \{(u_1, \ldots, u_n) \mid u_i = -2(k-1)j - 1 \text{ or } 0 \text{ or } 2(k-1)j + 1\}.$$

Let $\bar{z} \in K + kV$, we have :

$$\forall i = 1, \ldots, n \qquad -(2k-1)j \leq z_i \leq (2k-1)j.$$

For all i let us define $\bar{u}$ by

$$u_i = \begin{cases} 2(k-1)j+1 & \text{if} \quad (k-1)j+1 < z_i \leq (2k-1)j \\ 0 & \text{if} \quad -(k-1)j \leq z_i \leq (k-1)j \\ -2(k-1)j-1 & \text{if} \quad -(2k-1)j \leq z_i < -(k-1)j \end{cases}$$

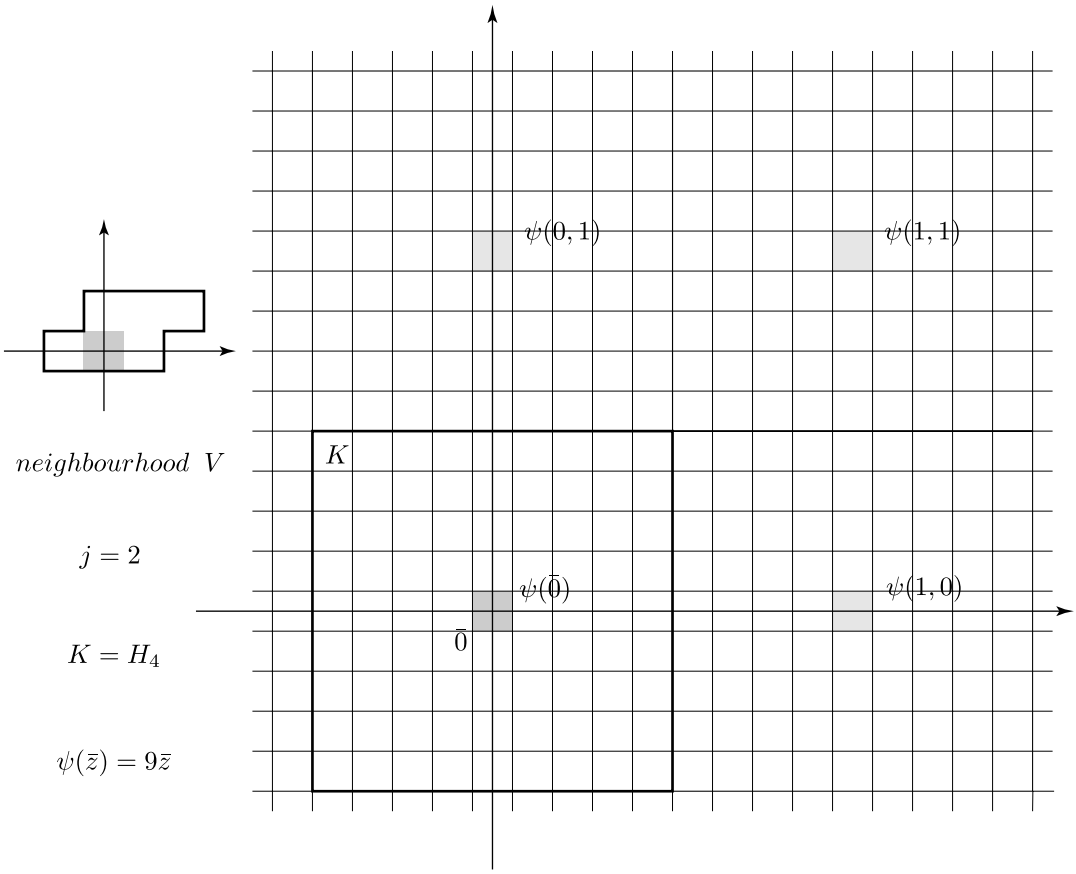


Fig.5 : n=2, k=3

$\bar{u}$ is in $\psi(H_1)$. It is easy to check that for all i, either $|z_i - u_i| \leq (k-1)j$ either $|z_i - u_i| \leq j-1$. But $j-1 < j \leq (k-1)j$ because $k \geq 2$. So for all i we have $|z_i - u_i| \leq (k-1)j$, which proves that $\bar{z} - \bar{u} \in H_{(k-1)j} = K$ hence $\bar{z} \in \psi(H_1) + K$. Condition (1) is satisfied.

Now for condition (2) : for $i = 0, \ldots, k-1$ we have

$$iV \subseteq iH_j = H_{ij} \subseteq H_{(k-1)j}$$

$$\bigcup_{i=0}^{k-1} iV \subseteq H_{(k-1)j}$$

$$K + \bigcup iV \subseteq H_{(k-1)j} + H_{(k-1)j} = H_{2(k-1)j}.$$

If $\bar{l} \neq \bar{0}$, then $\bar{l}$ has at least one non zero coordinate, so that $\psi(\bar{l})$ has at least one coordinate whose absolute value is at least $2(k-1)j+1$. Therefore

$$-\psi(\bar{l}) \notin K + \bigcup iV$$

and condition (2) is satisfied. An example in dimension 2 is given in Figure 5.

10.3.2 Example : $n = 1$, $k = 3$, $V = H_1$, $W = H_1$

This theorem naturally contains the weak acceleration of linear c.a's of scope 1, so let us see what it gives us for this example.

For $\mathcal{A}'$ it proposes

$$\psi(\bar{z}) = 5\bar{z} \qquad K = H_2$$

that is grouping cells by 5. But in chapter 7 (Figure 1) we would have grouped them only by 3 . Does this mean that by treating a particular case directly we can find better solutions than those given by the theorem ? No, at least not in the present case. Indeed, if we examine things a bit more closely, we notice that in chapter 7 we deal with half-lines. If cells are grouped by 3, condition (2) is satisfied for half-lines, but not for lines !

10.3.3 Strong speeding up of recognizers

Exactly as we did in section 7.2 in dimension 1, we can precede the weak accelerated c.a by a grouping process for the inputs. This process may be achieved in any direction, for example in $\mathbb{Z} \times \{0\} \times \ldots \times \{0\}$. It takes time $m+1$, where m is the length of the input.

Thus we have exactly the same strong speeding up results in any dimension n as we had in dimension 1.

Proposition 10.3.2 *For any $\mathbb{Z}^n$-c.a recognizing a language L in time $T(m) > m+1$, and any integer $k \geq 2$, there exists a $\mathbb{Z}^n$-c.a recognizing L in time*

$$m + 1 + \lceil \frac{T(m)}{k} \rceil$$

Corollary 10.3.3 *If a language L is recognized by some $\mathbb{Z}^n$-c.a in linear time $T(m) = am$ (a an integer), then it may be recognized by some $\mathbb{Z}^n$-c.a in time $1 + \lceil m(1+\varepsilon) \rceil$, for arbitrary $\varepsilon > 0$.*

10.3.4 Slowing down

We mention this possibility for the sake of completeness : to slow down a $\mathbb{Z}^n$-c.a by a factor k, as we did in dimension 1, we let it work one time-step out of k.

10.4 Application to neighbourhood changes

No more speeding up here, $k = 1$.

Inputs for the two automatons $\mathcal{A}$ and $\mathcal{A}'$ enter at the same rythm, the time scale is not changed. In this case, nothing forbids observing the outputs, which are unchanged, so the following results are valid for all c.a's, computers as well as recognizers.

In this section we shall declare that two c.a's are *equivalent*, (Cole uses the word "indistinguishable"), if they accomplish the same task in the same time, that is they recognize the same words or compute the same functions, and need the same time.

10.4.1 First result : neighbourhood can always be enlarged

This is of course obvious. Let us nevertheless observe that it is given by Cole's theorem. If $V' \supset V$, then build $\mathcal{A}'$ with

$$\psi = Id_{\mathbb{Z}^n}$$

$$K = \{\bar{0}\}.$$

In particular, for any $\mathbb{Z}^n$-c.a having stencil J_1, there exists an equivalent c.a $\mathcal{A}'$ with stencil H_1.

10.4.2 second result : neighbourhood H_1 is universal

Proposition 10.4.1 *For any $\mathbb{Z}^n$-c.a $\mathcal{A}$ (with arbitrary stencil V) there exists an equivalent c.a $\mathcal{A}'$ on $\mathbb{Z}^n$ with stencil H_1.*

Let then $j = max\{\|\bar{v}\| \mid v \in V\}$ and $\mathcal{A}'$ be the c.a defined by

$$\psi(\bar{z}) = (2j+1)\bar{z}$$

$$K = H_j.$$

Clearly, K is a finite subset of $\mathbb{Z}^n$ which contains $\bar{0}$. We check that the two conditions in Cole's theorem are satisfied.

Let us examine condition (1) : we have

$$V \subseteq H_j$$

$$K + kV = K + V \subseteq H_j + H_j = H_{2j}.$$

We also have

$$\psi(H_1) = \{(u_1, \dots, u_n) \mid u_i = -2j - 1 \text{ or } 0 \text{ or } 2j + 1\}.$$

Let $\bar{z} \in K + kV = K + V$, we have :

$$\forall i = 1, \dots, n \qquad -2j \leq z_i \leq 2j.$$

For all i let us define $\bar{u}$ by

$$u_i = \begin{cases} 2j+1 & \text{if} \quad j+1 \leq z_i \leq 2j \\ 0 & \text{if} \quad -j \leq z_i \leq j \\ -2j-1 & \text{if} \quad -2j \leq z_i \leq -j-1 \end{cases}$$

We have $\bar{u} \in \psi(H_1)$ and $\bar{z} - \bar{u} \in H_j = K$ so $\bar{z} \in \psi(h_1) + K$. Condition (1) is satisfied.

Here condition (2) reduces to $\bar{0} \notin \psi(\bar{l}) + K$. But for any $\bar{l} \neq \bar{0}$, $\bar{l}$ has at least one non zero coordinate, so $\psi(\bar{l})$ has at least one coordinate whose absolute value is at least $2j + 1$, so $-\psi(\bar{l})$ cannot be in $K = H_j$. Condition (2) is satisfied.

So we can limit ourselves to considering $\mathbb{Z}^n$-c.a's with stencil H_1 without any loss in generality.

In dimension 1, these are the c.a's with only first neighbours, so called "of scope 1". We discover here that it is no use studying linear c.a's of scope greater than 1, as far as the number of states is not considered.

10.4.3 Third result : neighbourhood J_1 is universal

At last, with Cole's theorem we shall prove

Proposition 10.4.2 *For any $\mathbb{Z}^n$-c.a with stencil H_1, there exists an equivalent c.a with stencil J_1.*

This is not an easy result.

We may start by some attempts with ψ a homothetie $\psi(\bar{z}) = \lambda(\bar{z})$ and set K having a simple shape, H or J : we find that we do not manage to satisfy conditions. So we try more complicated sets of the form $K = H_\alpha + J_\beta$. Fastidious calculations lead to smallest possible values

$$\lambda = \alpha + \beta + 1$$

$$\alpha = (n-1)^2 - (n-1) + 1$$

$$\beta = (n-1)^2$$

Cole proposes simpler values

$$K = H_{n^2} + J_{n^2}$$

$$\psi(\bar{z}) = (2n^2+1)\bar{z}$$

for which we shall now check that conditions are actually satisfied. Condition (2) reduces to

$$\forall \bar{z} \neq \bar{0} \quad -\psi(\bar{z}) \notin K.$$

But

$$K \subset H_{n^2} + H_{n^2} = H_{2n^2}$$

and if $\bar{z}$ has some non zero coordinate, then $\psi(\bar{z})$ has some coordinate with absolute value not less than $2n^2+1$, so cannot be in K.

It is condition (1) which is really tricky.

We must check that

$$K + H_1 \subseteq \psi(J_1) + K$$

that is

$$H_{n^2} + J_{n^2} + H_1 \subseteq \psi(J_1) + H_{n^2} + J_{n^2}.$$

Let us take $\bar{a} \in H_{n^2}$, $\bar{b} \in J_{n^2}$, $\bar{c} \in H_1$. We are looking for $\bar{q} \in J_1$, $\bar{r} \in H_{n^2}$, $\bar{u} \in J_{n^2}$ such that

$$\bar{y} = \bar{a} + \bar{b} + \bar{c} = \psi(\bar{q}) + \bar{r} + \bar{u}.$$

We have

$$\|\bar{y}\| \leq 2n^2 + 1.$$

Let E be

$$E = \{i \mid y_i > n^2\}.$$

In case E is empty, then

$$\bar{y} \in H_{n^2},$$

and we can take

$$\bar{q} = \bar{0} \quad \bar{r} = \bar{y} \quad \bar{u} = \bar{0}.$$

So let us suppose that E is not empty. Let μ be such that

$$y_\mu = max|y_i| = \|y\|$$

we clearly have

$$\mu \in E \quad \text{and} \quad |y_\mu| > n^2;$$

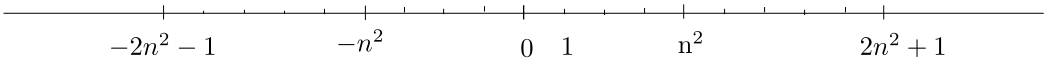


Fig.6

With the help of Figure 6 we can see that there exists

$$q_\mu = -1 \text{ or } 1$$

and

$$r_\mu \text{ with } |r_\mu| \leq n^2$$

such that

$$y_\mu = (2n^2+1)q_\mu + r_\mu.$$

If we take $u_\mu = 0$ we already have

$$y_\mu = (2n^2+1)q_\mu + r_\mu + u_\mu.$$

If for all $i \neq \mu$ we take $q_i = 0$ we have

$$\bar{q} \in J_1.$$

Let us complete definition of $\bar{r}$ and $\bar{u}$ for indexes i of $E - \{\mu\}$: since $n^2 \leq |y_i| \leq 2n^2 + 1$, it is possible to find r_i and u_i such that

$$|r_i| = n^2$$

$$|u_i| \leq n^2 + 1$$

$$y_i = r_i + u_i.$$

At last, for indexes i not belonging to E we take

$$r_i = y_i \quad \text{which is} \quad \leq n^2$$

$$u_i = 0.$$

It is clear that

$$\bar{y} = (2n^2 + 1)\bar{q} + \bar{r} + \bar{u}$$

and that

$$\bar{r} \in H_{n^2}.$$

We are left to show that $\bar{u} \in J_{n^2}$. But

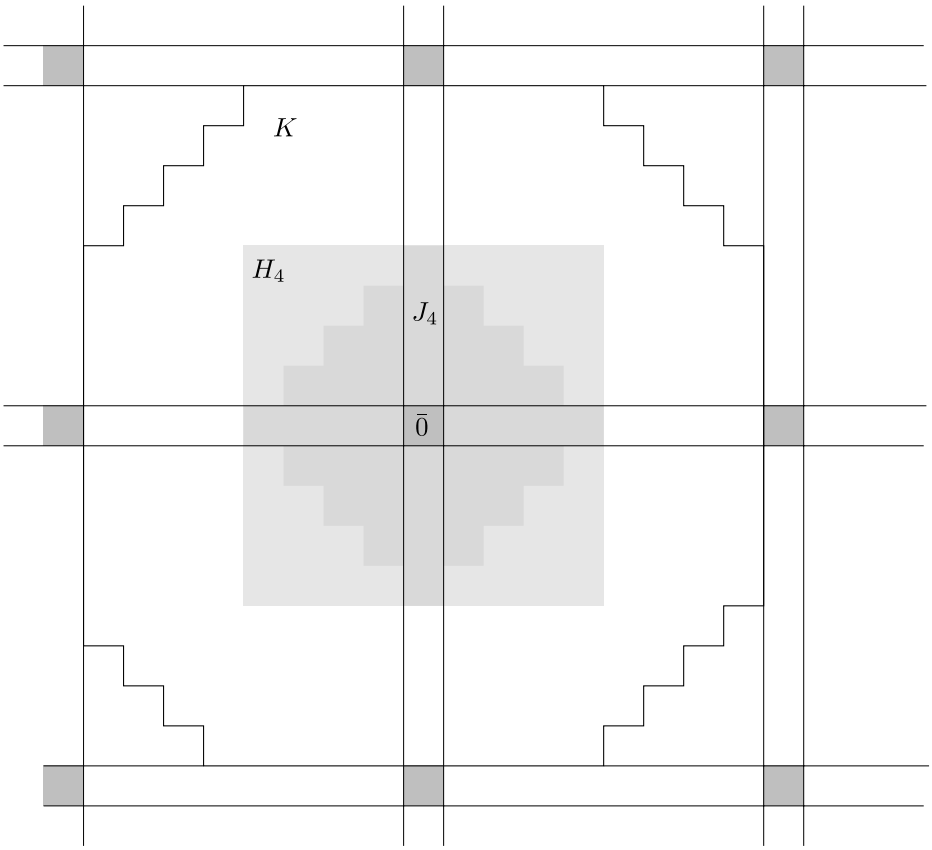
$$|\bar{u}| = \sum_{i=1}^{n} |u_i| = \sum_{i \in E - \{\mu\}} |u_i| = \sum_{i \in E - \{\mu\}} |y_i - r_i| = \sum_{i \in E - \{\mu\}} (|y_i| - n^2) \leq \sum_{i \in E} (|y_i| - n^2).$$

If this last sum is not greater than n^2 we are done. If it is not the case, then we have

$$n^2 < \sum_{i \in E} (|y_i| - n^2) \leq \sum_{i \in E} (|y_\mu| - n^2) \leq n(|y_\mu| - n^2)$$

hence

$$|y_\mu| - n^2 > n.$$



$\psi(\mathbb{Z}^2)$

Fig.7 : n=2

But

$$y_i = a_i + b_i + c_i$$

$$|y_i| \leq n^2 + |b_i| + |c_i|$$

$$|y_i| - n^2 \leq |b_i| + |c_i|$$

so

$$\sum_E (|y_i| - n^2) \leq \sum |b_i| + \sum |c_i| \leq n^2 + n.$$

As

$$|y_\mu| - n^2 > n$$

we have

$$|\bar{u}| = \sum_{E-\{\mu\}} (|y_i| - n^2) = \sum_{E} (|y_i| - n^2) - (|y_\mu| - n^2) < n^2$$

so $\bar{u} \in J_{n^2}$. (See Figure 7).

Proof of the proposition is thus achieved.

For the case of dimension $n = 2$, Cole's choice is

$$K = H_4 + J_4$$

$$\psi(\bar{z}) = 9\bar{z}$$

whilst the smallest possible values would give

$$K = H_1 + J_1$$

$$\psi(\bar{z}) = 3\bar{z}.$$

It is easy in this case to draw the figures and verify that both solutions are good.

Combining this proposition with the second result, we see that for any c.a there exists an equivalent c.a with stencil J_1.

As a conclusion, with no loss in generality, we may limit ourselves to considering only c.a's with stencil J_1.

10.5 Languages recognized by c.a's in real time

These languages are recursive languages, this is a consequence of Church's thesis. A direct check is in fact easy.

We shall now develop some useful tools

10.5.1 Study of the syntactical equivalences

Let us first recall their definitions, which were already given in 3.2.3. For any language L on alphabet X we define a family of equivalence relations in X^* as follows.

Syntactic equivalence mod L, denoted $\equiv^L$, for words of X^* is defined by

$$w \equiv^L w' \quad \text{if and only if} \quad \forall u \in X^* \quad wu \in L \Leftrightarrow w'u \in L.$$

In the same way we define, for any integer k, syntactic equivalence $\equiv_k^L$ by

$$w \equiv_k^L w' \quad \text{if and only if} \quad \forall u \in X^* \text{ such that } |u| \leq k \quad wu \in L \Leftrightarrow w'u \in L.$$

It may be handy to define also equivalence $\sim_k^L$

$$w \sim_k^L w' \quad \text{if and only if} \quad \forall u \in X^* \text{ of length } |u| = k \quad wu \in L \Leftrightarrow w'u \in L.$$

Note that

$$w \equiv^L w' \Rightarrow w \equiv_k^L w' \Rightarrow w \sim_k^L w'$$

and

$$\equiv_k^L \; = \; \cap_{h=0}^k \sim_h^L .$$

Another presentation of these equivalences can be given, using the following mapping from words to languages :

$$w \mapsto w^{-1}L = \{u \mid wu \,\epsilon\, L\} \; : \; X^* \longrightarrow \mathcal{P}(X^*).$$

Then

$$\begin{array}{lll} w \equiv^L w' & \Longleftrightarrow & w^{-1}L = {w'}^{-1}L \\ w \sim_k^L w' & \Longleftrightarrow & w^{-1}L \cap X^k = {w'}^{-1}L \cap X^k \\ w \equiv_k^L w' & \Longleftrightarrow & w^{-1}L \cap X^{\leq k} = {w'}^{-1}L \cap X^{\leq k} \end{array}$$

where

$$X^{\leq k} = \cup_{i=0}^k X^i.$$

Using this presentation, we may easily bound the indexes of the syntactical equivalences $\sim_k^L$ and $\equiv_k^L$. In the general case when the alphabet X has at least 2 letters :

$$\alpha = |X| \geq 2$$

the number of words of length k is α^k, and the number of words in $X^{\leq k}$ is

$$1 + \alpha + \ldots + \alpha^k = \frac{\alpha^{k+1} - 1}{\alpha - 1}.$$

The number of sets of words of length k is 2^{α^k}, and the number of sets of words of length at most k is $2^{\alpha^k + \ldots + 1}$. As a consequence, for any language L, the syntactic equivalences $\sim_k^L$ and $\equiv_k^L$ have finite indexes bounded by

$$\begin{array}{lll} index(\sim_k^L) & \leq & 2^{\alpha^k} \\ index(\equiv_k^L) & \leq & 2^{\alpha^k + \ldots + 1}. \end{array}$$

It is easy to show that these bounds are actually reached for some languages. Indeed, let

$$L_1, L_2, \ldots, L_N$$

be any sequence of languages in X^*, and a and b 2 letters of X. Then the language

$$L = aL_1 \cup baL_2 \cup \ldots \cup b^{N-1}aL_N$$

is such that

$$a^{-1}L = L_1\ ,\ \ (ba)^{-1}L = L_2\ ,\ \ldots\ ,\ \ (b^{N-1}a)^{-1}L = L_N.$$

If the languages L_i are all distinct, the N words

$$a\ ,\ ba\ ,\ \ldots\ ,\ b^{N-1}a$$

are thus in different classes, so that the number of classes is at least N. The language L obtained by this way from the sequence of all sets of words of length k has index of $\sim_k^L$ equal to 2^{α^k}. The language L obtained by this way from the sequence of all sets of words of length at most k has index of $\equiv_k^L$ equal to $2^{\alpha^k+\ldots+1}$.

These considerations will emphasize the interest of the following criterion of Cole, which, for c.a-recognizable languages L, bounds the indexes of the $\equiv_k^L$ relations by an exponential of a polynomial in k.

10.5.2 Cole's criterion in dimension n

We have already proved this criterion in section 3.2.3 in a very restricted frame : dimension 1, neighbourhood H_1, and for a half-line only.

We give it now in the most general circumstances.

Criterion 10.5.1 *If a language L is recognizable in strictly real time by some n-dimensional c.a $\mathcal{A}$, then there exists h such that, for all k, equivalence $\equiv_k^L$ has index (number of classes) less than*

$$h^{(2k+1)^n}.$$

Note : we point out that, for $n = 1$ the polynomial is $(2k+1)$ and not $(k+1)$ as in section 3.2.3. The reason is that we now deal with a complete line instead of a half-line only.

Indeed, if L is recognized in strictly real time by an n-dimensional c.a $\mathcal{A}$, it is recognizable in strictly real time by an n-dimensional c.a $\mathcal{A}'$ with stencil H_1. Number h of the criterion will be the number of states of $\mathcal{A}'$.

For any k, if two words w and w' belong to two distinct classes of $\equiv_k^L$, there

exists a word u of length at most k such that wu is recognized and $w'u$ is not : this results from the definition of relations $\equiv_k^L$.

But, state of cell $\bar{0}$ just after input of wu, which is accepting (/respectively of $w'u$, which is rejecting) depends only on u and states on

$$\bar{0} + \underbrace{H_1 + \ldots + H_1}_{|u| times} = |u|H_1 \subseteq kH_1 = H_k$$

just after input of w (/resp. w').

Configurations of states on H_k after input of w and w' should therefore be different. But as H_k has $(2k+1)^n$ cells, there are only

$$h^{(2k+1)^n}$$

such configurations. So there can't be more than this number of classes in $\equiv_k^L$.

It is interesting to note that this criterion may be considered a generalization to c.a's of one of the conditions in Nerode's theorem [44] for languages recognizable by finite automata (in strict real time goes in this case without saying). Indeed, finite automata are the extreme case of c.a's of dimension 0. If we put $n = 0$ in the statement of the above criterion, we obtain that there exists some constant h such that, for all k, equivalence $\equiv_k^L$ has at most h classes. This implies that equivalence $\equiv^L$ has index at most h too, and there we have condition (iii) of Nerode's theorem.

However, Nerode's condition (iii) was not only a necessary condition, it was also sufficient for a language to be recognizable by some finite automaton.

10.5.3 Converse of this criterion is false

Let K be a set of integers which is not recursive, and let $\Lambda = \{1^n \mid n \in K\}$, which is then a non recursive language.

Let us count the equivalence classes of $\equiv_k^\Lambda$: two words w and w' belong to two different classes if and only if there exists a word u of length at most k such that $wu \in \Lambda$ and $w'u \notin \Lambda$. But the alphabet is $\{1\}$, and there are only $k+1$ words with length less than k : 1^0,1, ... , 1^k. So w and w' are in two different classes if and only if the two $k+1$-tuples of 0's and 1's

$$(1_\Lambda(w), 1_\Lambda(w1), \ldots, 1_\Lambda(w1^l), \ldots, 1_\Lambda(w1^k))$$

$$(1_\Lambda(w'), 1_\Lambda(w'1), \ldots, 1_\Lambda(w'1^l), \ldots, 1_\Lambda(w'1^k))$$

where 1_Λ is the characteristic function of Λ, are different. So there can't be more than 2^{k+1} classes.

As a consequence there exists $h = 2$ such that, for any k, $\equiv_k^\Lambda$ has a number of classes less than

$$2^{(2k+1)^1}.$$

If the converse of Cole's criterion were true, Λ should be recognized by some linear c.a, which is not the case because it is not even recursive.

10.5.4 Families formed by these languages

In the large family of languages recognized in strictly real time we can distinguish sub-families :

$\mathcal{L}_n$ the family of languages recognized (in real time) by some n-dimensional c.a

$\mathcal{L}_n(X)$ the family of languages recognized (in real time) by some n-dimensional c.a with alphabet X.

We insist that X is finite, and the only interesting information about it is its number of elements.

We are interested in the closure properties of these families for several operations, some of them unary : complement, operation *

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup \ldots \cup L^m \cup \ldots ,$$

(where ε is the empty word), product with X^*, reversal,

others binary : union, intersection, set difference , product.

Let us first observe that if L is a language on alphabet X, then, for every $X' \supseteq X$

$$L \in \mathcal{L}_n(X) \Leftrightarrow L \in \mathcal{L}_n(X').$$

In fact it is clear that a c.a on X' recognizing L still recognizes it if we limit its input alphabet to X, and it is easy to modify a c.a with alphabet X recognizing L into a c.a with alphabet X' : we can decide that any letter of $X' \setminus X$ acts like some determined letter of X, and in addition deposits on cell $\bar{0}$ a definite mark which makes any state rejecting. So finally there is no real difference.

This comment allows us to consider all operations, unary or binary, as operating inside families of languages having a determined alphabet : we thus perform unions, products, ... only with languages on the same alphabet.

Thus, if the fact that all $\mathcal{L}_n(X)$ for any X are closed for some operation on languages implies that $\mathcal{L}_n$ too, the converse is also true. So it is equivalent to state for some determined operation, that $\mathcal{L}_n$ is closed or that all the $\mathcal{L}_n(X)$ are.

On the other hand, should some family $\mathcal{L}_n(X)$ not be closed for some operation then family $\mathcal{L}_n$ would not be either. (In this case, for any larger alphabet X', $\mathcal{L}_n(X')$ is not closed either).

10.5.5 Closure properties of the $\mathcal{L}_n$ families

$\mathcal{L}_n$ is closed under boolean set theoretical operations

If L is recognized by the n-c.a $\mathcal{A}$, its complement is recognized by the c.a obtained by exchanging accepting and rejecting states. If L_1 is recognized by $\mathcal{A}_1$ and L_2 by $\mathcal{A}_2$, the product c.a (resulting from the simultaneous working of the two), recognizes $L_1 \cap L_2$ if accepting states are the couples of two accepting states, and recognizes $L_1 \cup L_2$ if its accepting states are the couples where at least one of the two states is accepting, while $L_1 \setminus L_2$ is the intersection of L_1 and the complement of L_2.

It is not known wether $\mathcal{L}_n$ is closed under operation * or not

All $\mathcal{L}_n(X)$ are closed under the right product with X^*

Indeed, if L is recognized by some n-c.a $\mathcal{A}$, LX^* is recognized by the n-c.a obtained by deciding that any accepting state definitely marks cell $\bar{0}$ with a sign that makes any state accepting.

If $|X| \geq 2$, no family $\mathcal{L}_n(X)$ is closed for the left product with X^*

Of course, if X has only a single letter x, then $\{x\}^*L$ is the set of all words on $\{x\}$ of length greater than the length of the smallest word of L and obviously belongs to $\mathcal{L}_n(\{x\})$ for any n.

This case is not interesting, so we suppose that X has at least two letters.

As in chapter 3, P_3 will denote the set of palindromes on alphabet X having length 3 at least.

In section 3.4.3 we proved that P_3 is recognized (in strictly real time) by some c.a on $\mathbb{N}$. We shall easily admit, before we prove it later on (10.5.6), that consequently it belongs to $\mathcal{L}_n(X)$ for any n.

As for X^*P_3, we proved in 3.4.5 that it does not belong to $\mathcal{L}_1(X)$, using Cole's criterion in dimension 1.

We had then proved, concerning equivalence $\equiv_k^{X^*P_3}$, that, m being any integer,

$$\text{for} \quad k = 2m + 3$$

this equivalence had at least

$$2^{2^m} \text{ classes .}$$

But, for any n and any h, it is clear that, for m large enough

$$2^m > log_2 h.(4m + 7)^n$$

hence

$$2^{2^m} > h^{(4m+7)^n} = h^{(2k+1)^n}.$$

Cole's criterion in dimension n fails, so that X^*P_3 cannot belong to $\mathcal{L}_n(X)$.

Families $\mathcal{L}_n(X)$ are not closed, neither under the product, nor for the reversal

Indeed P_3 belongs to $\mathcal{L}_n(X)$ and so does X^* (recognized by any n-c.a where all states are accepting). Yet product X^*P_3 is not in $\mathcal{L}_n(X)$.

P_3X^* belongs to $\mathcal{L}_n(X)$, but its reverse X^*P^3 does not

These counter-examples demand that X has at least 2 letters. If X has only one single letter, the reverse operation is the identity, so that closure for reversal is trivial.

As far as we know, the question whether $\mathcal{L}_n(\{x\})$ is closed for the product remains open.

No $\mathcal{L}_n$ contains the family of context-free languages

Observe that language X^*P_3 is context-free and is not recognized (in strictly real time) by any c.a of any dimension whatever.

10.5.6 Power of c.a's increase with their dimension

More precisely, real-time recognition increases with dimension.

It is intuitively clear that an (n+1)-c.a has more possibilities than an n-c.a, so that $\mathcal{L}_n \subseteq \mathcal{L}_{n+1}$. However we shall detail the argument.

Let us first notice, that for any c.a we can suppose (if necessary a second quiescent state may be introduced), that the original quiescent state can only maintain itself, so that it never appears in the active area of the s.t.d.

Let us first show that a c.a on $\mathbb{N}$ (of scope 1) can be modified in a c.a on $\mathbb{Z}$ (of scope 1) : for this we complete it with cells -1,-2,..., in quiescent state at time 0, we complete transition function by

$$\forall q \in Q \quad \delta'(e, e, q) = e$$

so that cells having a negative number constitute a quiescent area, and we adapt the transition by defining

$$\delta'(e, q, d) = \delta(\beta, q, d) \quad \forall q, d \in Q$$

so that this quiescent area replaces the original left border. Clearly the $\mathbb{Z}$-a.c obtained accomplishes the same task as the $\mathbb{N}$-c.a, in the same time.

Let us now prove that any n-c.a with stencil J_1^n (J_1 stencil of $\mathbb{Z}^n$) can easily be modified in an $(n+1)$-c.a with stencil J_1^{n+1} (J_1-stencil of $\mathbb{Z}^{n+1}$) :

- first we place it in $\mathbb{Z}^{n+1}$ by giving all cells a $n+1$-st coordinate of value 0
- all cells with non zero $n+1$-st coordinate are set in quiescent state at time 0
- if C is a J_1^{n+1}-configuration of states of Q we define

$$\delta'(C) = \delta(C \text{ restricted to } J_1^n)$$

which is to say that states of the cells which find themselves in the part of its J_1^{n+1}-neighbourhood which is not in its J_1^n-neighbourhood have no influence on a cell. This amounts in fact to keeping the ancient J_1^n-neighbourhood.

It is clear that these modifications bring no change to the run of the c.a nor to the running time.

We can thus conclude

$$\mathcal{L}_n(X) \subseteq \mathcal{L}_{n+1}(X)$$

$$\mathcal{L}_n \subseteq \mathcal{L}_{n+1}$$

10.5.7 This power increases strictly

Now we want to show that, for any n,

$$\mathcal{L}_n(X) \neq \mathcal{L}_{n+1}(X)$$

by producing a language of $\mathcal{L}_{n+1}(X)$ not belonging to $\mathcal{L}_n(X)$.

This happens to be a rather difficult result.

$\boxed{\mathcal{L}_0 \neq \mathcal{L}_1}$

Let us not disregard first proving that $\mathcal{L}_0 \neq \mathcal{L}_1$. A c.a of dimension 0, which extends in no direction, is reduced to cell 0. This cell can have no neighbour, its state depends only on its proper state and the input at preceding time-step. If it works in strictly real time, we recognize it is a classical finite automaton. So family $\mathcal{L}_0$ is the family of regular (rational) languages. Family $\mathcal{L}_1$, which contains non regular languages ($\{a^n b^n\}$, the sets of palindromes, see chapter 3), and even non context-free languages (the set of square words), is considerably more large.

$\boxed{\mathcal{L}_1 \neq \mathcal{L}_2}$

This is the only case in which we shall detail the proof entirely.

Definition of language L

Let Λ be the language on alphabet $\{0, 1, a\}$ of all words

$$w_p a \dots a w_m a \dots\dots a w_0$$

where the w_i's are non-empty words on $\{0, 1\}$, that is words of

$$W = \{0, 1\}^+ = \{0, 1\}^* \setminus \{\varepsilon\}$$

where ε is he empty word.

And let L be the language on {0,1,a,d} of words

$$w_p a \dots a w_m a \dots\dots a w_0 d^m \tilde{w}_m$$

formed of one word of Λ and one suffix $d^m \tilde{w}_m$, where $1 \leq m \leq p$ and $\tilde{w}_m$ is the reversal of one of the words of W which compose the first part, precisely the one of index m.

L does not belong to $\mathcal{L}_1$

We shall show that Cole's criterion fails :

$$\forall h \quad \exists k \quad \text{such that} \quad index(\equiv_k^L) > h^{2k+1}.$$

Let p be an integer that we reserve ourselves to choose later. Let us consider the particular words of Λ

$$w_p a \dots a w_m a \dots\dots a w_0 \quad \text{with} \quad w_i \in \{0,1\}^p \quad \text{and} \quad w_0 = 0^p \text{ fixed .}$$

There are 2^{p^2} of these words. If two such words ω and ω' are distinct, there must be some m between 1 and p such that

$$w_m \neq w_{m'}.$$

From this we get

$$\omega d^m \tilde{w}_m \in L \quad \text{while} \quad \omega' d^m \tilde{w}_m \notin L$$

with

$$|d^m \tilde{w}_m| \leq p + p = 2p.$$

Thus for $k = 2p$, ω and ω' are not equivalent for relation $\equiv_k^L$. So this equivalence must have at least 2^{p^2} classes. If we have taken p large enough, for example

$$p > 5 \ \log h$$

then

$$p^2 > 5p \log h > (4p+1) \log h = (2k+1) \log h$$

$$2^{p^2} > h^{2k+1}$$

and

$$index(\equiv_k^L) > h^{2k+1}$$

so L does not belong to $\mathcal{L}_1$. Proving it belongs to $\mathcal{L}_2$ will need some work.

A simple and fundamental mechanism

We first present a linear c.a which harbours the very simple mechanism that we shall afterwards systematically use. It works with two elementary moves :

- if a cell contains 3 elements, the right one passes in the right neighbouring cell at next time-step
- if a cell contains only 1 element, at next time-step it draws the leftmost element from its right neighbour, providing this right neighbour is not empty.

(we can say that a cell is balanced only when it contains 2 elements, less is too little, more is too much, and it recovers balance by exchanging elements with the right neighbouring cell, see table of rules in Figure 8)

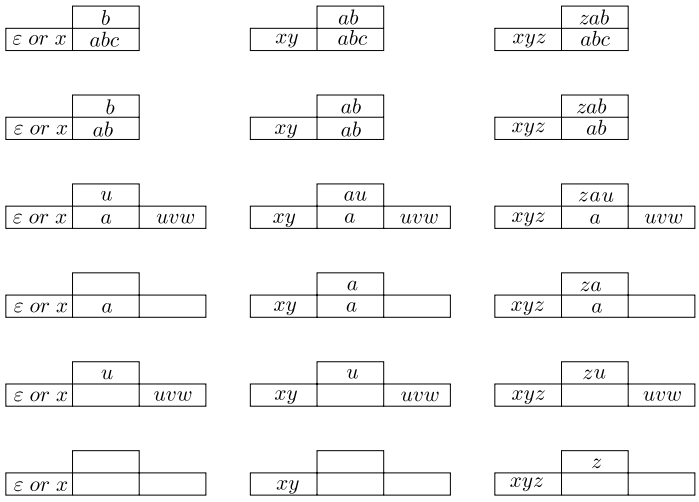


Fig.8 : table of rules

and a few simple principles :

- letters 0 and 1 enter cell 0 and pack there before being pushed on the right by the preceding mechanism

- input of letter d definitely modifies the working of cell 0 : after d has entered, input of some letter 0 or 1

  - on the same letter : destroys this letter

  - on a different letter : destroys this letter and sets a permanent rejecting state

  - on no letter : settles the input letter in cell 0

- the only accepting state is when cell 0 is empty and the rejecting state has not appeared.

Functioning of this c.a is illustrated in Figure 9. It is clear that it recognizes input words $wd\tilde{w}$ and rejects all other words.

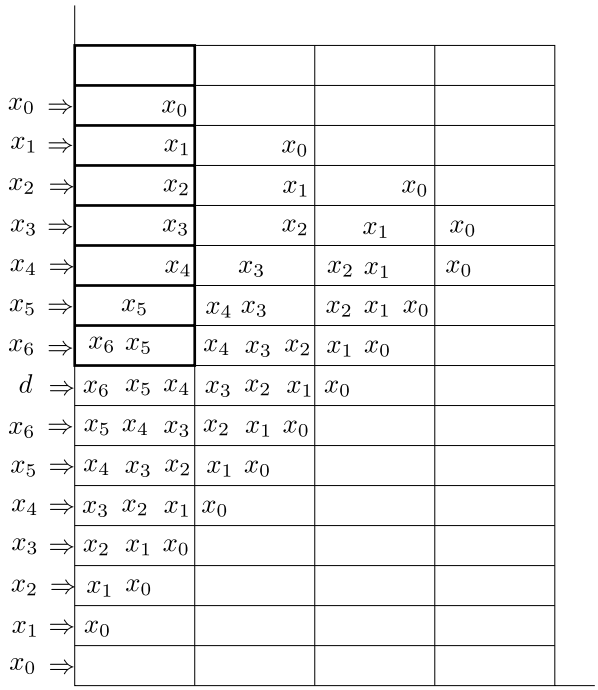


Fig.9

L is in $\mathcal{L}_2$

We shall now describe a c.a on $\mathbb{N}^2$ with stencil H_1 recognizing L. This c.a will work using the preceding mechanism, in two directions : in the first direction to string and unstring letters (at most 3 per cell), and in the second to stack and unstack words (at most 3 per cell), or rather pieces of length 3 of tracks intended to contain the words. Figure 10 will enlighten explanations.

How does the c.a work with some input $\omega = w_p a \dots a w_m a \dots a w_0 d_m w$?
(In Figure 10 the input word taken as example is

$$01010101a11a011101a00a1010dd101110 \quad)$$

The initial state is an impulse state, where cell 0 contains a first piece of track. Tracks will extend rightwards, piece by piece, at each time-step.

When letters 0 or 1 enter, they fill the (first) track beginning in cell 0, by pushing through the successive cells.

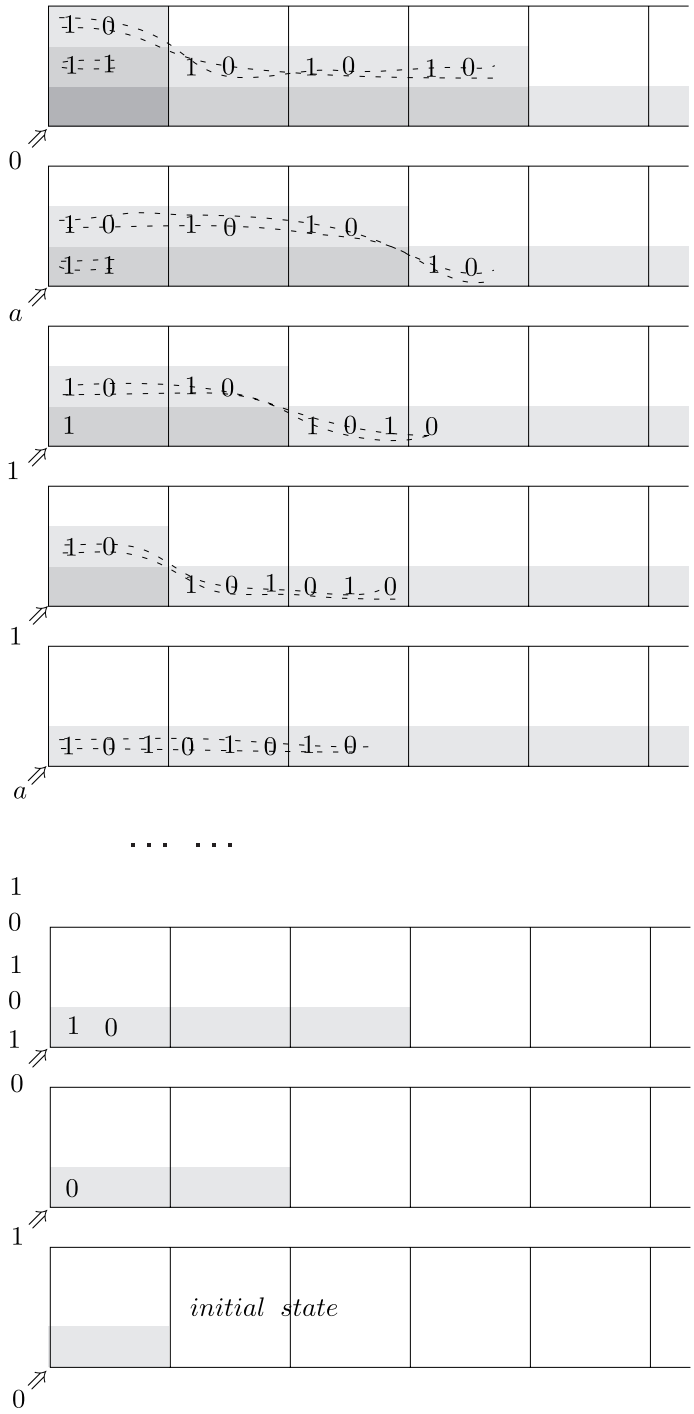


Fig.10-1

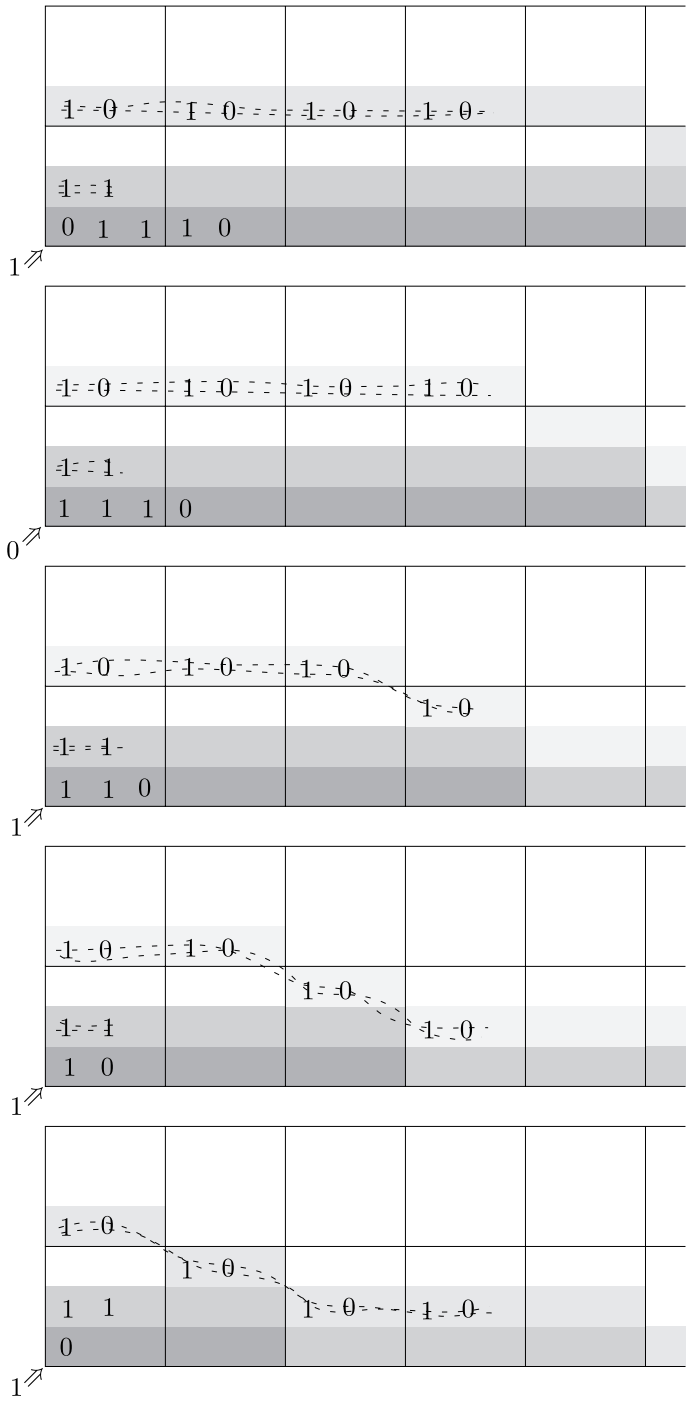


Fig.10-2

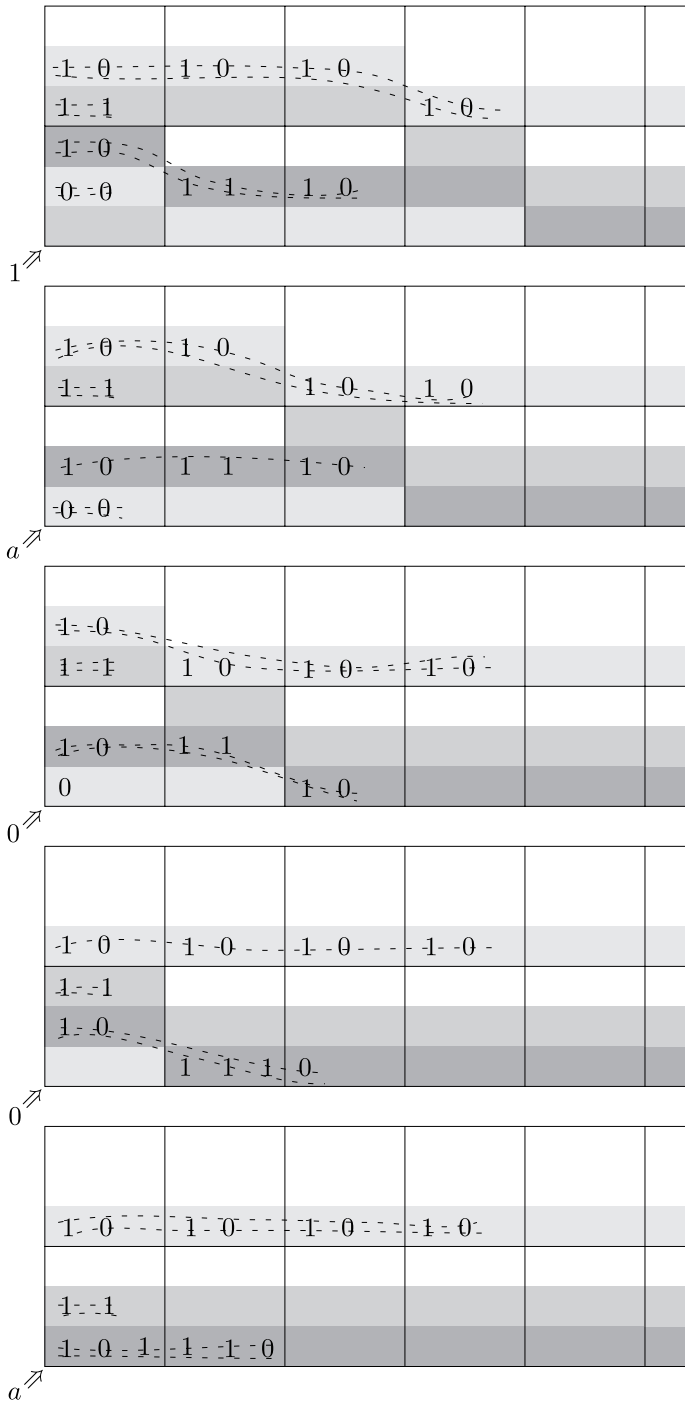


Fig.10-3

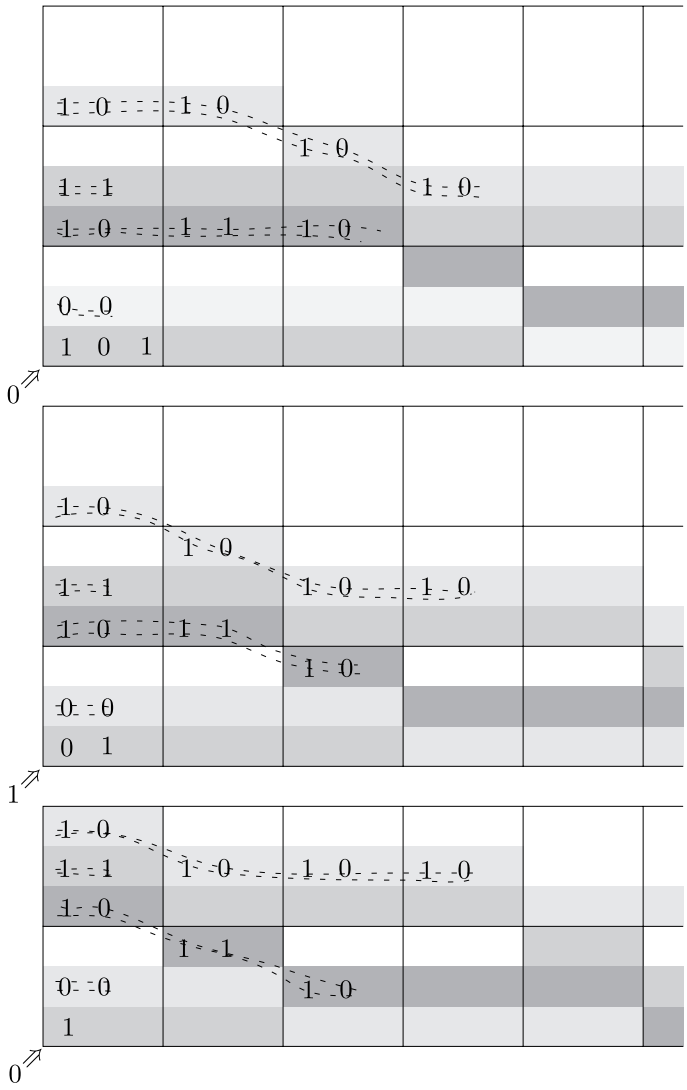


Fig.10-4

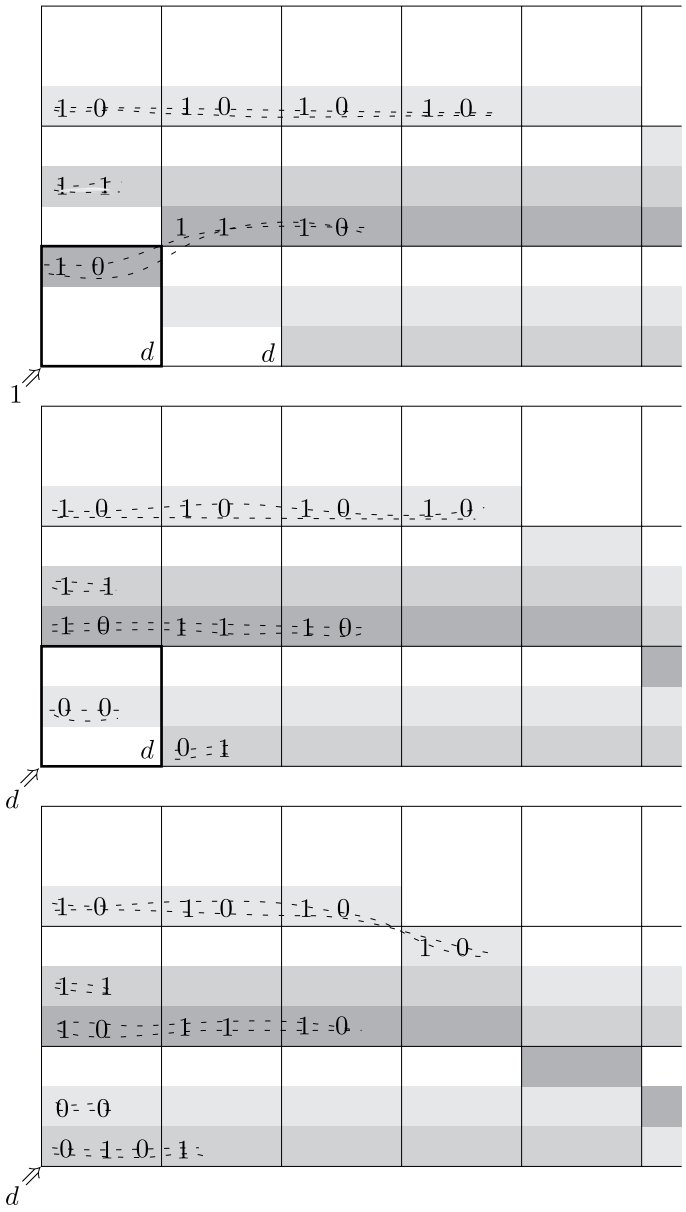


Fig.10-5

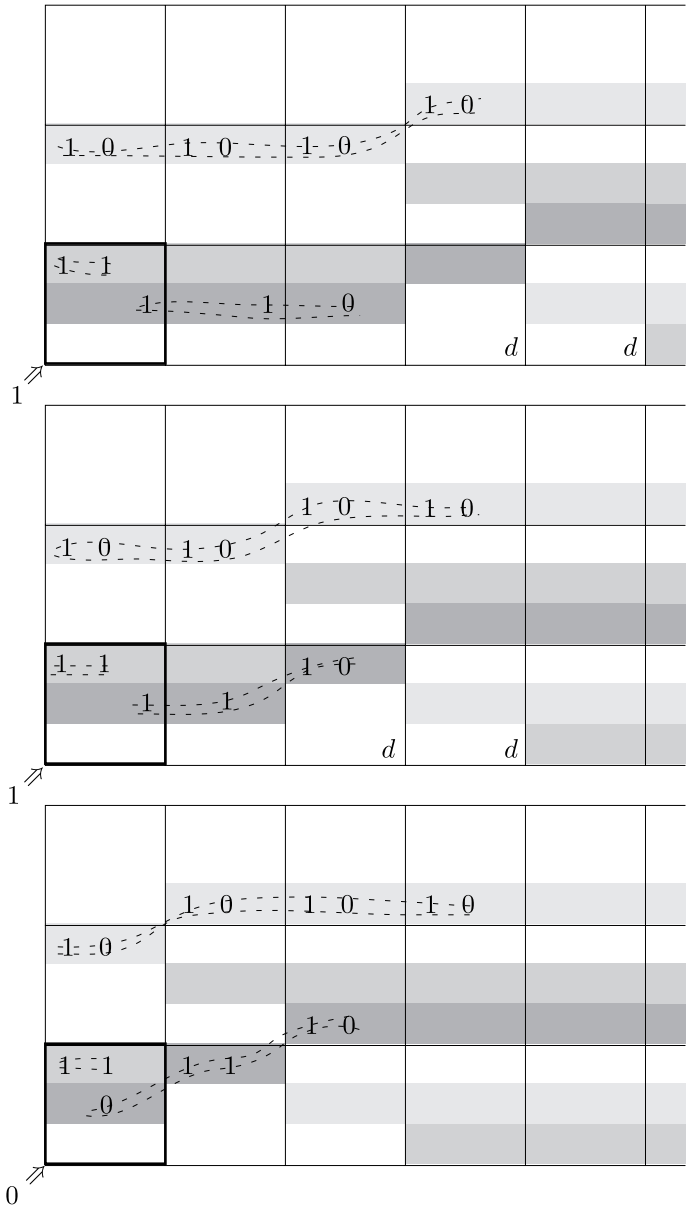


Fig.10-6

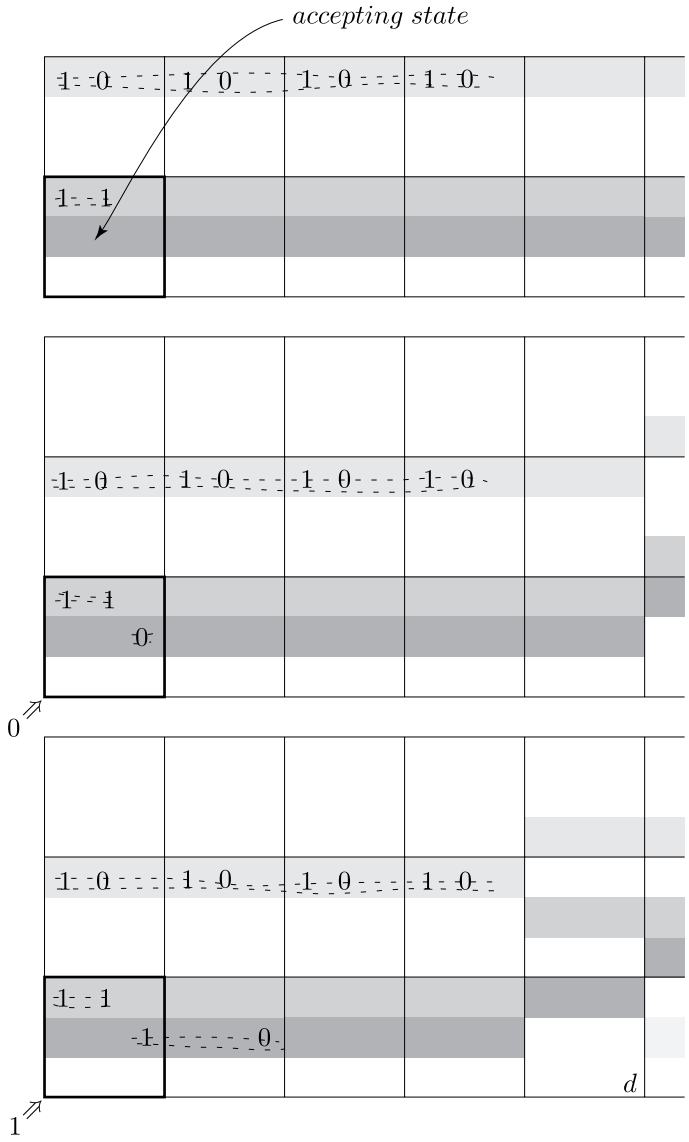


Fig.10-7

When input is an a-letter, a new piece of track is created, seed for a second track. This piece pushes the first piece of the first track (in the second direction). Now, as the second track progressively extends (in direction 1), all the first track will be progressively pushed (in direction 2), piece after piece, in the successive cells.

When a new word of 0 and 1 letters arrives, it will settle in this new track.

And so on. Now as soon as 3 pieces of track are stacked in a cell, one is pushed out in next cell, in direction 2. So the words w_p, ... , w_m, ... , w_0

progressively settle on successive tracks.

When the first d-letter arrives cell 0 undergoes a definite equal change, (it is now marked in black on Figure 10), which will concern the effect of the future entering of letters 0 and 1 : instead of settling, they will destroy the likes of them, or make cell 0 rejecting.

But before, each d-letter entering starts the destruction of a track (with the letters carried) : this destruction progresses as letter d advances in direction 1. And as this goes along, pieces of tracks become scarce in cells, so that more ancient tracks are progressively drawn back. As a result, when the last part w of the input word arrives, as many tracks as there were d-letters are (at least partly) destroyed. So the track beginning in cell 0 is the track carrying word w_m.

If $w = \tilde{w}_m$, word $\tilde{w}_m$ is progressively destroyed. If the accepting states are all states where the first track still present in cell 0 is empty, it is clear that all words of $\mathcal{L}_2$ are accepted.

We are left to show that all other words are rejected. If w is too short, equal to a prefix of $\tilde{w_m}$, the first track is not emptied when input ends. If w is too long, equal to $\tilde{w_m}$ with a suffix, the supplementary letters enter cell 0, which will not be empty at the end. If w differs from $\tilde{w_m}$, at least one of w's letters (which could be new a or d-letters) arrives on a different letter in cell 0, so cell 0 becomes rejecting. If there are more than p d-letters, there is no track left in cell 0, we decide that cell 0 becomes rejecting. If there is no d-letter, track in cell 0 cannot be emptied.

$\boxed{\mathcal{L}_n \neq \mathcal{L}_{n+1}}$

Definition of language L_n

Let us try and generalize preceding proof. And to start with, we must define the appropriate L_n language on alphabet $\{0, 1, a_1, \ldots a_n, d_1, \ldots d_n\}$.

Let us first present the sequence of languages Λ_1, Λ_2, ... , Λ_n : Λ_1 is the language formed by the products w of W separated by a_1's, Λ_2 is the language of products of words of Λ_1 separated by a_2's, and so on, Λ_n is the language of products of words of Λ_{n-1} separated by a_n's.

L_n is then obtained by extending the words of Λ_n by suffixes of form $d_n^{m_n} \ldots d_1^{m_1} \tilde{w}$, $m_i \geq 1$, convenient, in the sense that they must correspond to one of the w's composing the word of Λ_n : this means that, by destroying the m_n first (from the right) words of Λ_{n-1}, then (in what is left) the m_{n-1} first words of Λ_{n-2}, ... , and at last the m_1 first words of W, we find a w, and that the end of the suffix is precisely is reversal.

L_n does not belong to $\mathcal{L}_n$

We want first to show that L_n is not in $\mathcal{L}_n$ by showing that

$$\forall h \quad \exists k \quad \text{such that} \quad index(\equiv_k^L) > h^{(2k+1)^n}.$$

For some fixed p, which we shall choose later, we shall consider a particular subset of Λ_n : the one we obtain with words of W of length p to start with, by gathering $p+1$ of these words (the first on the right being 0^p) with p a_1-letters, then $p+1$ such words with p a_2-letters, and so on, finishing by gathering $p+1$ words of Λ_{n-1} (the first one having only 0's) with p a_n-letters. Let us give one or two examples :

with $n = 2, p = 3$

$$110a_1100a_1010a_1000 \;\; a_2 \;\; 111a_1100a_1011a_1000 \;\; a_2$$

$$111a_1010a_1111a_1000 \;\; a_2 \;\; 000a_1000a_1000a_1000$$

et avec $n = 4, p = 1$

$$1a_10a_20a_10 \;\; a_3 \;\; 1a_11a_20a_10 \quad a_4 \quad 0a_10a_20a_10 \;\; a_3 \;\; 0a_10a_20a_10$$

The number of these words is $2^{p^{n+1}}$. But if 2 such words, ω and ω' are distinct, they have at least one of their w factors which are different. Then there exists a sequence $m_n, m_{n-1}, \ldots, m_1$, $1 \leq m_i \leq p$ and a word w such that

$$\omega d_n^{m_n} \ldots d_1^{m_1} \tilde{w} \in L_n$$

$$\omega' d_n^{m_n} \ldots d_1^{m_1} \tilde{w} \notin L_n.$$

As

$$|d_n^{m_n} \ldots d_1^{m_1} \tilde{w}| \leq (n+1)p$$

the two words ω and ω' are in different classes of equivalence $\equiv_k^{L_n}$ if we take

$$k = (n+1)p.$$

Relation $\equiv_k^{L_n}$ will so have at least $2^{p^{n+1}}$ classes. But, for any h, we can choose p so that

$$2^{p^{n+1}} > h^{(2k+1)^n}$$

that is

$$p^{n+1} > (2k+1)^n . \log h = [(2n+2)p+1]^n . \log h$$

for this it suffices that

$$p^{n+1} > [(2n+3)p]^n . \log h = (2n+3)^n p^n . \log h$$

or

$$p > (2n+3)^n . \log h$$

That is how we choose p at the very beginning .

L_n is in $\mathcal{L}_{n+1}$

Initial state of cell 0 is the bud for a track that will extend in direction 1. Each a_1-letter will create the bud for a new track. The successive tracks progressively reconstruct a dimension 2-subspace. Each a_2-letter will create the bud for a new plane layer, that is a unit square which will extend simultaneously in directions 1 and 2, or better expressed, a unit slab around which (stencil is H_1) new slabs are generated at each time-step, ... , each a_n-letter will create the bud for a new n-dimensional subspace, that is an n-dimensional unit cube which will extend in the n first directions simultaneously.

Words of W slip in the tracks, then words of Λ_1 in the plane layers, and so on, words of Λ_n take place in the subspaces of dimension n, whose piling up reconstructs the entire $n+1$-dimensional space. Now each d_n-letter will destroy a dimension n subspace as it advances in the n first directions, each d_{n-1}-letter a dimension $(n-1)$ subspace as it advances in the $n-1$ first directions, ... , finally each d_1-letter destroys a track as it advances in direction 1. Finally word w will meet its reversal starting in the first track of the first plane ... of the first n-subspace starting in cell 0, and it will exactly destroy this word and empty the track.

This is far from a proof. But we hope it helps understand the part played by the successive supplementary dimensions to recognize the successive L_n-languages, and that this can convince one that each increase of the dimension makes new languages recognizable.

Chapter 11

Synchronization of a pair with delay

This chapter is based on Mazoyer's work in [38]

Up to now, we have supposed that communication between adjacent cells needs no delay, so that signals can propagate at speed 1. Now we introduce delays in the intercommunications between cells in linear c.a's. We first consider lines of two cells. The case of lines with an arbitrary number of cells is the subject of next chapter.

But we shall always suppose that the delay for communication between two cells is symmetric.

11.1 Introduction of the notion of delay

11.1.1 Preliminary comment on couples and pairs

In a line each automaton has 2 I/O ports, a left one and a right one. Using the fictitious border state β as in 1.1.3 and following chapters, the transition function for the 2 cells of a line of 2, that we may call a couple, is thus

$$< A_1, t+1 >= \delta(\beta, < A_1, t >, < A_2, t >) = \delta_1(< A_1, t >, < A_2, t >)$$

$$< A_2, t+1 >= \delta(< A_1, t >, < A_2, t >, \beta) = \delta_2(< A_2, t >, < A_1, t >).$$

In a pair, we do not distinguish a left and a right cell, the 2 automata have one single I/O port, and the same transition function

$$< A_i, t+1 >= \delta(< A_i, t >, < A_j, t >).$$

It seems that in the first case we may have 2 different transition functions for the 2 cells, and not in the second case. But the couple is reduced to a pair if

$\delta_1 = \delta_2$, and the pair may become a couple if we distinguish states for A_1 from states for A_2.

Thus, there is no fundamental difference between a pair and a couple.

11.1.2 The notion of delay

In the case of a pair communicating with delay τ, the transition function has arity 2 and

$$< A_i, t+1 >= \delta(< A_i, t >, < A_{3-i}, t-\tau >) \text{ where } i = 1, 2.$$

Now, how do we intend to visualize the evolution of such a pair ? Some imagination will help us here, moreover well suited to the large delays that we want to consider. We shall lift our eyes from cell clusters up to the skies, and dream of stars exchanging rays of light perceived long after they were sent. We shall represent automata interacting with delays according to this imagination, by attributing delays to distances that signals, sent by the automata according to their states, must cross. Thus, if the interaction delay between A_1 and A_2 is τ, we shall represent them at a distance of τ, that signals will cross at speed 1, one distance unit per time unit, and we shall represent the space-time diagram as shown in Figure 4 : the fictitious signal sent by state q leaves the (square representing the) automaton at the time when it appears, and is perceived by the other automaton at the time when it penetrates (the square representing) it.

We insist that the signals that we evoke here have nothing in common with the signals of preceding chapters. Those were trajectories of states from cell to cell, at various speeds. The ones we imagine now, cross, at constant speed, a void space that has no mathematical existence, only meant to help us visualize delays.

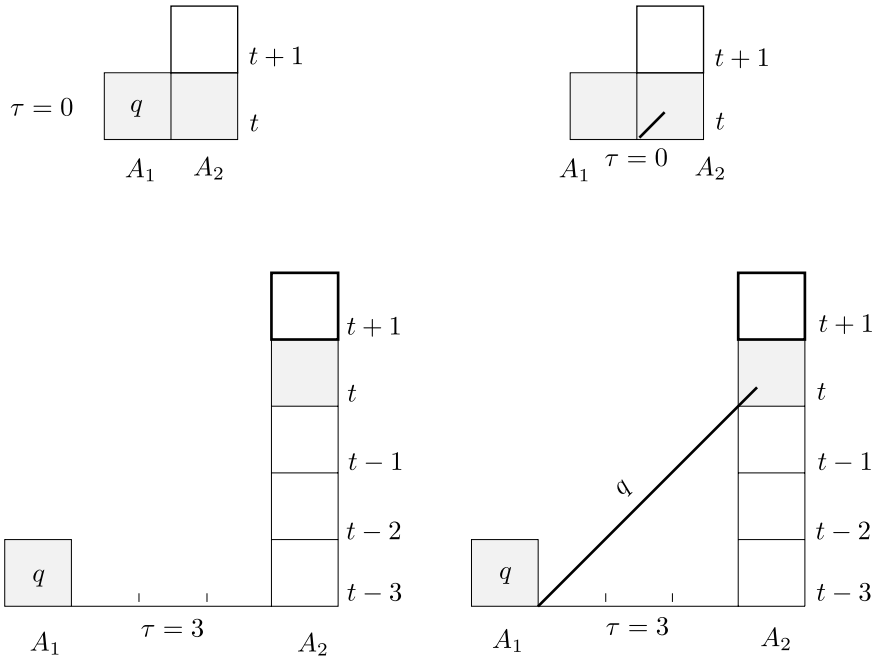


Fig.4

11.1.3 The clock signal

Let us first examine the simplest and quickest possible dialog between the two automata : a single (non quiescent) state s appears during one time-unit on one automaton, then on the other one as soon as it is perceived by it, the corresponding signal is reflected from one automaton to the other (Figure 5). State s appears on each automaton periodically, every $2\tau + 2$ time-steps, and takes time $\tau + 1$ to go from one automaton to the other. This elementary exchange gives the basical rythm for the functioning of our system. Its half-period being $\tau + 1$ and not τ, we shall define a new delay, the reaction delay

$$\Delta = \tau + 1$$

as opposed to the perception delay τ. Let us note that

$$\tau \geq 0 \qquad \text{and} \qquad \Delta \geq 1.$$

It is clear that our pair of automata can function only with such exchanges as we have just mentioned. In the sequel, we shall describe these exchanges by speaking of signals sent, received, reflected, rather than speaking of states being perceived.

We have not spoken of other states than s. In fact, two states are sufficient for the clock signal : s and the quiescent state.

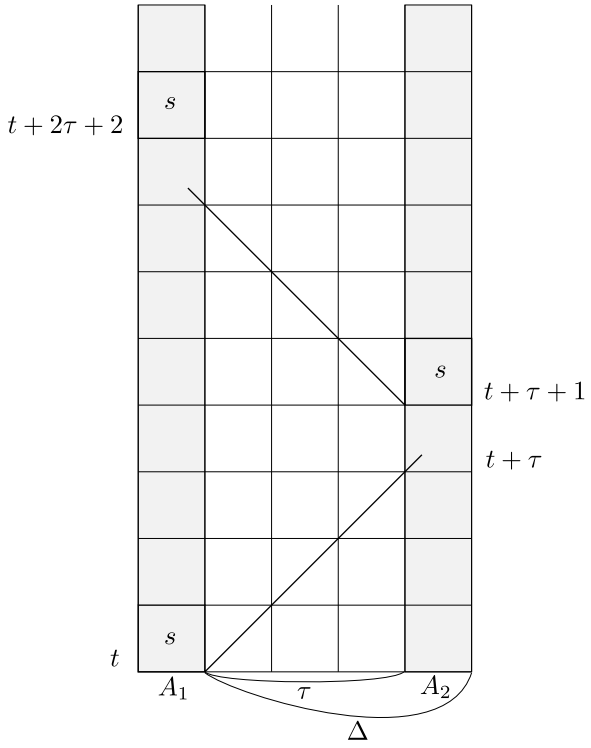


Fig.5

11.1.4 The synchronization problem

We consider the problem for the family of pairs with arbitrary delay, all integers being possible values for this delay.

We look for a set of states Q, containing the quiescent state, a general (commanding) state and a fire state, and a transition function δ such that, whatever the delay Δ, if the pair starts at time 0 in state

$$< A_1, 0 >= \text{general state} \qquad < A_2, 0 >= \text{quiescent state},$$

then, the fire state appears for the first time and simultaneously on the two automata, at a certain time depending on Δ (the synchronization time).

For such elements (set Q and transition δ) we shall use the name *synchronizing solution.*

It is quite imaginable that this could be a real problem : we may think of two satellites needing to be synchronized. It is probably impossible to synchronize them exactly by some external action. So they must achieve this only by

communicating with one another. This implies that the process is initiated by one only of the two automata.

11.1.5 Linear lower bounds for synchronization time

Let us suppose that (Q, δ) is a synchronizing solution. Initial state is state G on A_1, while A_2 is in quiescent state. Whatever the delay Δ, A_2 necessarily remains in this state up to time $\Delta - 1$ when it receives the signal sent by the general. So certainly

$$\forall \Delta \quad T(\Delta) \geq \Delta.$$

A_1 can receive no signal before time $2\Delta - 1$ as a non quiescent state may not appear on A_2 before time Δ. Therefore states of A_1 before time 2Δ do not depend on the second automaton. Should the fire state appear before time 2Δ, it would appear likewise if the second automaton were at any distance $\Delta' > \Delta$. If we consider $\Delta' > 2\Delta$, then for the pair with delay Δ' we would have $T(\Delta') < \Delta'$ (see Figure 6). From this we conclude that

$$\forall \Delta \quad T(\Delta) \geq 2\Delta.$$

Now states of A_2 from time Δ to time $3\Delta - 1$ for the pair with delay Δ, or from time Δ' to time $\Delta' + 2\Delta - 1$ for the pair with delay $\Delta' > \Delta$ (Figure 7), depend only on states of A_1 up to time $2\Delta - 1$, so they are exactly the same. Should $T(\Delta)$ be strictly less than 3Δ, then the fire state would appear among these states and $T(\Delta')$would then be less than $2\Delta'$. From this we conclude that

$$\forall \Delta \quad T(\Delta) \geq 3\Delta.$$

As we shall see in 11.3.4 and 11.4.5, this is the best possible linear lower bound and the above arguments cannot be extended further on.

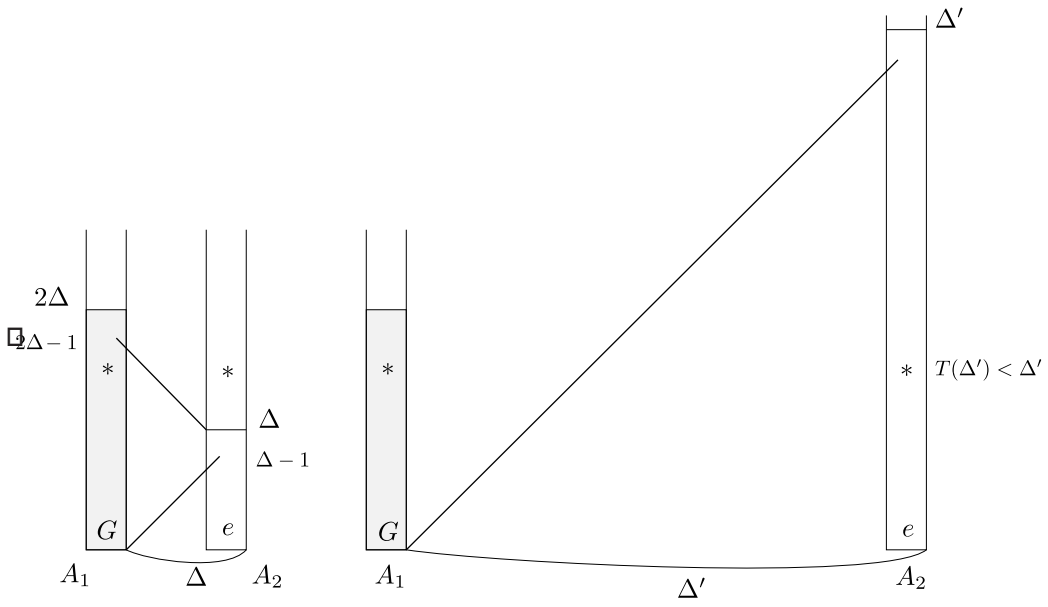


Fig.6

11.2 Transitions for successive divisions by 2

We define a system of states and transitions. This system is structured by the clock signal s, emitted by state G at time 0 (cf.11.1.3) and which determines on A_1 periods of time of 2Δ units. Period number 0 starts at time 0, and the i-th period starts at time $i2\Delta$.

The general idea is the following : during period 0, signals are emitted at regular time intervals of 2 units, and at each next period only one out of two (approximately) will be reflected. These signals thus get further apart and less frequent, they break the successive periods in larger and larger portions. Process ends when one single portion covers the entire period.

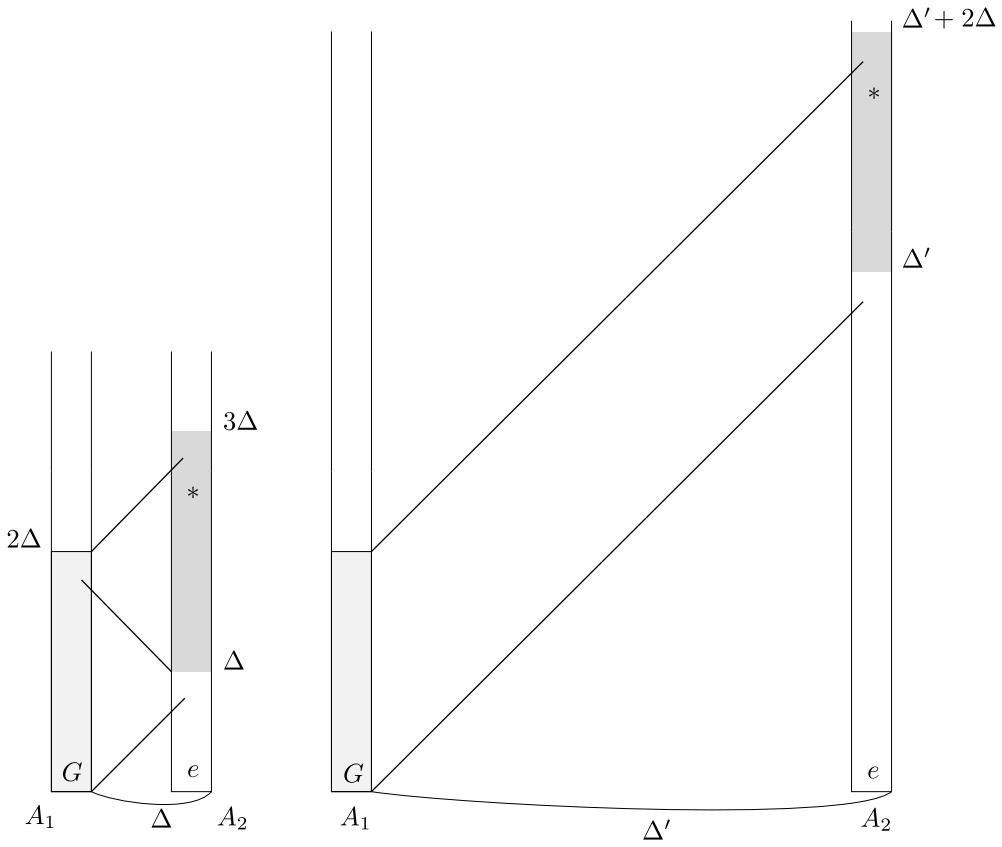


Fig.7

Let us examine this more precisely, and to begin with we set

$$\Delta = 2^p + \varepsilon_{p-1}2^{p-1} + \ldots + \varepsilon_1 2 + \varepsilon_0 \qquad p \geq 0$$

11.2.1 Period 0

This period is devoted to emitting signals ; state G, which emits the clock signal s, is followed by states

$$\bar{i} \ , \ p \ , \ \bar{p} \ , \ i \ , \ \bar{i} \ldots \qquad \text{till } s \text{ returns,}$$

p and i emitting r signals.

The 2Δ long period is thus divided in Δ sequences of pairs of successive states $G, \bar{i}$ or $i, \bar{i}$, and $p, \bar{p}$ (Figure 8). According to wether s returns on $\bar{p}$ or $\bar{i}$, we know that Δ is even or odd, i.e that $\varepsilon_0 = 0$ or 1.

The total number of signals emitted, s included, is Δ.

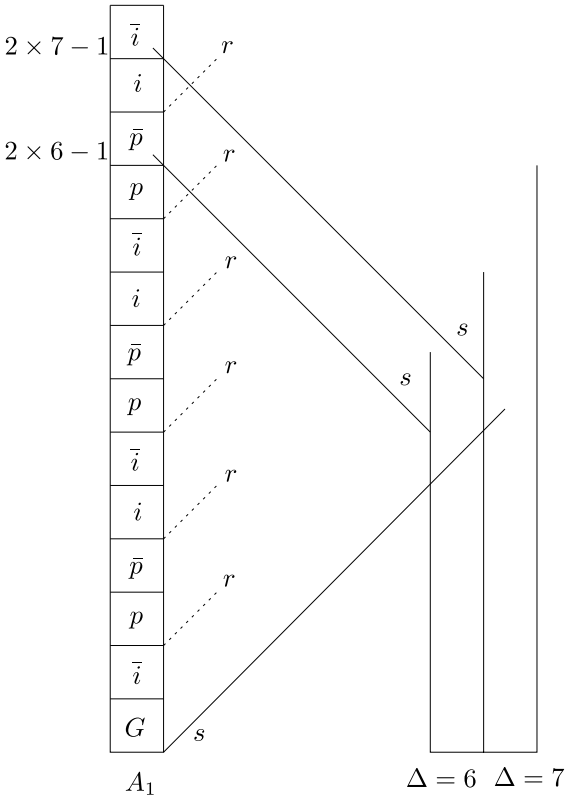


Fig.8

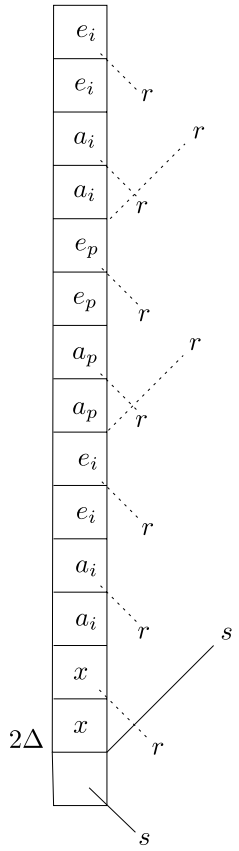


Fig.9

11.2.2 Period 1

No more signals are emitted now. We introduce states

- x, a_p and a_i which absorb r signals
- e_p and e_i which reflect them.

Changes of states are determined by the arrival of the r signals reflected by A_2 (Figure 9), and they succeed one another in the following order

$$(x)\ \ a_i\ \ e_i\ \ a_p\ \ e_p\ \ a_i\ \ \ldots .$$

In the present case each state thus remains 2 time-steps on cell A_1.

State x appears only if s has returned on $\bar{i}$ (indicating that $\varepsilon_0 = 1$), and then it suppresses the first r signal. One out of two of the $\Delta = 2^p + \ldots + \varepsilon_1.2$

signals arriving next are absorbed by the a states, so that the number of signals starting off again (s always included) is

$$\frac{\Delta - \varepsilon_0}{2} = \varepsilon_1 + 2\varepsilon_2 + \ldots + 2^{p-1}.$$

The r signals are sent back every 4 time-steps, the first one starting off at time

$$2\varepsilon_0 + 4 = 2(\varepsilon_0 + 2)$$

of the period.

The number of signals reflected (s included) equals the number of $[(x)ae]$ sequences. If this number is even, at the end of the period s falls on state e_p, if it is odd, s falls on state e_i. In the first case $\varepsilon_1 = 0$, and we decide that s falling on e_p sets state a_i, in the second case $\varepsilon_1 = 1$, we decide that s falling on e_i sets state x, which will suppress the first r signal arriving in period 2 (see Figure 14).

11.2.3 Following periods

During period 2, $\varepsilon_1 + 2\varepsilon_2 + \ldots + 2^{p-1}$ signals arrive and $\varepsilon_2 + \ldots + 2^{p-2}$ start off again, at intervals of 2^3 time units. The first r signal sets off at time $2(\varepsilon_0 + 2\varepsilon_1 + 2^2)$.

It is easy to check by induction on i that during period i, $\varepsilon_i + \ldots + 2^{p-i}$ signals (s included) start off again, at intervals of 2^{i+1} time units, the first r signal setting off at time

$$2(\varepsilon_0 + 2\varepsilon_1 + \ldots + 2^{i-1}\varepsilon_{i-1} + 2^i),$$

and ε_i is determined, its value being 0 or 1 according to wether s falls on e_p or e_i.

During period $p-1$ (Figure 10) $\varepsilon_{p-1}+2$ signals start off again (which amounts to 2 or 3 $[(x)ae]$ sequences), the first r signal setting off at time

$$2(\varepsilon_0 + \ldots + 2^{p-2}\varepsilon_{p-2} + 2^{p-1}),$$

and the value of ε_{p-1} is determined.

During period p, one or two of the r signals are absorbed, the only signal that starts off is s, at time

$$2(\varepsilon_0 + 2\varepsilon_1 + \ldots + 2^{p-1}\varepsilon_{p-1} + 2^p) = 2\Delta.$$

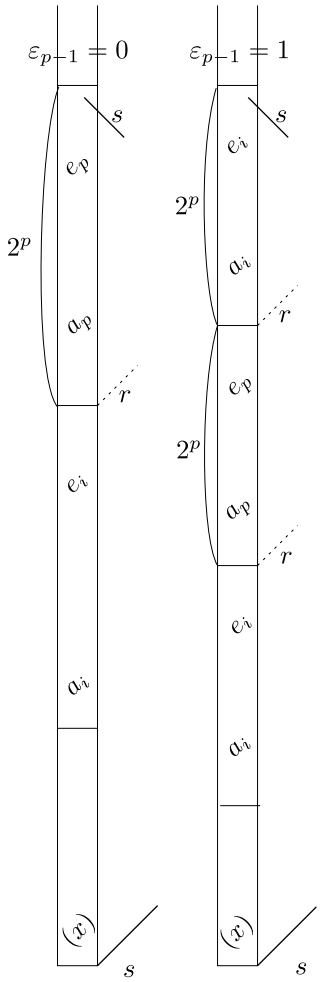


Fig.10 : period $p-1$

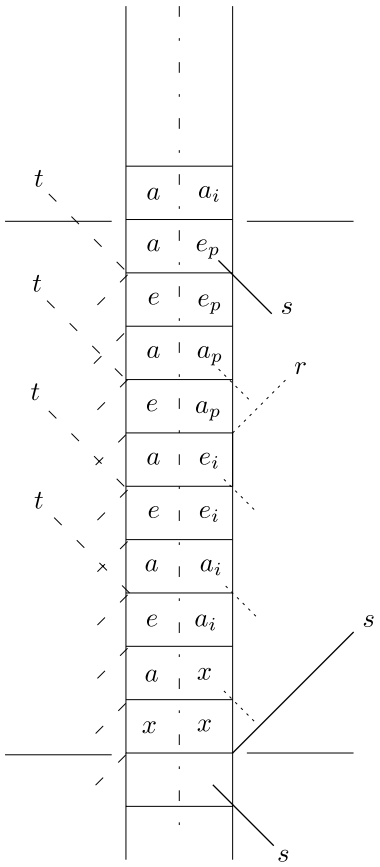


Fig.11

11.3 A first solution

We shall complete the preceding system so as to obtain a synchronization in period p.

11.3.1 Locating period p

To be advised of the entering in period p, end of period $p-1$ must have been spotted. Having this in aim, we shall distinguish the first a, e states from the

next ones. Thus states succeeding to one another will be

$$(x)\ a_i^1\ e_i^1\ a_p^1\ e_p^1\ a_i^2\ e_i^2\ a_p\ e_p\ a_i\ e_i\ a_p\ \ldots$$

If s falls on e_p^1 or e_i^2 (end of second and third a, e sequence), we shall thereby know that we are at the end of period $p-1$.

This will determine A_1 to enter state *syn*, which sends signal *ssyn*, which sets A_2 in turn in state *syn* (see Figures 14, 15) and is reflected.

11.3.2 The shifted signal

Besides, at the same time as we emit the r signals, that is in period 0, we shall emit a second signal system which will work in a similar way. For the sake of clarity, in figures we shall represent these signals going leftwards.

So, during period 0, all states (not G) emit a t signal (see Figure 14).

From period 1 up, states on A_1 will be double (Figure 11), they will be couples, first (or left) component managing the t signals, second (or right) component managing the r signals. Let us point out that the a and e states of left component, whose only task is spacing the t signals, will never have indexes as those of the right component, which compute the ε_i and spot the p-th period.

Clearly, the first t signal starts, in each period, at the time half of the starting time of the first r signal. Thus in period p this signal will start at time Δ.

By this mean we have succeeded, by progressive shiftings, to obtain a signal synchronized with the reflection of *ssyn*.

11.3.3 Fire

During period p (see Figures 14 and 15) we shall need states $(x)\ a\ e$ of left component to determine the first t signal to start off (at time Δ). On the contrary, on the rightside a single *syn* state, indicating that we are in period p, is enough. This state returns signal *ssyn* instead of s, *ssyn* determines state *syn* on A_2 and is reflected. Fire state is set by

- *ssyn* falling on *syn* (on A_1)
- t falling on *syn* (on A_2).

So we have synchronization, at time

$$2\Delta(p+1) = 2\Delta(\lfloor \log \Delta \rfloor + 1).$$

11.3.4 Short lines

For $\Delta = 2 = 2^1 + 0$ and $\Delta = 3 = 2^1 + 1$, $p = 1$, we want to synchronize at the end of period 1. For this, state *syn* must be set by s falling on the first $\bar{p}$ or the second $\bar{i}$, that we must thus distinguish. So on the whole we have states (Figure 12)

$$G\ \bar{i}_1\ p_1\ \bar{p}_1\ i_2\ \bar{i}_2\ p\ \bar{p}\ i\ \bar{i}\ p \ldots$$

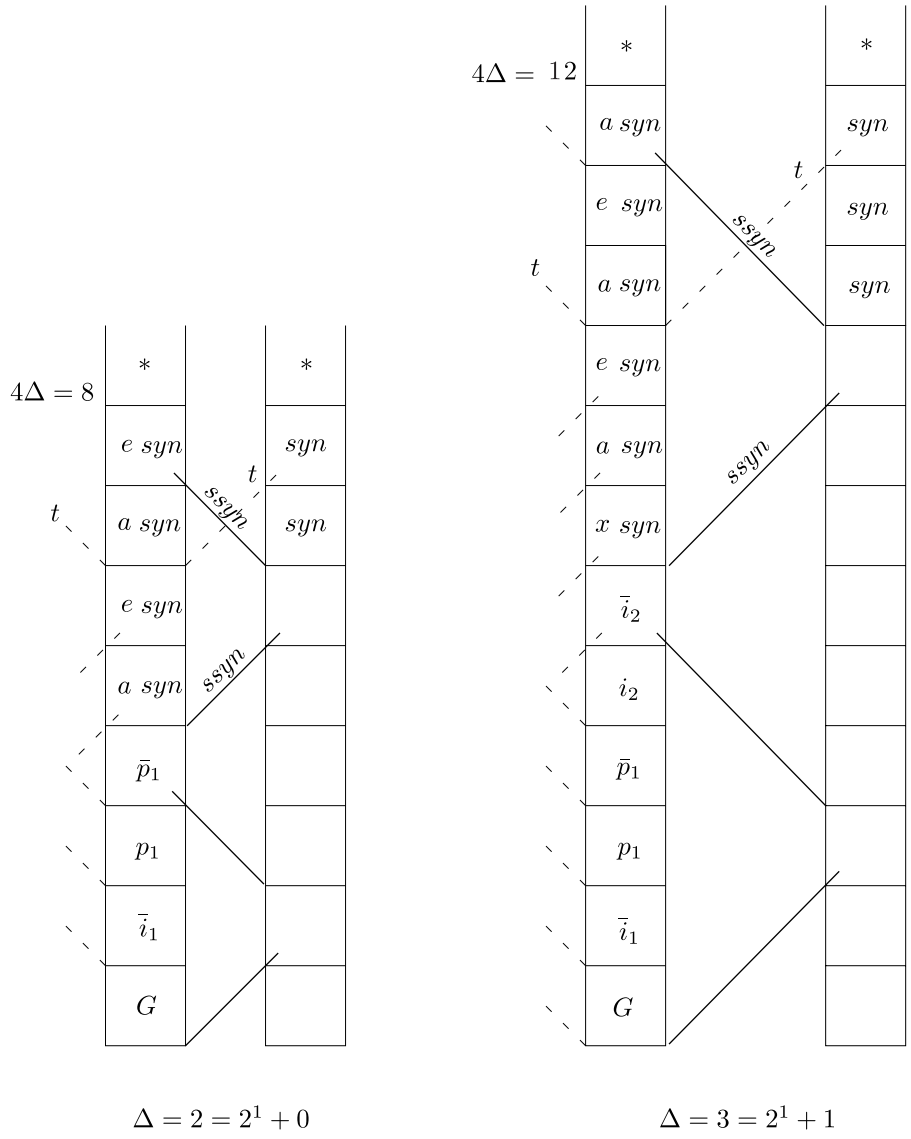
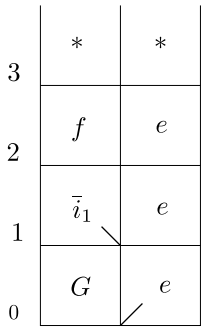


Fig.12

For $\Delta = 1$, $p = 0$ and $2\Delta(p+1) = 2\Delta = 2$. We have seen that synchronization at time 2 cannot be thought of. But we can synchronize at time 3 if we add the particular rules indicated in Figure 13 : s falling on $\bar{i}_1$ sets state f, f gives fire on A_1 and fire sets on A_2 when f is perceived.



$\Delta = 1$

Fig.13

Finally synchronizing time is

$$T_2[\Delta] = \begin{cases} 2\Delta(\lfloor \log \Delta \rfloor + 1) & if \quad \Delta \geq 2 \\ 3\Delta = 3 & if \quad \Delta = 1 \end{cases}$$

11.3.5 Counting the states

To achieve this tedious task, it is essential we should clearly identify the objects to be counted. Particularly, as we have made use of signals, we must emphasize that a signal sent by some state is indeed part of this state. Let us give an example :

- state (e, a_i) not sending any signal
- and the same state (e, a_i) sending signal r

are not the same state. Better have a clear notation : we shall always have 1 or 2 states, accompanied by 1 or 2 signals, so we shall write these signals on the right if they are s or r, and on the left if they are t, as we did in the figures. Here again let us give a few examples

$$(., G, s) \quad (t, p_1, r) \quad (t, \bar{i}, .)$$

$$(t, a, a_p, r) \quad (., e, e_i, .) \quad (., x, x, s) \quad (., a, syn, ssyn)$$

and on A_2

$$(., e, .) \quad (t, e, .).$$

For the while we just give the number of states we have counted, which is 55. Because we shall detail the counting in a more general case in 11.4.4.

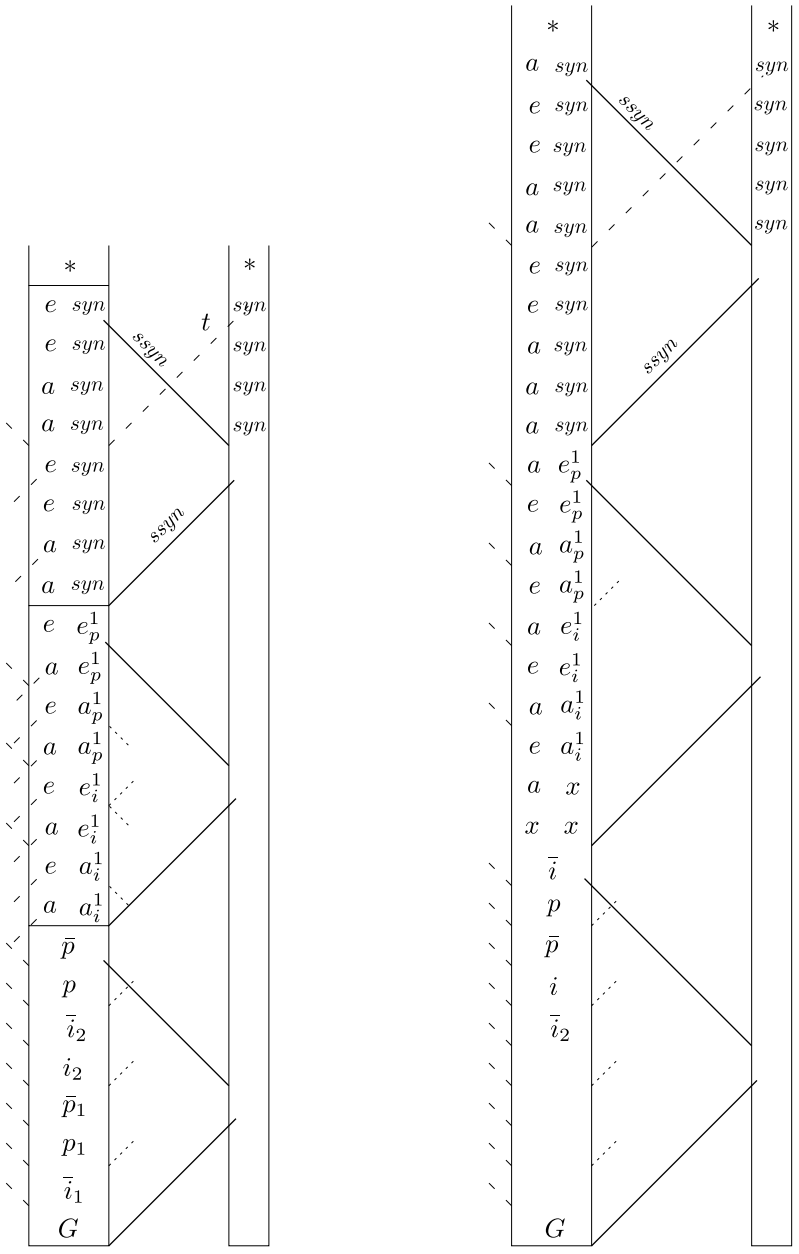
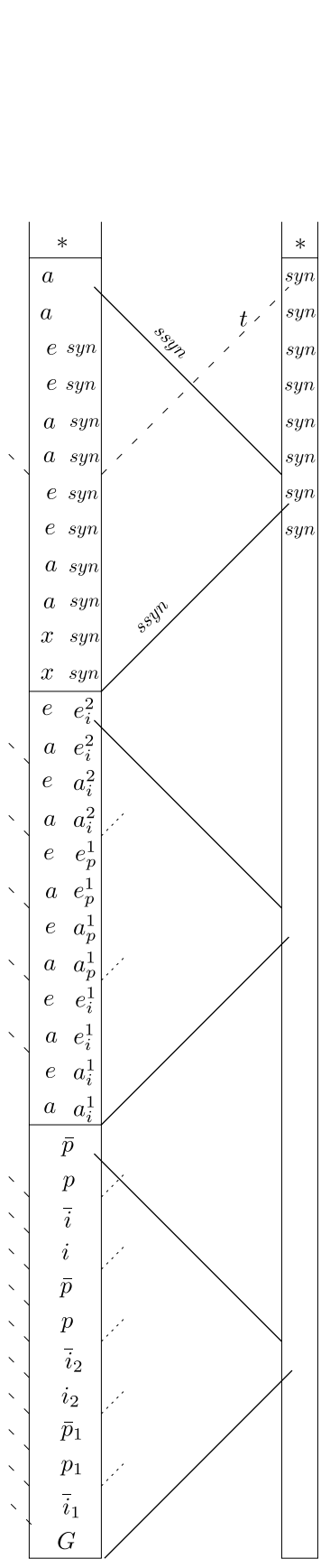
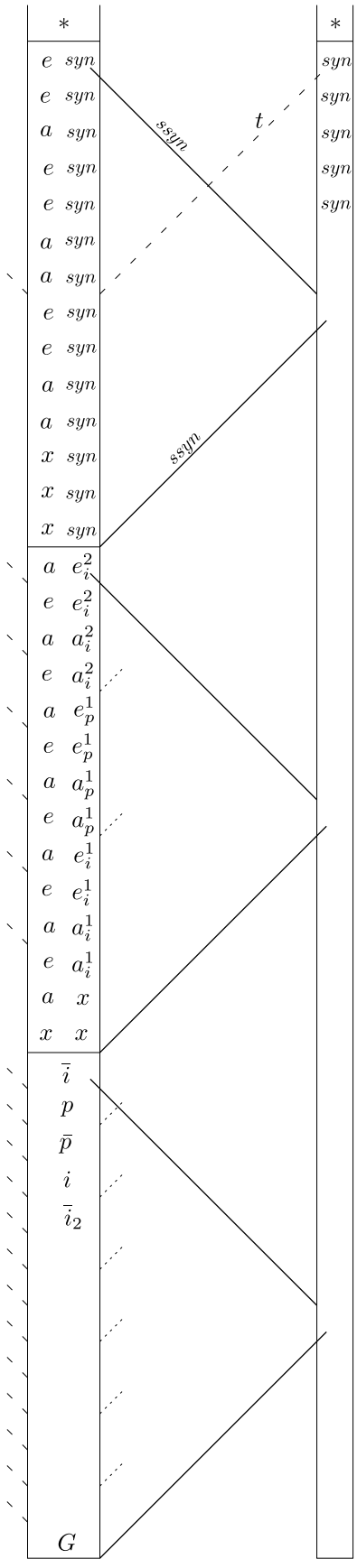


Fig.14 : $(m = 2)$ all cases $p = 2$



$\Delta = 6 = 2^2 + 2^1 + 0$



$\Delta = 7 = 2^2 + 2^1 + 1$

Fig.14 : $(m = 2)$ all cases $p = 2$

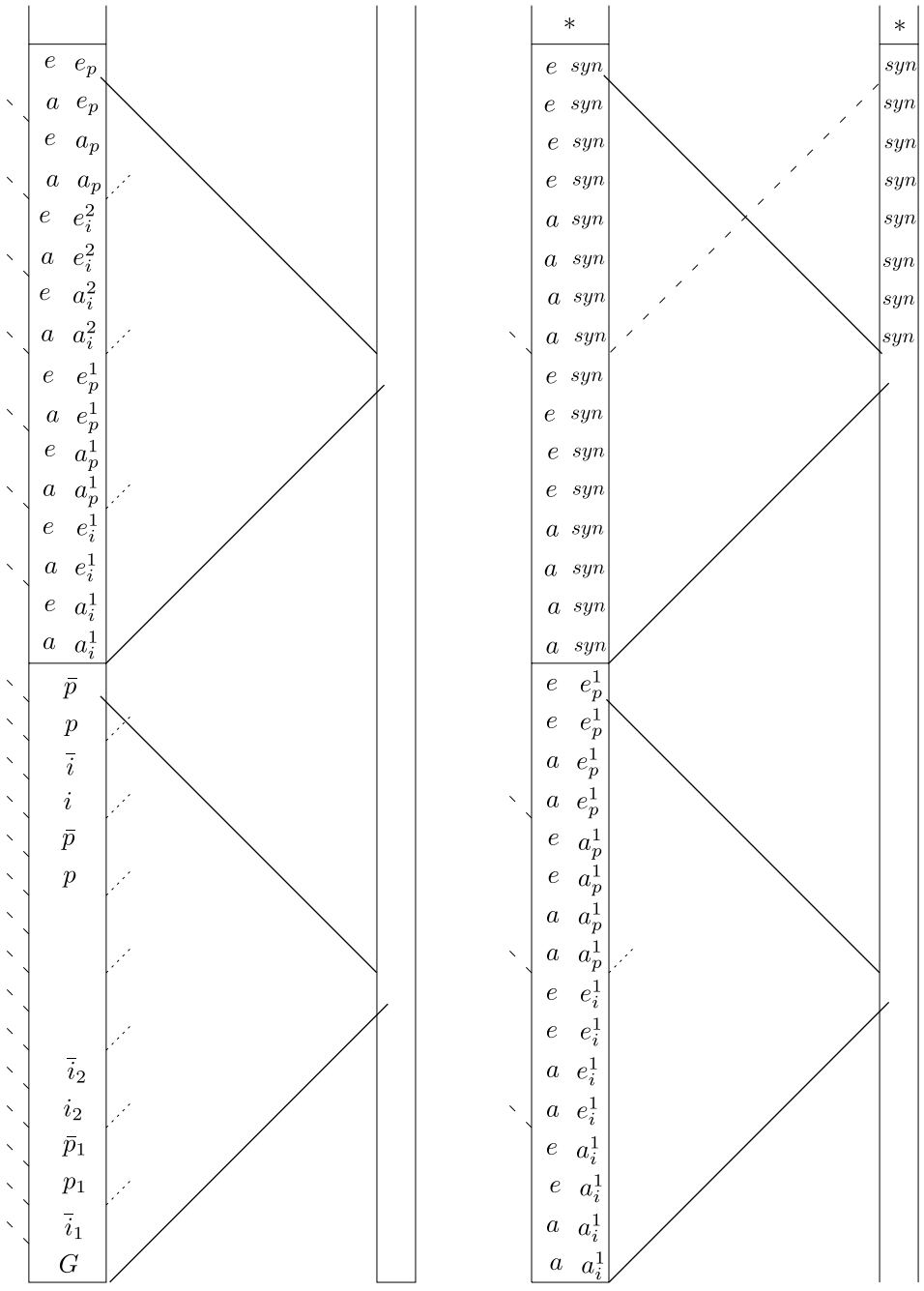


Fig.15 : $(m = 2) \quad p = 3$

11.4 A family of solutions

Solution in basis 2, which we have precisely explained, will help us understand the general solution, with basis m, necessarily more cumbersome but which works with exactly the same principle. The only little difference will appear at the end. Let then m be some integer, $m \geq 2$, and

$$\Delta = \varepsilon_p m^p + \varepsilon_{p-1} m^{p-1} + \ldots + \varepsilon_1 m^1 + \varepsilon_0 \qquad p \geq 0\ ,\ 0 \leq \varepsilon_i < m\ ,\ \varepsilon_p \neq 0$$

11.4.1 In period 0

r and t signals are emitted. States that succeed to one another are

$$(.,G,s) \quad (t,\bar{M}_1,.) \quad (t,M_2,r) \quad (t,\bar{M}_2,.) \quad \ldots$$

$$(t,M_i,r)\ \ (t,\bar{M}_i,.)\ldots(t,M_{m-1},r)\ \ (t,\bar{M}_{m-1},.) \quad (t,M_0,r)\ \ (t,\bar{M}_0,.)$$

$$(t,M_1,r) \quad (t,\bar{M}_1,.) \qquad \ldots\ldots\ldots$$

Δ s and r signals are emitted (Figure 16).

If s returns on $\bar{M}_j$, then $\varepsilon_0 = j$: indeed, as the length of a sequence $M_1\bar{M}_1 \ldots M_0\bar{M}_0$ is $2m$, we have

$$2\Delta = 2m \times \ \textit{number of complete sequences}\ + 2j$$

so

$$2\varepsilon_0 = 2j.$$

This determines as next state state $(.,x_j,x_j,s)$

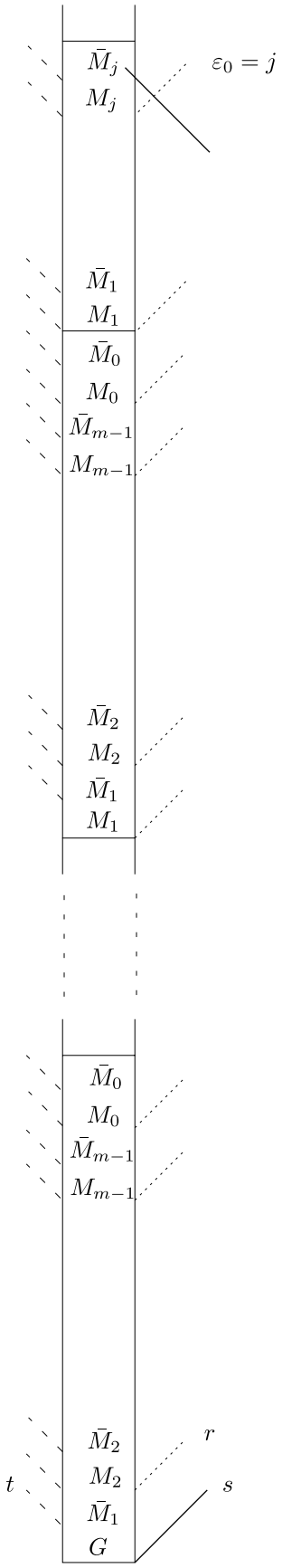

$\bar{M}_j$
M_j
$\varepsilon_0 = j$
$\bar{M}_1$
M_1
$\bar{M}_0$
M_0
$\bar{M}_{m-1}$
M_{m-1}
$\bar{M}_2$
M_2
$\bar{M}_1$
M_1
$\bar{M}_0$
M_0
$\bar{M}_{m-1}$
M_{m-1}
$\bar{M}_2$
M_2
$\bar{M}_1$
G
r
s
t

Fig.16

11.4.2 In period 1 and following periods

Figure 17, where $m = 3$, can serve as illustration.

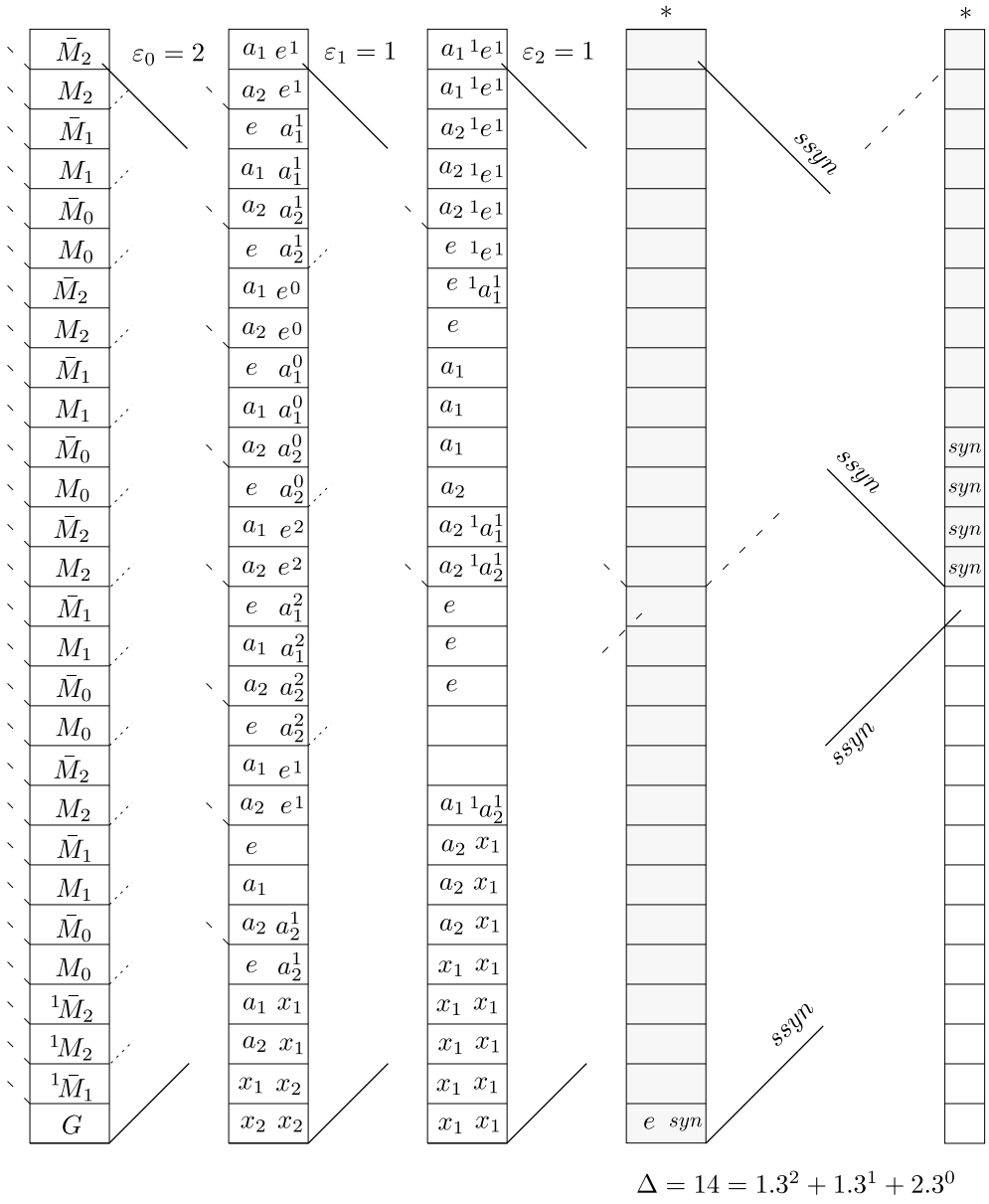


Fig.17

Suppression of signals

It is possible that 1 to $m-1$ of the r and t signals have to be suppressed, for what we shall need $m-1$ suppression states

$$x_{m-1} \quad x_{m-2} \quad \dots \quad x_1,$$

each of them, when receiving some $r(/t)$, which it does not send back, changes into the next one. As for x_1, it changes into a_{m-1}.

If, at the beginning of any period we have state $(., x_j, x_j, s)$, then j of the r and t signals will be suppressed.

Spacing

To make m times less frequent the r and t signals after suppression of some of them by the x states, we shall need m states

$$a_{m-1} \quad \dots \quad a_1 \quad e \qquad a_{m-1} \quad \dots$$

each a_i absorbing the $r(/t)$ signal which changes it into next state, e reflecting it.

In period 0, r signals are separated by 2 time-units, in period k they will be separated by $2m^k$, length of the $[ae]$ sequences of this period.

Determining the ε_k's

We shall have to count, modulo m, the $[(x)ae]$ sequences in period k : for this we must number them

$$[a^1_{m-1} \dots a^1_1 e^1] \qquad [a^2 \;\; e^2] \qquad \dots \qquad [a^{m-1} \;\; e^{m-1}] \qquad [a^0 \;\; e^0].$$

The number of these sequences in period k will be

$$\left\lfloor \frac{2\Delta}{2m^k} \right\rfloor = \varepsilon_p m^{p-k} + \dots + \varepsilon_{k+1} m + \varepsilon_k$$

modulo m this number is ε_k. So, if s returns on e^j we have $\varepsilon_k = j$, and the first state of next period must be

$$(., a_{m-1}, a^1_{m-1}, s) \;\text{ if } \quad j = 0$$

$$(., x_j, x_j, s) \;\text{ if } \quad j \neq 0$$

so that $\varepsilon_k = j$ signals may be suppressed before the m-spacing.

Reflection of the signals

It is easy to check by induction that during period i

- $\varepsilon_i + \dots + \varepsilon_p m^{p-i}$ s and r signals start off and we have as much $[(x)ae]$ sequences
- the first t signal starts off at time $\varepsilon_0 + \varepsilon_1 m + \dots + \varepsilon_{i-1} m^{i-1} + m^i$
- next ones are separated by m^i time-steps.

Period p

Namely, during period p on the right we have ε_p $[(x)ae]$ sequences, $1 \leq \varepsilon_p \leq m-1$, s returns thus on the first series of sequences

$$[a^1\ \ e^1] \qquad [a^2\ \ e^2]\ \ \ldots\ldots\ \ [a^{m-1}e^{m-1}]$$

that we shall distinguish (by index 1 left up) so as to recognize the end of period p

$$[{}^1a^1\ \ {}^1e^1] \qquad [{}^1a^2\ \ {}^1e^2]\ \ \ldots\ \ [{}^1a^{m-1}\ \ {}^1e^{m-1}].$$

First t signal starts off at time

$$\varepsilon_0 + \ldots + \varepsilon_{p-1}m^{p-1} + m^p$$

and next ones are separated by m^p time-steps.

At the end of this period ε_p is determined, equal to j if s falls on ${}^1e^j$.

Difference with basis 2 construction

It comes from the fact that ε_p must be determined as the other ε_k's, knowing it is not null not being sufficient when basis is greater than 2. So one more period will be necessary, and synchronization will occur in period $p+1$.

In order that, in period $p+1$ the first t signal should start off at time

$$\Delta = \varepsilon_0 + \ldots + \varepsilon_p m^p,$$

we must suppress the first $\varepsilon_p - 1$ t signals. For this we establish that s falling on ${}^1e^j$ (of right component) sets state

$$(., x_{j-1}, syn, ssyn) \quad \text{or, if } j=0 \quad (., e, syn, ssyn)$$

and that

$$(., x_1, syn, .) \quad \text{is followed by} \quad (., e, syn, .)\ \ (\text{not}(., a_{m-1}, syn, .))$$

which reflects t. Next signals have no importance whatever, the simplest is to decide that state $(., e, syn, .)$ is permanent.

Synchronizing time is thus

$$T_m(\Delta) = 2\Delta(p+2) = 2\Delta\lfloor \log_m \Delta \rfloor + 4\Delta.$$

11.4.3 Short lines

We think it wiser to examine them separately and carefully.

For $p=1$ every thing works out normally.

Let us examine case $p=0$

$$\Delta = \varepsilon_0 \quad 1 \leq \varepsilon_0 \leq m-1.$$

Is it really possible in this case to obtain synchronization at time indicated by the general formula above, which is

$$2\Delta(p+2) = 4\Delta \ ?$$

We recognize that we are in this case when, in period 0, s returns on the first sequence

$$\bar{M}_1 \ \ \bar{M}_2 \ \ \ldots \ \ \bar{M}_{m-1}$$

that, for the purpose, we distinguish by an index 1, left up

$$^1\bar{M}_1 \ \ ^1\bar{M}_2 \ \ \ldots \ \ ^1\bar{M}_{m-1}.$$

s falling on $^1\bar{M}_j$ (1 indicating that $\Delta = \varepsilon_0$ and j that $\varepsilon_0 = j$, so $\Delta = j$) must then bring state

$$(., a_{j-1}, syn, ssyn),$$

$j-1$ t signals will be absorbed by the successive a states and the jth one, which starts off at time $j = \Delta$, will be reflected, everything works nicely (examples in Figure 18).

Conclusion : for the synchronization problem for pairs with arbitrary delay Δ, we have found a family of solutions $(\mathcal{A}_m)_{m\geq 2}$, whose synchronizing time is

$$T_m(\Delta) = 2\Delta\lfloor \log_m \Delta \rfloor + 4\Delta.$$

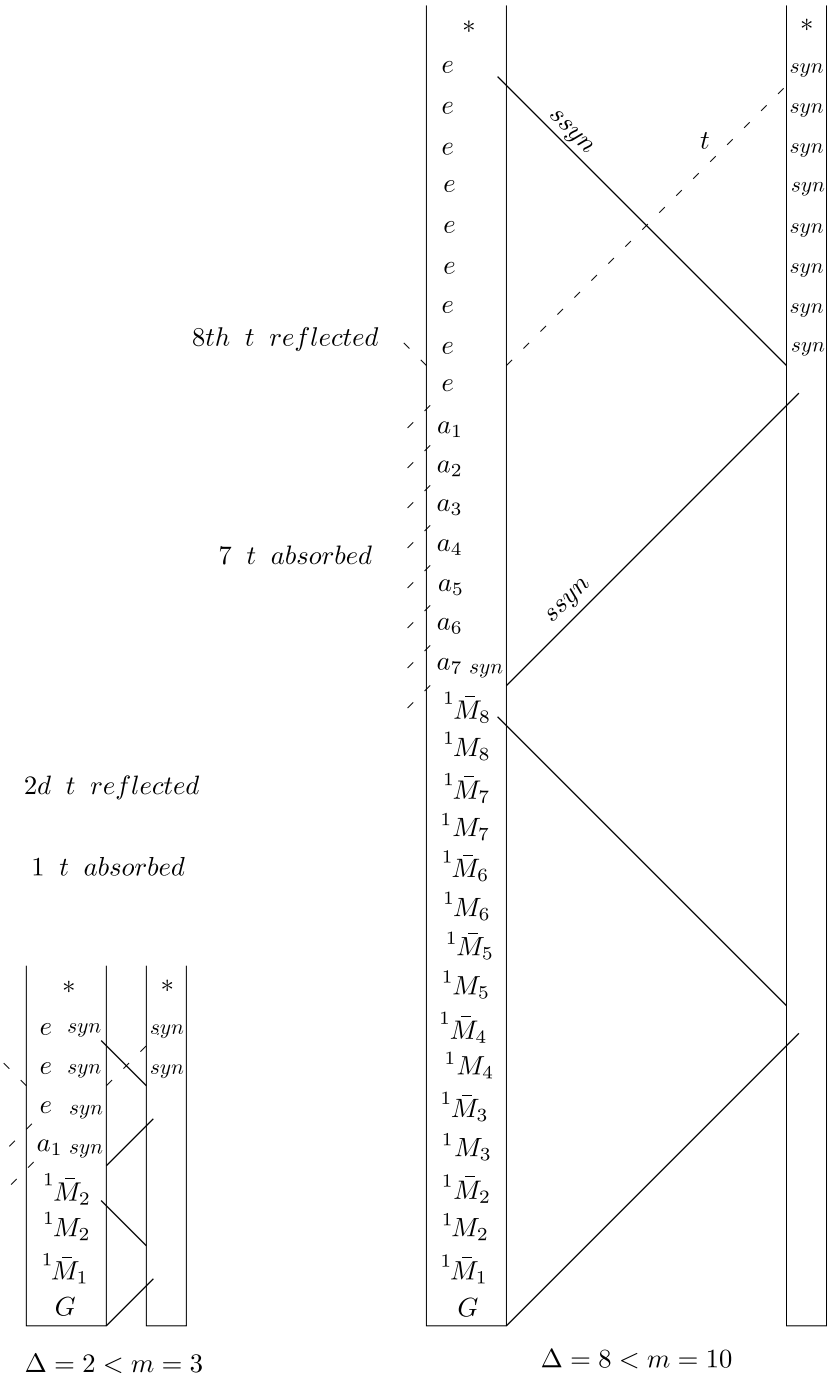

*
e
e
e
e
e
e
e
e
e
a1
a2
a3
a4
a5
a6
a7 syn
1M̄8
1M8
1M̄7
1M7
1M̄6
1M6
1M̄5
1M5
1M̄4
1M4
1M̄3
1M3
1M̄2
1M2
1M̄1
G
*
syn
syn
syn
syn
syn
syn
syn
syn
syn
ssyn
t
8th t reflected
7 t absorbed
ssyn
2d t reflected
1 t absorbed
Δ = 8 < m = 10
Δ = 2 < m = 3

Fig.18

11.4.4 Counting the states

In period 0

on A_1 we may find

$$\begin{array}{cccccc} & & (.,G,s) & (t,{}^1\bar{M}_1,.) & \dots & (t,{}^1M_{m-1},r)\ (t,{}^1\bar{M}_{m-1},.) \\ (t,M_0,.)\ (t,\bar{M}_0,.) & (t,\bar{M}_1,r) & (t,{}^1\bar{M}_1,.) & \dots & (t,{}^1M_{m-1},r)\ (t,{}^1\bar{M}_{m-1},.) \end{array}$$

which makes $4m-2$ states

In period 1 and following

we find couples having

- as possible left components

$$x_{m-1}\ \dots\ x_1\ a_{m-1}\ \dots\ a_1\ e$$

- as possible right components

$$\begin{array}{cccccccccc} {}^1a^1_{m-1} & \dots & {}^1a^1_1 & {}^1e^1 & \dots & {}^1a^{m-1}_{m-1} & \dots & {}^1a^{m-1}_1 & {}^1e^{m-1} & \\ a^1_{m-1} & \dots & a^1_1 & e^1 & \dots & a^{m-1}_{m-1} & \dots & a^{m-1}_1 & e^{m-1} & a^0_{m-1}\dots a^0_1e^0 \end{array}$$

Let us momentarily forget indexes up at the right of the right component to examine how left and right states are associated : next to two $[a_{m-1}\dots a_1e]$ sequences on the left we have on the right a sequence $[a_{m-1}a_{m-1}\dots a_1a_1ee]$ (Figure 19), with a shifting of 0 to $m-1$ time-steps, depending on first state of the period.

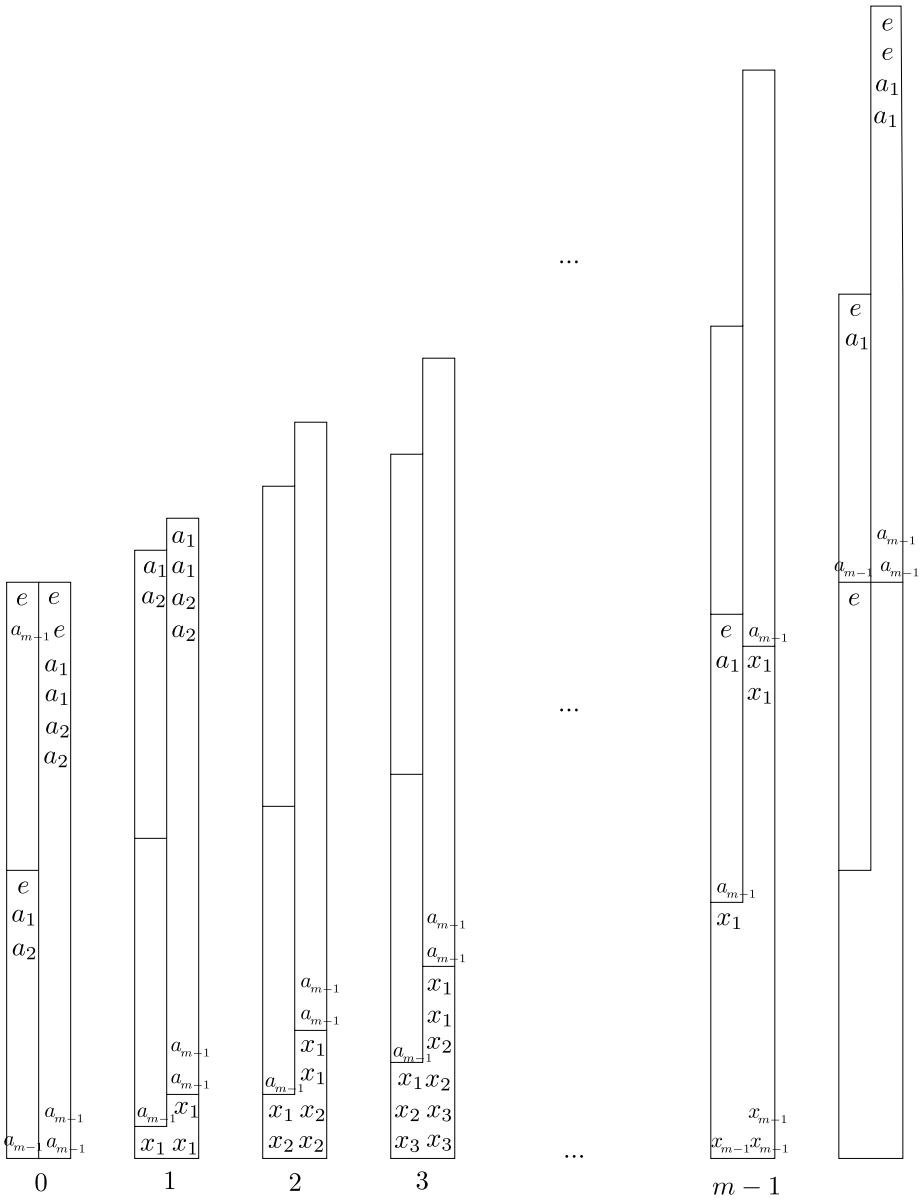


Fig.19

Let us first count couples with an x state : we count

$$0 + 2 + \ldots + 2(m-1) = m(m-1),$$

but if we look more attentively we note that one couple out of two, except the last $2(m-1)$, is obtained twice, so we really have only $(m+2)(m-1)/2$.

Let us now count couples with two a states : in each column of Figure 19, corresponding to shiftings from 0 to $m-1$, we find $2m$ couples, but from shifting 1 up, one couple out of two has already been found in preceding column, so the total number of couples is

$$2m + m(m-1) = m^2 + m.$$

If we reintroduce the up indexes of the right states, the number of couples is multiplied by $2m-1$, so the number becomes

$$(2m-1)(m^2+m) = 2m^3 + m^2 - m$$

and for the central periods we have a total of

$$2m^3 + 1.5m^2 - 0.5m - 2.$$

At last we must mention that on the right, state ${}^1a^1_{m-1}$ will be sometimes associated with a starting off of signal s, sometimes not, and the other states ${}^1a^i_{m-1}$ and all states a^i_{m-1} may be or not associated with a reflection of signal r, which doubles the number of a_{m-1} states on the right, and finally adds $2m^2 + m + 1$ to our count.

In period $p+1$

- on A_1 we find states

$$(.,x_{m-1},syn,ssyn)\ldots(.,x_1,syn,ssyn)\ \ (.,e,syn,ssyn)\ \ (t,e,syn,.)$$

- on A_2

$$e\ \ (.,e,s)\ \ (.,e,r)\ \ (t,e,r)\ \ (.,syn,ssyn)\ \ syn$$

- on A_1 and A_2 fire state *

a total of $m+8$ states. The total number of states of $\mathcal{A}_m$, from period 0 to period $p+1$, is thus

$$2m^3 + 3.5m^2 + 5.5m + 5.$$

For $m = 2$ this amounts to 46, a little less than the number of states of the first solution, whose synchronizing time was also a little less.

11.4.5 Reducing the synchronization time

We shall now proceed to a trifling modification of the $\mathcal{A}_m$'s into the $\mathcal{B}_m$'s whose synchronizing time will be

$$T_m(\Delta) = \begin{cases} 2\Delta\lfloor\frac{\log\Delta}{\log m}\rfloor + 4\Delta & \text{if } \Delta \geq m \\ 3\Delta & \text{if } \Delta < m \end{cases}$$

We just add $m-1$ states

$$f_{m-1} \quad \cdots \quad f_2 \quad f_1$$

such that each of them gives the next one and f_1 gives the fire, and decide that s falling on ${}^1\bar{M}_j$ (thus indicating that $p=0$) gives f_j and A_2 enters state fire as soon as it perceives state f_j (Figure 20).

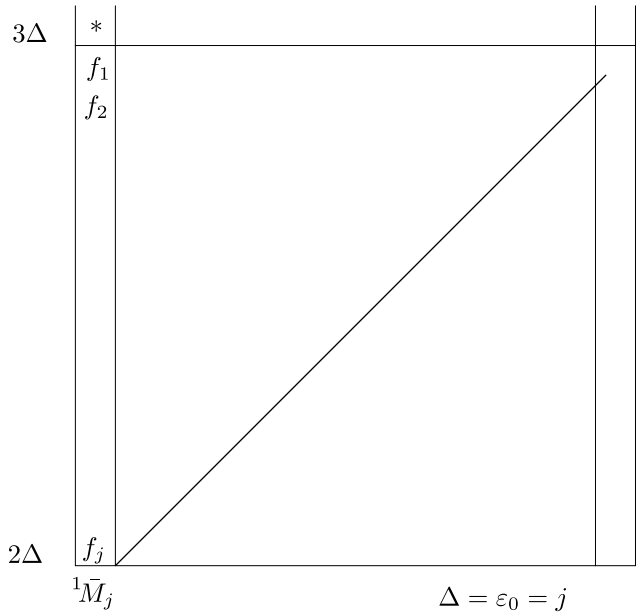


Fig.20

The family of solutions $(\mathcal{B}_m)_{m\geq 2}$ realizes, for the small delays, the minimal possible time. Because of this property, this family has a theoretical interest which we shall use later on.

11.5 Lower bounds for synchronization time

We now prove that synchronization times obtained in the preceding sections are optimal in order of magnitude.

Let $\mathcal{A}$ be a synchronizing solution, with N states. From now on, we shall consider very large delays, much larger than N and even than powers of N.

We shall number the periods in a different manner, more suitable for our new purpose :

$$(k-1)\Delta \leq \text{period k} < (k+1)\Delta.$$

Now the odd periods are determined by the clock signal's return on A_1, the even periods by its return on A_2 (Figure 21).

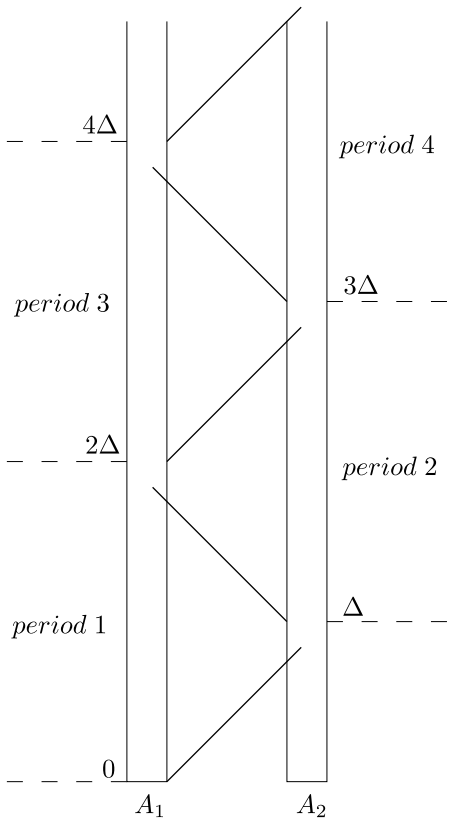


Fig.21

11.5.1 Periodicity of states and first result

Period 1

A_2 being quiescent up to time $\Delta - 1$ included, during period 1, A_1 progresses without being influenced by A_2. Among the $N + 1$ first states of this period, 2 must be the same, so there exists

$$0 \leq i_1 < i_1 + h_1 \leq N$$

such that

$$< A_1, i_1 >=< A_1, i_1 + h_1 > .$$

From time i_1 to time $2\Delta - 1$, A_1 thus behaves periodically with period h_1. Let us observe that $i_1 \leq N - 1$ and $h_1 \leq N$.

So, from some instant of time not greater than $N - 1$ of period 1, evolution of A_1 is periodical with period not greater than N.

Period 2

From time $\Delta - 1 + i_1$, that we shall call time $i_1 - 1$ of period 2, A_2 repeatedly receives the same series of h_1 states. Let us consider, on A_2, the instants where the first state of these series is perceived. Among the $N + 1$ states of the first $N + 1$ of these instants, two must be the same (Figure 22).

So there exists

$$0 \leq i_2 < i_2 + h_2 \leq N$$

such that

$$< A_2, \Delta - 1 + i_1 + i_2 h_1 > = < A_2, \Delta - 1 + i_1 + (i_2 + h_2) h_1 > .$$

from what follows that from time

$$\Delta - 1 + i_1 + i_2 h_1$$

so a fortiori from time

$$\Delta + i_1 + i_2 h_1$$

and up to time $3\Delta - 1$, A_2 behaves periodically with period $h_2 h_1$. Let us observe that

$$i_1 + i_2 h_1 \leq N - 1 + (N - 1)N = N^2 - 1 \qquad \text{and} \qquad h_1 h_2 \leq N^2$$

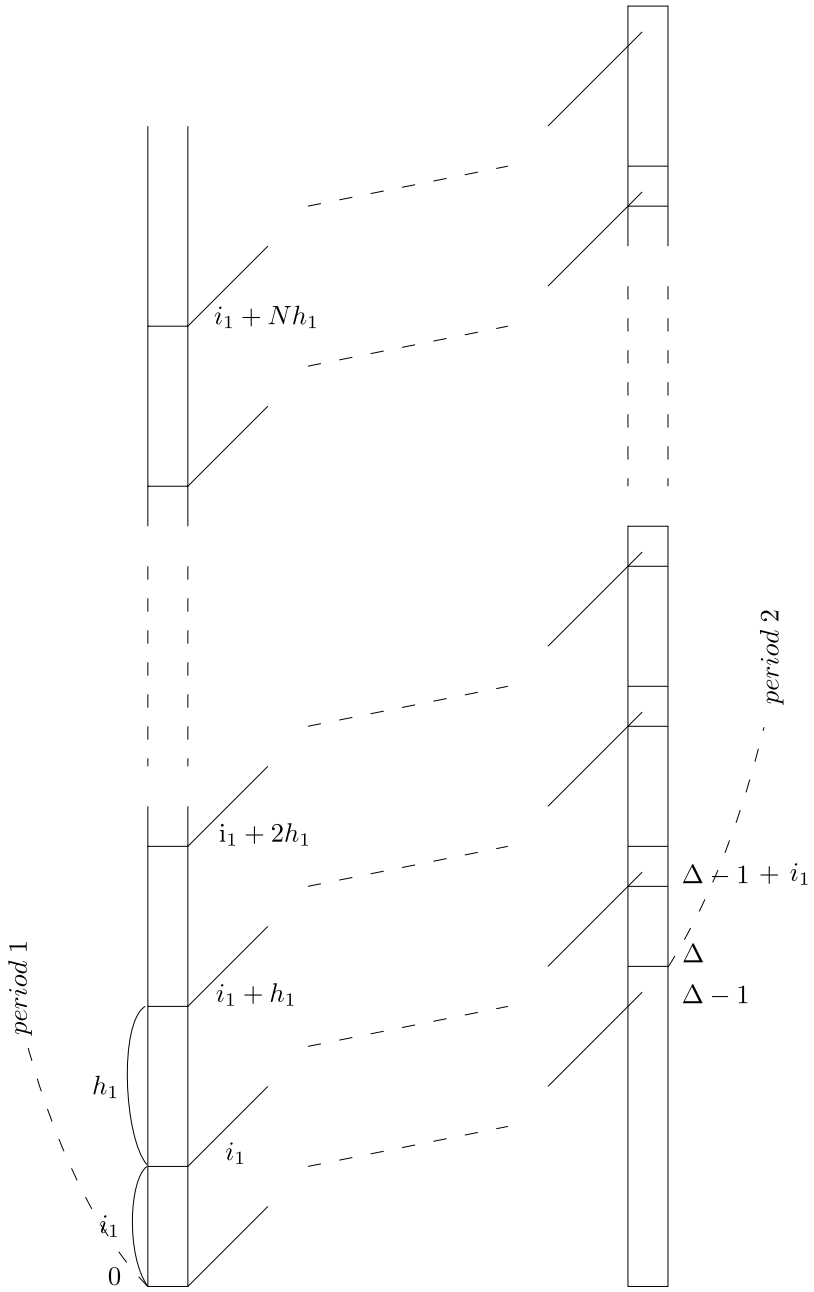

$i_1 + Nh_1$
$i_1 + 2h_1$
$i_1 + h_1$
i_1
i_1
0
h_1
period 1
period 2
$\Delta - 1 + i_1$
Δ
$\Delta - 1$

Fig.22

Following periods

By induction on k it is easy to check that, during period k,

- from some time $\leq N^k - 1$

- the automaton concerned behaves periodically with period $\leq N^k$

as a result : any state appearing during period k appears there before time

$$(k-1)\Delta + 2N^k - 1.$$

11.5.2 A first proposition and a corollary

Proposition 11.5.1

$$\forall \Delta \geq 2N^{k-1}, \quad T(\Delta) \geq k\Delta$$

Proof is by induction. For $k = 1$, 2 and 3, this result is already known, and for all Δ.

Consider $\Delta \geq 2N^k$: if fire should appear before time $(k+1)\Delta$, end of period k, from preceding result, it should have appeared before time

$$(k-1)\Delta + 2N^k - 1 < (k-1)\Delta + \Delta = k\Delta.$$

If, by induction hypothesis, for $\Delta \geq 2N^{k-1}$ fire cannot appear before time $k\Delta$, we have a contradiction !

Figure 23 illustrates this proposition, and makes next corollary intuitive.

Corollary 11.5.2 *no synchronizing solution can synchronize in time a linear function of* Δ

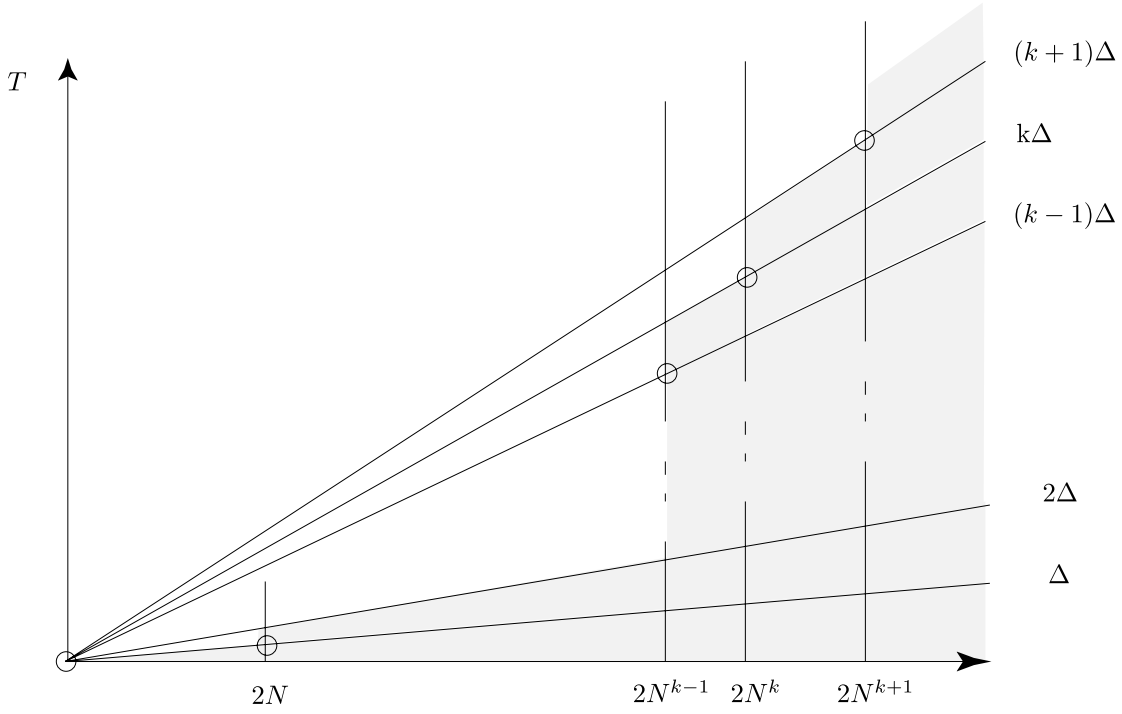


Fig.23

But we can get more ...

11.5.3 A second proposition and a consequence

Proposition 11.5.3

$$\forall \Delta \quad T(\Delta) > \frac{1}{\log N}\Delta(\log \Delta - 1)$$

Proof : for all Δ (≥ 2), there exists some integer k (≥ 1) such that $2N^{k-1} \leq \Delta < 2N^k$. This k satisfying

$$N^k > \frac{\Delta}{2}$$

$$k > \frac{\log(\Delta/2)}{\log N} = \frac{\log \Delta - 1}{\log N}.$$

From proposition 11.5.1 results that

$$T(\Delta) \geq k\Delta > \Delta\frac{\log \Delta - 1}{\log N}$$

We shall now compare the synchronization time of solutions $\mathcal{A}_m$ with this lower bound. For the $\mathcal{A}_m$'s :

$$T_m(\Delta) = 2\Delta(\lfloor \log_m \Delta \rfloor + 2) \leq 2\Delta(\frac{\log \Delta}{\log m} + 2).$$

The $\mathcal{A}_m$'s have number of states $N = 2m^3 + 3,5m^2 + 5,5m + 5 \approx 2m^3$.

For such a number of states, the preceding lower bound is

$$T_{min}(\Delta) = \frac{1}{3logm + 1}\Delta(\log \Delta - 1)$$

which gives

$$\frac{T_m(\Delta)}{T_{min}(\Delta)} = 6\left[1 + \frac{1}{3\log m}\right]\frac{1 + \frac{2\log m}{\log \Delta}}{1 - \frac{1}{\log \Delta}} \approx 6$$

if m is large and Δ very large.

This remarkable result leads us to think that the solutions of Mazoyer are quite good.

Let us think about criterions for appreciating solutions.

11.5.4 Reflection on the notion of optimality

A synchronizing solution

for a class of c.a's consists of some automaton A and its transition function which, starting from an initial configuration where one automaton is in state general and the others are quiescent, leads the whole c.a to the fire state at some determined instant of time whose value depends on the particular c.a in the class.

Example : for the class of finite lines without delay we have several solutions, Minsky's solution (taking time about $3n$, n length of the line), those of Goto, Waksman, Balzer, Mazoyer (in time $2n - 2$).

For the class of pairs with delay we have an infinite number of solutions,the $\mathcal{A}_m$ and the $\mathcal{B}_m$.

Which are the best solutions ?

For the finite lines, Goto's solution is better than Minsky's in time, but worse as regards the number of states (it has thousands), Mazoyer's solution is better in both respects.

For the pairs with delay, if $m' > m$, $\mathcal{A}_{m'}$ is better in time than $\mathcal{A}_m$, but it has more states.

When shall we declare that a solution is optimal ?

The first idea, the simplest one too, is to consider only the time and to demand as much as possible : that the solution realizes the best possible time for each network of the family considered.

In this acceptation, solutions of Goto and the other minimal time ones are all optimal. For pairs, the best possible time for each pair is 3Δ, because less is impossible and this time is effectively realized by the $\mathcal{B}_m$'s for $m > \Delta$. As there is no solution in time 3Δ, there is no optimal solution.

We see that this first idea, if it has the advantage of simplicity, very much lacks subtlety : it does not place Mazoyer, Balzer and Waksman on the podium, and indistinctly rejects all solutions for pairs with delay. So we go on thinking and striving for a good definition.

The example of pairs seems less particular and miraculous than that of the lines in as much as gaining in time appears to necessitate increasing the means, that is the number of states : we observe this fact for our two families of solutions, and the value of our lower bound, smaller if the number of states is greater, confirms it. And we know that this is the normal and general rule.

It seems thus more realistic to compare solutions realizing the same time, or solutions having the same number of states.

For lines, among all minimum time solutions, Mazoyer (6 states) is then recognized as the best.

For pairs with delay, if we evaluate Mazoyer's $\mathcal{A}_m$ solutions with this point of view, they appear to be excellent, as, with the same number of states, their time is of the same order, not as the minimal time which we do not know, but as a lower bound, very loosely calculated. We may certainly declare them quasi-optimal, and risk the conjecture that they are the fastest possible ones for their number of states.

Chapter 12

Synchronizing a line with non uniform delays

The interest of the solution we hereafter present and which is due to Mazoyer [37], is emphasized if we recall Jiang's results. Jiang [30] describes a solution for the synchronization problem of the class of all networks of finite automata, whatever be the graph of their (symmetrical) connections and the delays for communication through them. The synchronizing time of this solution is of order

$$O(DR^3)$$

where DR, the "delay-radius", is the delay for communication between the "general" automaton and the farthest among the "soldier" automata. The idea of this solution (cf. chapter 13) consists in using a spanning tree with the general as root for the graph of the network, and to replace synchronization of this tree by the synchronization of the subtrees, having generals at their roots. These synchronizations must be properly delayed for a good orchestration of the whole. The automata will compute the necessary delays from the delays between one another by circulating signals. Synchronization of each subtree will be likewise decomposed, till the subtrees are reduced to single nodes.

Mazoyer's solution has very general features in common with Jiang's : it is also a divide-and-conquer process, the line is repeatedly broken in smaller sublines, and the synchronizations of the sublines are conveniently delayed. But the involved techniques are completely different : the method for breaking the line is related to Minsky's solution for a line without delays (see chapter 1), and the computations of delays does not use Jiang's techniques. To be sure, its time is noticeably better, being of order $O(DR^2)$ but for a class which is also incomparably more restricted.

Indeed the family we consider now consists of particularly simple networks, lines with delays. The class is nevertheless quite important, as the number n of automata and the communication delays as well, are completely arbitrary.

In Figure 1, delays $\tau_{i,i+1}$ between A_i and A_{i+1} are represented as in chapter 11, by proportional distances between the automata.

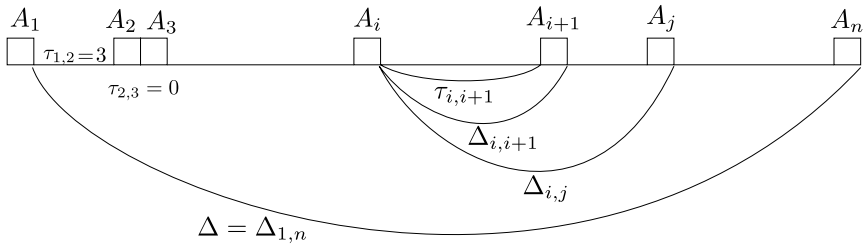


Fig.1

As for the case of a pair in chapter 11, we shall prefer using what we have called the "reaction" delays

$$\Delta_{i,i+1} = \tau_{i,i+1} + 1 \quad \geq 1.$$

We shall denote $\Delta_{i,j}$ the cumulated delay from i to j

$$\Delta_{i,j} = \Delta_{i,i+1} + \Delta_{i+1,i+2} + \ldots + \Delta_{j-1,j}$$

and Δ the cumulated delay from the first to the last automaton

$$\Delta = \Delta_{1,n}$$

which is Jiang's delay-radius in this case. For a couple (line of two) without delay we have $\Delta = 1$ and we decide that $\Delta = 0$ characterizes a line reduced to one single automaton.

For $\Delta > 1$, the synchronizing time of the solution we give next will be

$$T(\Delta) = 2\Delta^2.$$

So as not to be interrupted by technical details when we present the synchronizing process, we shall first show how the automata can do computations about their delays. To this effect, we expose several examples, the results of which will be used in the sequel, to delay sub-synchronizations.

Let us underline once and for all that the left(/right)-end automaton has his left(/right) input unused, so it knows all along that it is the first(/last) automaton of the line.

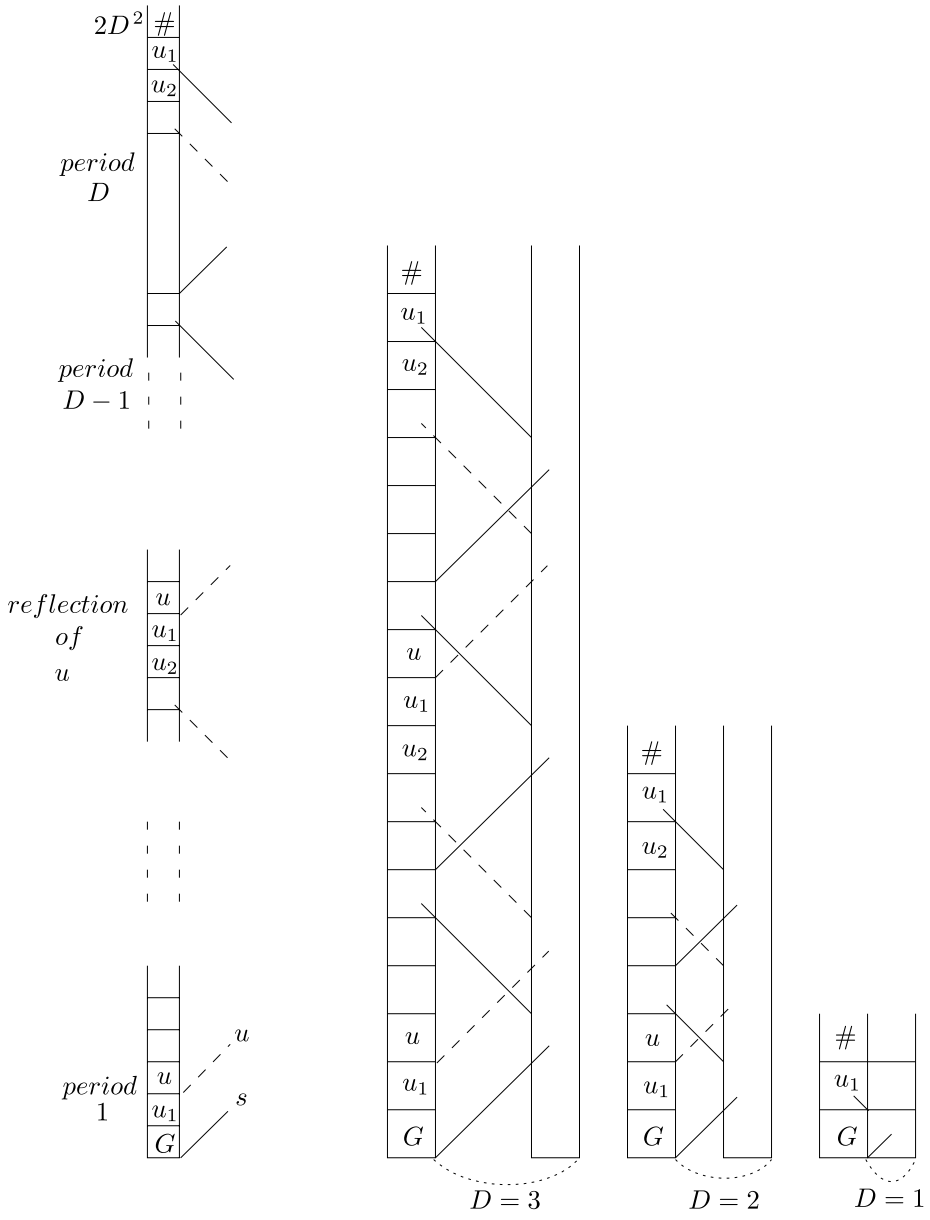

$2D^2$
#
u_1
u_2
period D
period D − 1
reflection of u
u
u_1
u_2
period 1
u
s
G
D = 3
D = 2
D = 1

Fig.2 : $T_1(D) = 2D^2$

12.1 Some times which are computable with a pair of cells

We deal now with time computations that an automaton can perform with the communication delay between itself and a single second automaton, let this (reaction) delay be D. We insist that these times are not synchronization times, as they are obtained only on the automaton which starts the computation. To avoid confusions, the state marking these times will be denoted by a new symbol, $\#$, and not by $*$.

The lower bound we have for these times is here

$$T(D) \geq 2D.$$

Indeed to compute with the delay, a signal must be sent and the response signal must come back, and this takes two times the reaction delay.

As in chapter 11, the basic signal is the clock signal s, which appears on the automaton which starts it at successive times 0 (starting time), $2D$, $4D$, ...

12.1.1 Time $T_1(D) = 2D^2$.

In addition to the clock signal s, we use a signal u, emitted in first period at time 2, reflected normally by the second automaton and with delay 2 on the first one : the states needed, u_2, u_1 and u are represented in Figure 2. Signal u being delayed by 2 time-steps at each period, is delayed by exactly $2D$ at period D, which means that at the end of this period, s returns on state u_1. We decide that this circumstance sets up state $\#$ (Figure 2).

12.1.2 Time $T_2(D) = 2D(D-1)$.

To be more precise, time will be exactly

$$T_2(D) = \begin{cases} 2D(D-1) & \text{if } D > 1 \\ 2D = 2 & \text{if } D = 1 \end{cases}$$

Minor modifications on the preceding example suffice (see Figure 3) :

- u starts at time 4, (so a new state u_3 is needed)

- for the case when $D = 1$: return of s on state u_3 sets up state $\#$.

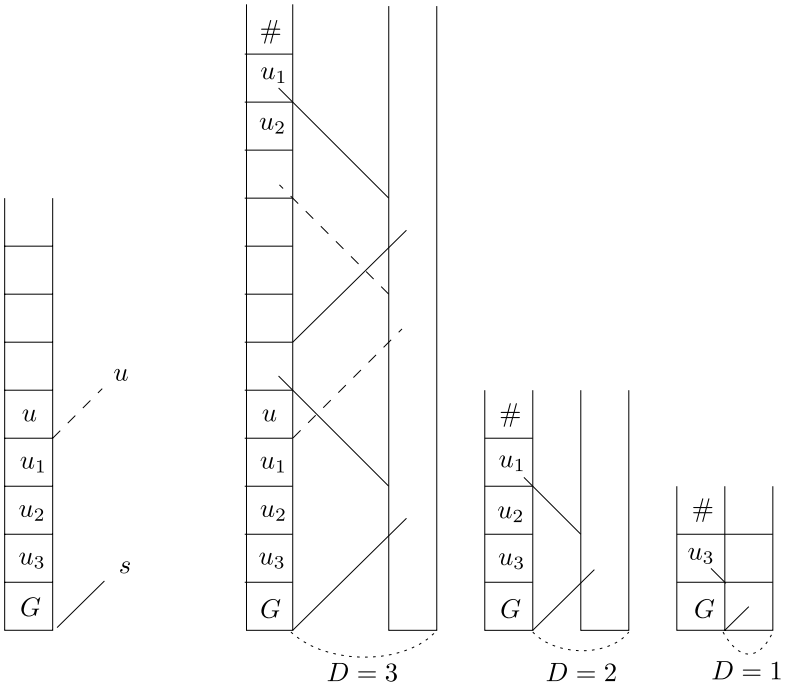


Fig 3 : $T_2(D) = \begin{cases} 2D(D-1) & \text{if } D > 1 \\ 2D = 2 & \text{if } D = 1 \end{cases}$

12.1.3 Time $T_3(D) = 2D(D-2)$.

More precisely

$$T_3(D) = \begin{cases} 2D(D-2) & \text{if } D > 2 \\ 2D & \text{if } D = 1 \text{ or } 2 \end{cases}$$

Now (Figure 4)

- u starts at time 6 (new states u_4 and u_5 are needed)

- for cases when $D = 2$ or $D = 1$: return of s on u_3 or u_5 sets up state #.

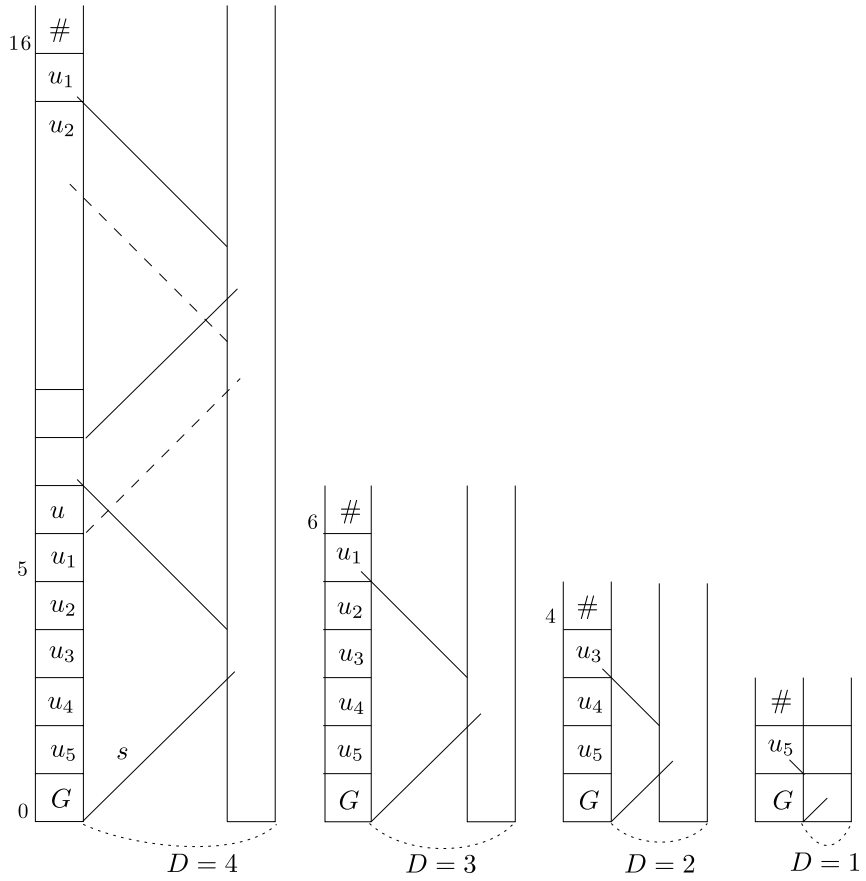


Fig 4 : $T_3(D) = \begin{cases} 2D(D-2) & \text{if } D > 2 \\ 2D & \text{if } D = 1 \text{ ou } 2 \end{cases}$

The times which follow now are obtained by an extra delay, of one period or half a period, on preceding times. No new figures are needed, a few pencil strokes on the old ones are sufficient to visualize things.

To get computation times in 12.1.4 to 12.1.6, we start with the signals used for delay T_1.

12.1.4 Time $T_4(D) = 2D^2 + D$.

We use a supplementary signal denoted v. It starts at time 1 in period 1, and is delayed by one time-step at each reflection on the first automaton. Its cumulated delay in period D is then D. We just decide that state + is obtained when v returns to cell 1 after state # has appeared. Time T_4 is obtained when state + appears.

12.1.5 Time $T_5(D) = 2D^2 + D - 1$.

If we emit signal v at time 0, we obtain time T_5. Except in case $D = 1$: for this case we simply decide that the return of s and v together on state u_1 sets state $+$.

12.1.6 Time $T_6(D) = 2D^2 + 2D - 2$.

Our signal v, still starting at time 0, is now delayed by 2 time-steps at each reflection on the first automaton, exactly as u is. And for case $D = 1$ we likewise decide that the return of s and v together on state u_1 sets state $+$.

To get the next times, we start with the signals for delay T_3

12.1.7 Time $T_7(D) = 2D(D-2) + D - 1 = 2D^2 - 3D - 1$.

More precisely

$$T_7(D) = \begin{cases} 2D(D-2) + D - 1 & \text{if } D > 2 \\ 2D & \text{if } D = 1 \text{ or } 2 \end{cases}$$

Here we use signal v started in period 1 at time 2, reflected on the first automaton with delay 1, this if $D > 2$. For the remaining cases, we just decide that the return of s on state u_3 or u_5 sets state $+$.

12.1.8 Time $T_8(D) = 2D(D-1) - 1$.

Thereagain let us be quite precise

$$T_8(D) = \begin{cases} 2D(D-1) - 1 & \text{if } D > 2 \\ 2D & \text{if } D = 1 \text{ or } 2 \end{cases}$$

This time can be expressed

$$T_8(D) = T_3(D) + 2D - 1$$

We emit signal v at time 5, and this signal is reflected on the first automaton with delay 2.

For the cases $D = 1$ or $D = 2$, we decide that state $+$ is set by the return of s on u_3 or u_5.

12.2 Some times computable with three automata

We deal here with computations done by the central automaton on the delays d and D ($d, D \geq 1$) with the left and right automata. These lines of three work mainly with the two clock signals s and S, which determine the "small" and the "big" periods, as we shall call them, having values $2d$ and $2D$.

We shall continue to speak of signals, so convenient, though we know that they are fictitious and we have only states perceived with delays. For example, when we say that a signal is sent to a particular automaton, this means that only the perception of the state by the concerned automaton is of interest for us, perception by the other automaton having no consequence.

12.2.1 Time $T_9(D, d) = 2D(2d - 2)$

During the first "small" period, signals r are emitted, starting at time 4. They are destroyed at the rythm of 1 per "big" period : each return of S sets up state a, which absorbs one signal r and disappears (Figure 5). In period $(2d - 2)$, no more signal r arrives, S returns on state a, and this sets up state #.

12.2.2 Time $T_{10}(D, d) = 2D^2 + 2D - d$

We rewrite this time as follows :

$$2D^2 + d + 2(D - d).$$

To spot the D-th period $2D$ we use the same u signal as in the first example of the preceding section.

To obtain the shift of $d + 2(D - d)$ time-steps, we use a signal v emitted at time 1, which is delayed $d - 1$ times by 1 time-step and $(D - d)$ times by 2 time-steps. To this effect, during period 1 we emit, from time 2 up and every 2 time-steps (Figure 6)

- r-signals till s returns, a total of $d - 1$

- then t-signals till S returns, a total of $D - d$.

At each big period, one signal r or t (the first arriving) is absorbed. Return of S at the end of a period where at least one r signal has started sets a state that delays reflection of v by 1 time-step, and at the end of a period where no signal r but at least one signal t has started delays this reflection by 2 time-steps (here again details are tedious and present no particular interest).

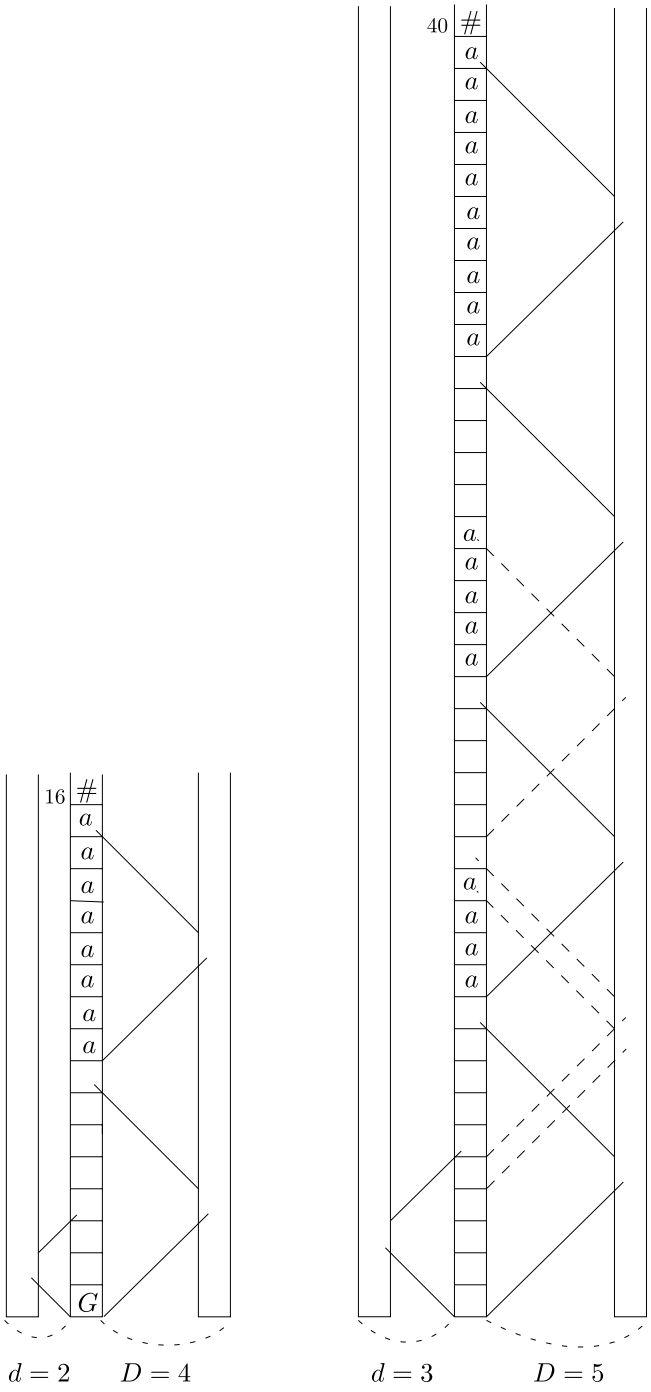


Fig.5 : $T_9(D,d) = 2D(2d-2)$

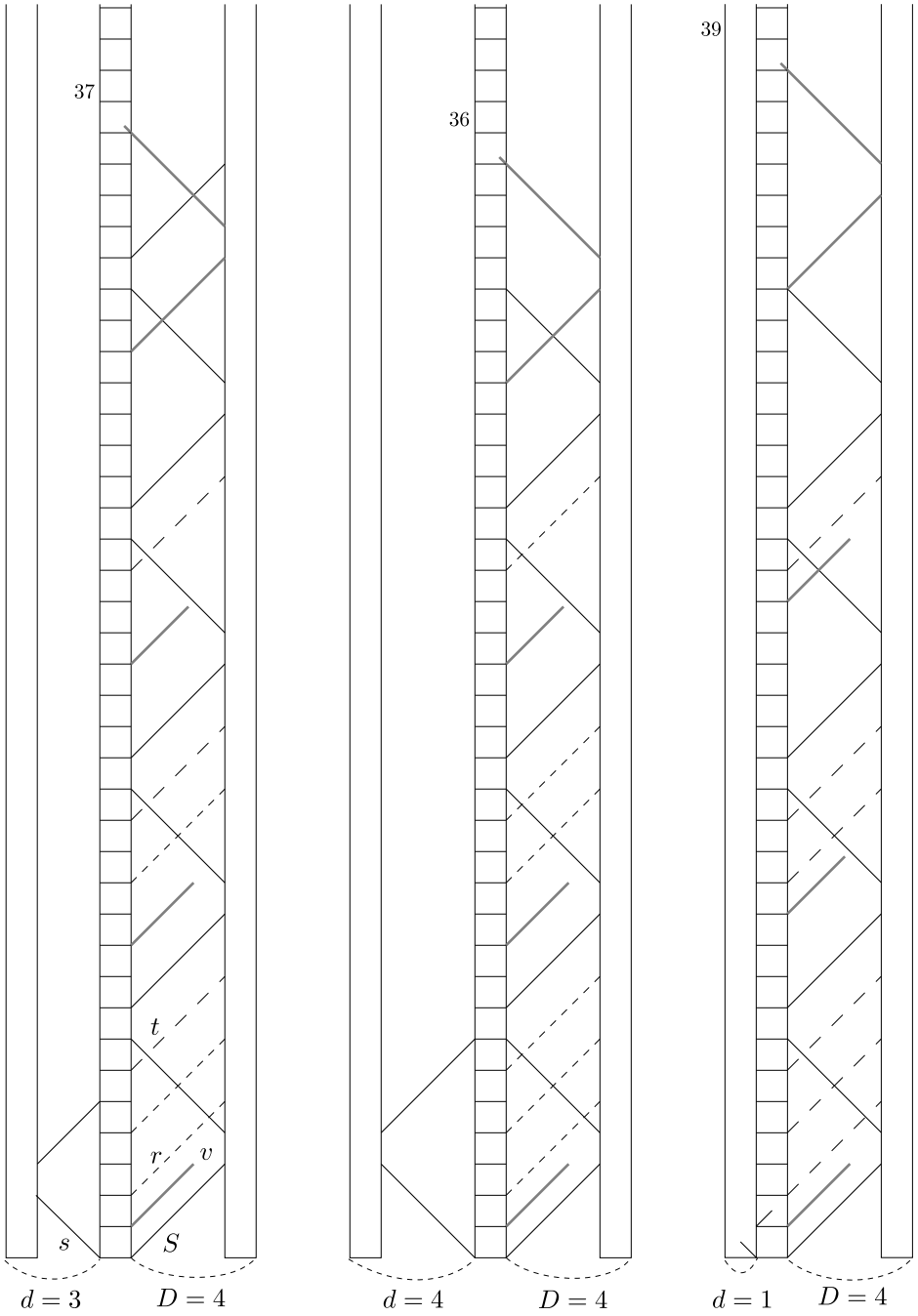


Fig.6 : $T = 2D^2 + 2D - d$

12.3 synchronization of a line of automata with delays in time $2\Delta^2$.

Mazoyer's method uses a simple transposition of Minsky's technique for lines, presented in chapter 1. He breaks the line in two halves at the meeting point of a signal of speed 1/3 with the reflection of a speed 1 signal, to restart a same process on the two sublines, till the sublines are too small to be broken.

Synchronization time will be exactly

$$T(\Delta) = \begin{cases} 2\Delta^2 & \text{if } \Delta > 1 \\ 3 & \text{if } \Delta = 1 \\ 1 & \text{if } \Delta = 0 \end{cases}$$

12.3.1 Breaking in two a line with delay $\Delta > 1$.

The automaton in state general is the first one (i.e. the left one). In the sequel, on the sublines, state G will be either on the left end or on the right end, so we shall have all the rules symmetrical of those we shall next describe.

Here, to realize the equivalent of a signal of speed 1/3 (on the fictitious line) we shall use a (fictive naturally) signal Z running a zigzag between cells. Instead of going directly from cell c to its neighbour cell c', signal Z goes from c to c', then is reflected (Z_1) back to c and finally returns (Z_2) to c'. Z_2 crosses c', thereby becoming Z anew (see Figure 7).

State G at the left border sends rightwards (that is through its connexion to the next automaton) the zigzag signal Z and a fast signal S which crosses all the automata and is at last reflected by the right end automaton (that is by the the right border state) thus becoming R. In addition, signal Z_2 puts all automata in state red (automaton i at time $3\Delta_{1i}$) and signal R puts all automata in state blue (automaton j at time $\Delta + \Delta_{jn}$).

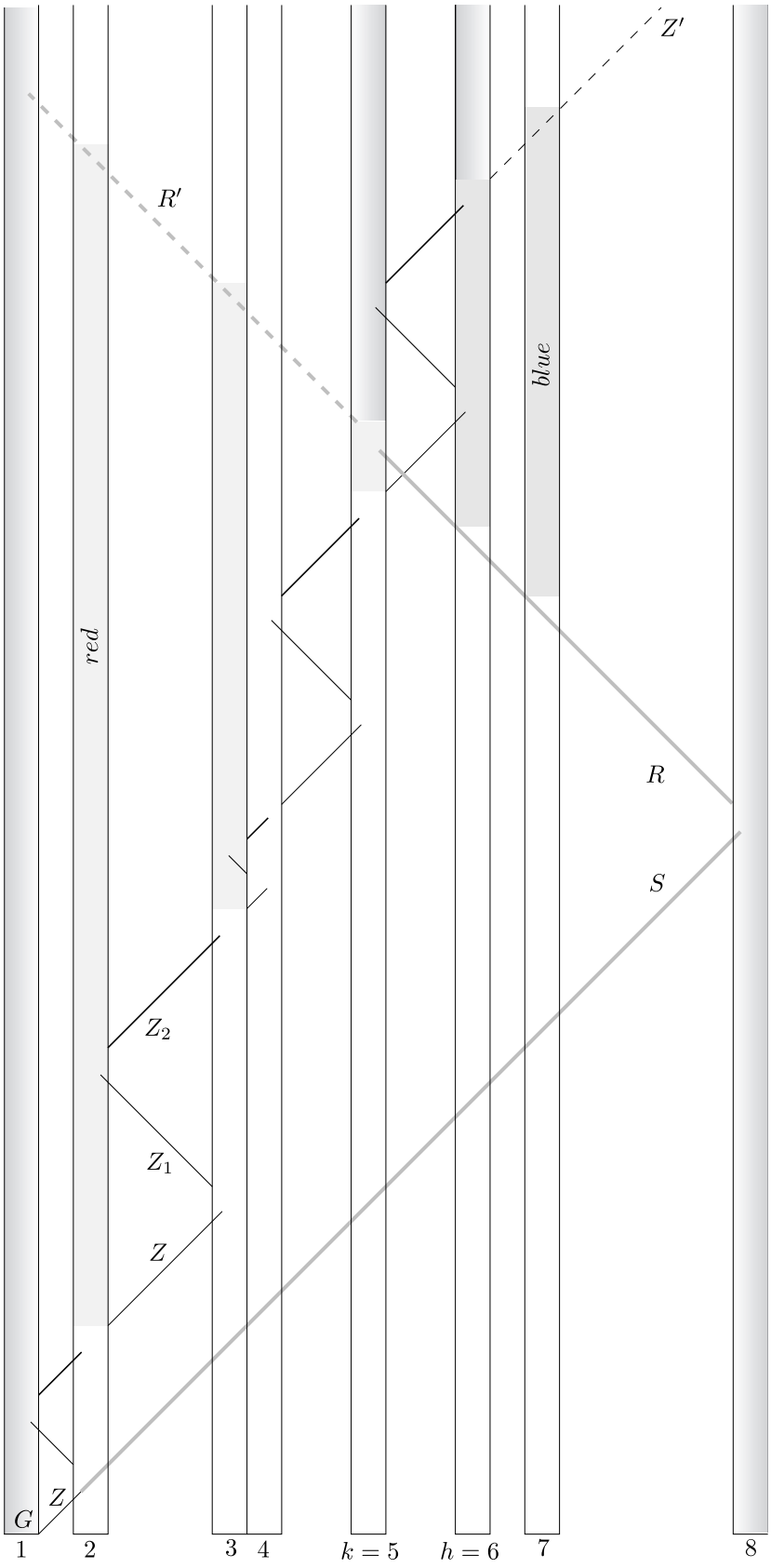


Fig.7

On the left side automata, R falls after Z_2, so on a red state or on the left border, and on the right side automata, on the contrary, Z_2 comes after R, that is on a blue state or on the right border. This circumstance will permit us distinguish the left and right automata. Let us note that Z_2 and R may arrive at the same time on some automaton k, this if and only if

$$3\Delta_{1k} = \Delta + \Delta_{kn}$$

that is

$$\Delta_{1k} = \Delta_{kn}$$

that is if automaton k is exactly at the middle (equidistant in delay from the two extremities). In this case automaton k passes in state "middle" or m (at time $3\Delta_{1k} = \Delta + \Delta_{kn}$) and signals Z_2 and R, after having crossed each other become Z' and R' (Figure 8). The line is broken in two half-lines $[1, k]$ and $[k, n]$ equal (in delay), and automaton in state m will act as right border for signals coming from the left and as left border for signals coming from the right.

If there is no automaton right at the middle there will be

- a first automaton k on which R falls on a red state
- and a first automaton $h = k + 1$ on which Z_2 falls on a blue state

k is the greatest integer such that

$$3\Delta_{1k} < \Delta + \Delta_{kn}$$

that is, since $\Delta = \Delta_{1k} + \Delta_{kn}$,

$$\Delta_{1k} < \Delta_{kn}$$

while h is the smallest integer such that

$$3\Delta_{1h} > \Delta + \Delta_{hn}$$

that is

$$\Delta_{1h} > \Delta_{hn}.$$

When k receives R (see Figure 7), it enters state rb (at time $\Delta + \Delta_{kn}$) and R becomes R'. likewise, when h receives Z_2 it enters state lb (at time $3\Delta_{1h}$) and becomes Z'. We can consider that state m is lb and rb together. Signals R' and Z' rub out respectively the red and blue states, after what they vanish when arriving at the borders.

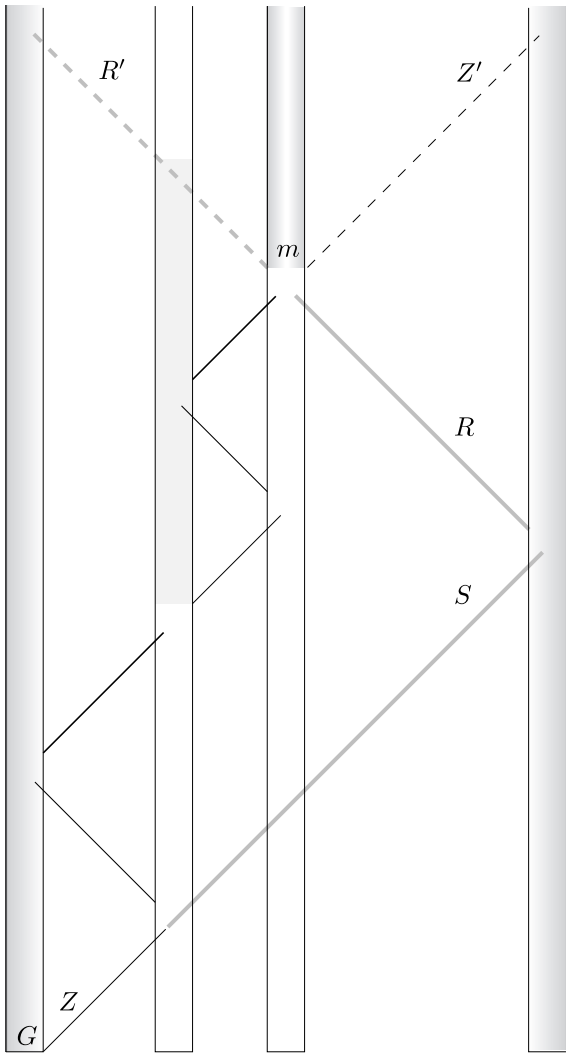


Fig.8 : equal half-lines

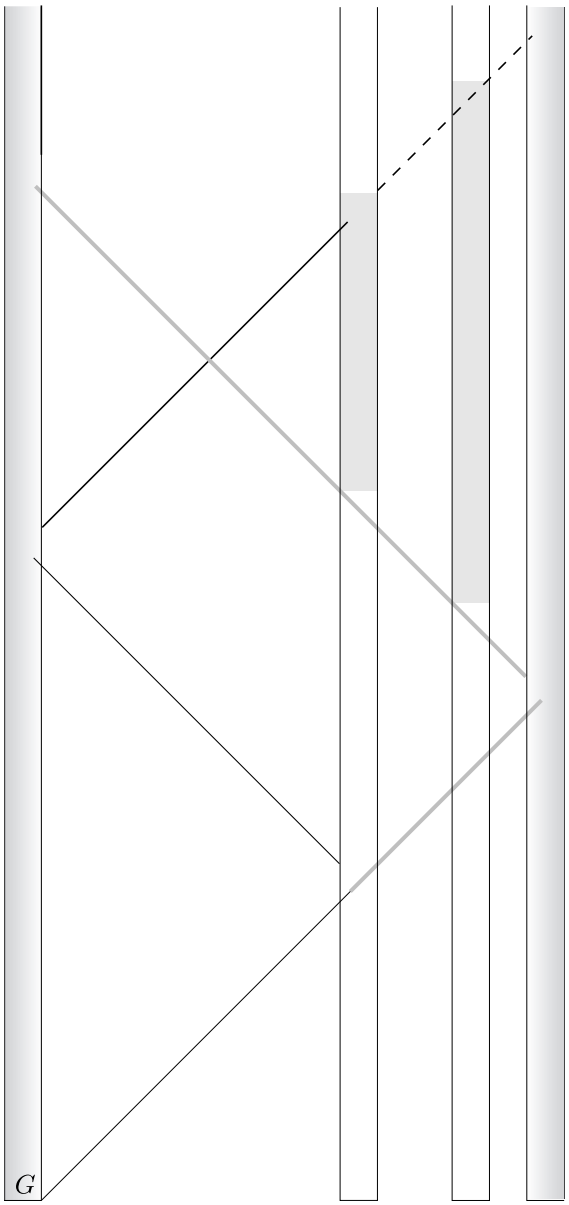


Fig.9 : a half-line reduced to a single automaton

k and h are the two automata just left and just right of the theoretical middle of the line ($h = k + 1$). The half-lines $[1, k]$ and $[h, n]$ are thus the most equal possible. This does not exclude that a half-line can be reduced to the only border automaton (Figure 9).

12.3.2 Synchronization of the very small lines, with delay $\Delta \leq 1$.

These are the lines too small to be broken, and which result from the above breaking process.

Their synchronizations are illustrated in Figure 10.

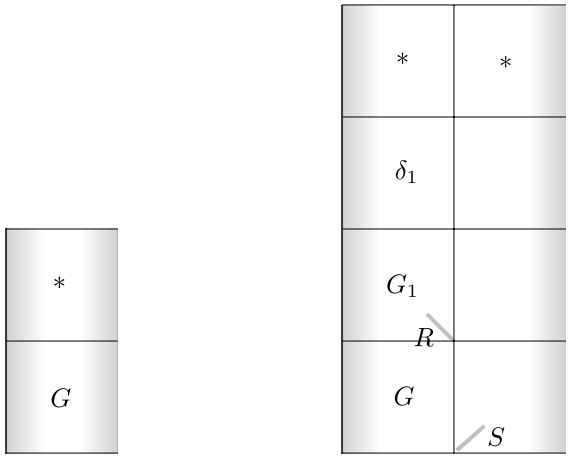


Fig.10

$\Delta = 0$ characterizes a line which is reduced to a single automaton. We have announced that such a line synchronizes immediately. For this we decide that a line which cumulates states left border or *lb*, right border or *rb*, and general G, immediately enters fire state. The word "cumulate" is a practical way of alluding to the fact that our states are in fact products of states.

$\Delta = 1$ characterizes a line which is reduced to 2 automata without delay, in immediate contact with each other. We have announced that such a line synchronizes in 3 time-steps. Let us comment rules of Figure 10 : the immediate return of R suffices to inform the first automaton that $\Delta = 1$, and this one communicates the information immediately, that is one time-step later to the second. To implement this, we decide that the state G becomes state G_1, and that G_1 receiving R becomes state δ_1, and that any automaton in state δ_1 or in contact with this state enters fire state at next time-step.

12.3.3 Locating the half-lines of delay 1

In the sequel we shall have to know, at the moment when a delimiting automaton is determined, if the half-line it determines has delay 0, 1, or more. If this half-line is reduced to a single automaton, we know this because signal R or Z_2 has fallen not on a red or blue state but on a border state. It is less easy to know that the new line consists of two automata in contact (with delay 0).

If the half-line is at the left (see Figure 11) : when the breaking process starts, if Z_1 comes back on G_1, the left automaton knows that it is in contact with the second one, and at next time-step the latter has the information. It then suffices to memorize this information in addition to the red character of the state.

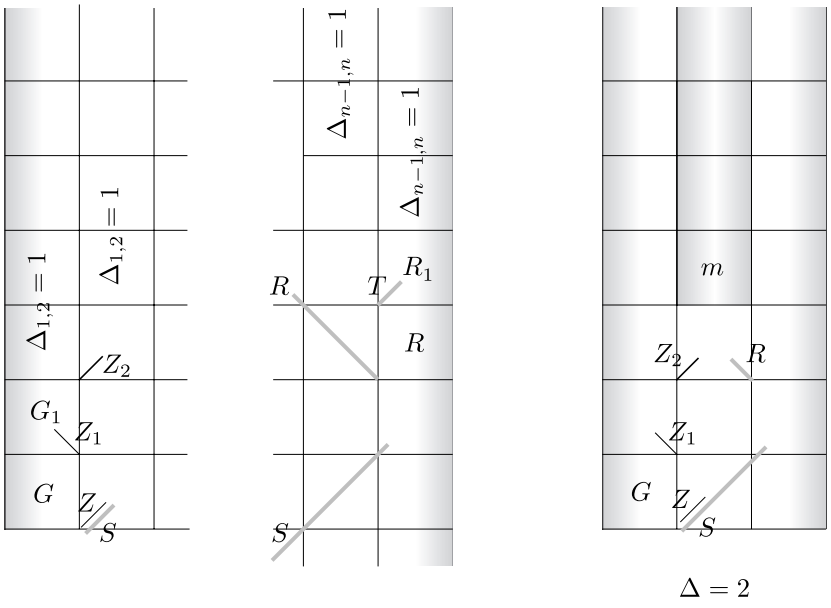


Fig.11

If the half-line is at the right : right end automaton n receiving S enters state R, which sends signal R back and progressively vanishes through state R_1. If R is reflected in T on the first automaton it meets, automaton $n-1$ is in contact with n if and only if T falls on state R_1. This information is then known by n and $n-1$ one time-step later, and can be memorized.

Let us not forget noticing that this will always be early enough, that is before the breaking in two, except in the case of a line of 3 automata with no delay. But in this last case the cell in state middle perceives the two borders, so knows that the two half-lines have delay 1.

12.3.4 Postponing synchronization of half-lines

Half-lines $[1,k]$ and $[h,n]$ $(h = k+1)$ will normally have synchronizing times $T(\Delta_{1k})$ and $T(\Delta_{hn})$. We hope to synchronize the line by using these partial synchronizations. Since these half-lines are created at times $\Delta + \Delta_{kn}$ for $[1,k]$ and $3\Delta_{1h}$ for $[h,n]$, for these sub-synchronizations to happen at the same time and more precisely at time $T(\Delta)$ we must start them with respective delays (Figure 12) :

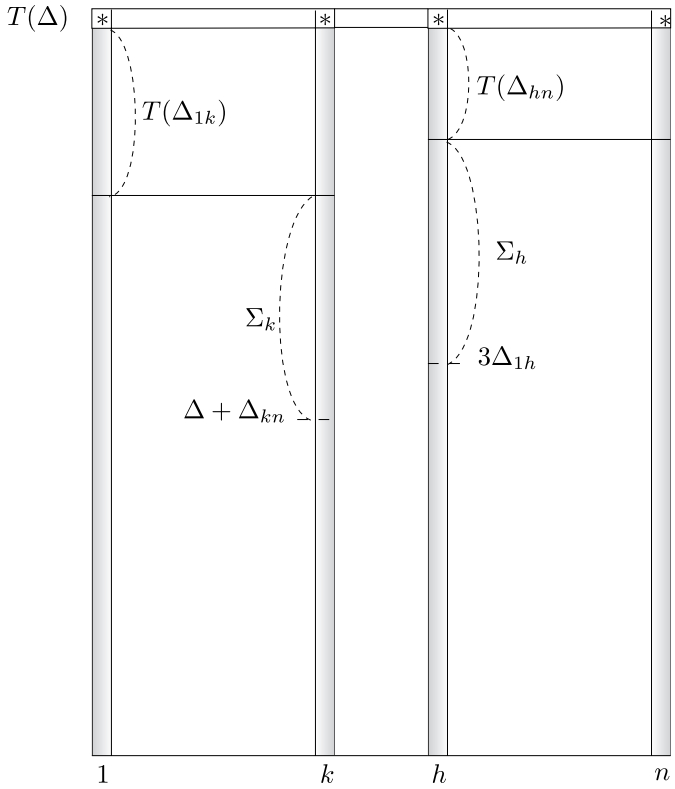


Fig.12

$$\Sigma_k = T(\Delta) - (\Delta + \Delta_{kn}) - T(\Delta_{1k})$$

$$\Sigma_h = T(\Delta) - 3\Delta_{1h} - T(\Delta_{hn}).$$

The line has delay $\Delta > 1$, so $T(\Delta) = 2\Delta^2$. But the half-lines $[1, k]$ and $[h, n]$ may have any delays, 0, 1 or more, so that we shall have different expressions according to the different cases.

For each of the two respective automata k and h, we shall express these delays in terms of the distances to the extremity automata, denoting by D the greatest one and d the smallest one :

$$\text{for } k : d_k = \Delta_{1k} \text{ and } D_k = \Delta_{kn}$$

$$\text{for } h : D_h = \Delta_{1h} \text{ and } d_h = \Delta_{hn}$$

using the fact that $\Delta = D_k + d_k = D_h + d_h$.

We may then obtain the following expressions :

$$\Sigma_k = \begin{cases} [2D_k^2 + 2D_k - d_k] + [2D_k(2d_k - 2)] & \text{if } d_k > 1 \\ 2D_k^2 + 2D_k - 2 & \text{if } d_k = 1 \quad (\Rightarrow D_k \geq 1) \\ 2D_k(D_k - 1) - 1 & \text{if } d_k = 0 \quad (\Rightarrow D_k > 1) \end{cases}$$

$$\Sigma_h = \begin{cases} [2D_h^2 + D_h] + [2D_h(2d_h - 2)] & \text{if } d_h > 1 \\ 2D_h^2 + D_h - 1 & \text{if } d_h = 1 \quad (\Rightarrow D_h \geq 1) \\ 2D_h(D_h - 2) + (D_h - 1) & \text{if } d_h = 0 \quad (\Rightarrow D_h > 1) \end{cases}$$

(If $\Delta_{1k} = \Delta_{kn}$, $k = h$, $D = d$, $\Sigma_k = \Sigma_h = 6D^2 - 3D$, and the case $D = 1$ of a line of 3 cells with no delay is dealt with in 12.3.3).

We observe that these delays are times that our automata k and h can compute by exchanging signals with the extremity automata. Indeed, using the notations from 12.1 and 12.2

$$\Sigma_k = \begin{cases} T_{10}(k,n,1) + T_9(k,n,1) & \text{if } \Delta_{1k} > 1 \\ T_6(k,n) & \text{if } \Delta_{1k} = 1 \\ T_8(k,n) & \text{if } \Delta_{1k} = 0 \text{ and } \Delta_{kn} > 2 \\ 3 & \text{if } \Delta_{1k} = 0 \text{ and } \Delta_{kn} = 2 \end{cases}$$

$$\Sigma_h = \begin{cases} T_4(h,1) + T_9(h,1,n) & \text{if } \Delta_{hn} > 1 \\ T_5(h,1) & \text{if } \Delta_{hn} = 1 \\ T_7(h,1) & \text{if } \Delta_{hn} = 0 \text{ and } \Delta_{1h} > 2 \\ 1 & \text{if } \Delta_{hn} = 0 \text{ and } \Delta_{1h} = 2 \end{cases}$$

In case $d = 0$, we must separate the case when $D = 2$ because then times T_8 and T_7 have different expressions, which are no more the values we want for Σ_k and Σ_h.

Naturally, to perform these computations all the automata must have the states described in the first section.

It is also necessary that at the time when k and h are determined, these automata know what computation they should start, and for this they must know if their half-line has delay greater than 1, or is reduced to two automata, or to a single one, and in this case if $\Delta = 2$.

The half-line is reduced to two automata if the separating automaton is in contact with the corresponding border, information which it may have (in time) as we have seen (12.3.3). The half-line is reduced to a single automaton if the separating automaton is a border, and in this case, we must know if the broken line had delay $\Delta = 2$. For this it is sufficient to know if R and Z_1 return together on G_2, which the first automaton knows at time 3, and communicates to the second who knows it at time 5 (Figure 13) : so the two automata know just in time that their delays must then be 3 and 1.

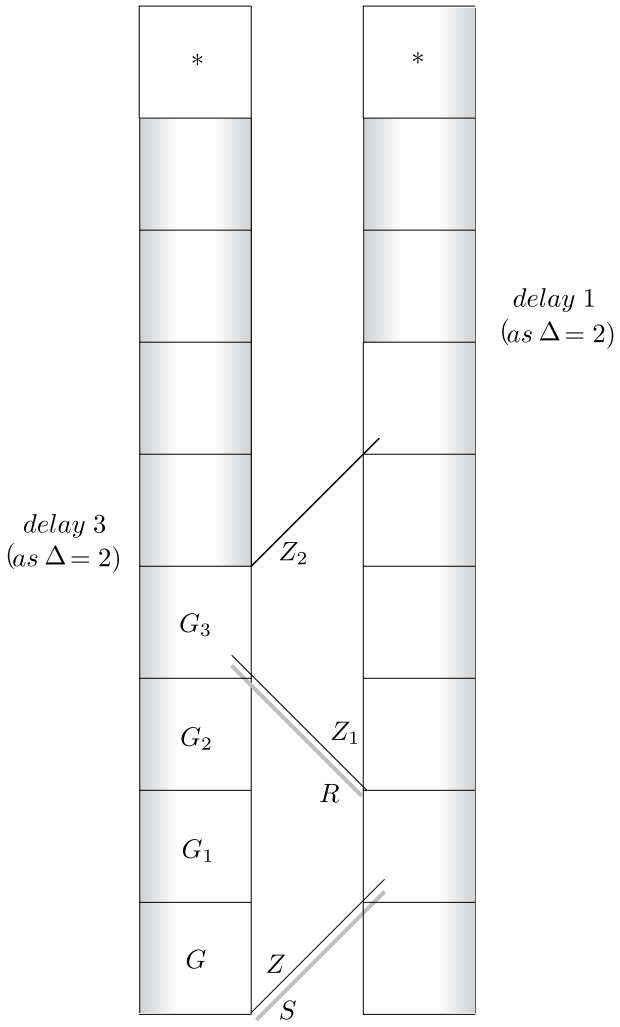


Fig.13

When the separating automata have computed their delays, they become generals for their half-line, with of course all the symmetrical transition rules for the left half-line.

In this way the initial line is progressively broken in very small lines of 1 automaton or 2 siamese automata, all synchronizing at time $2\Delta^2$.

12.4 Comparison with Jiang's general result

Jiang has not treated the case of a line separately. As an instance of the general case of tree-structured networks, for which his solution is in time

$$T_J = 118DR^3 + 1,$$

a line, for which $DR = \Delta$, may so be synchronized in time

$$T_J = 118\Delta^3 + 1.$$

Mazoyer's solution in time $2\Delta^2$ is indeed much faster.

Chapter 13

Synchronization of a network of finite automata

13.1 Definition of a network of finite automata

The word suggests automata interconnected by wires. Our mathematical model must be more precisely defined. Here as in all the preceding chapters the automata are all identical. The basic automaton has d inputs and outputs (abbreviated as IO), numbered 1 to d. The wires connect 2 IO's, and in our particularly simple model, 2 IO's of 2 distinct automata (there are no loops). We also exclude double connections. Wires communicate the state of the automaton at one end to the automaton at the other end, in both directions. Thus the automata and wires form a non-oriented connected graph, with a supplementary precision : the IO's are labeled. An example is given in Figure 1.

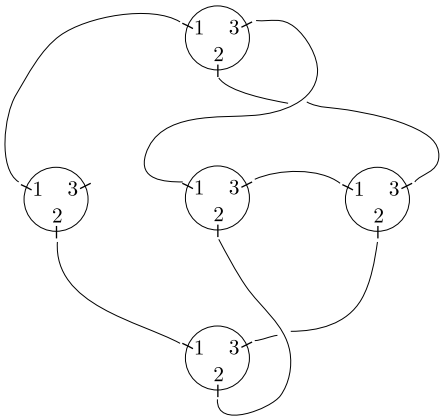


Fig.1

The lines and the $\mathbb{Z}^n$-c.a's of the preceding chapters are particular regular networks

- in lines the labels of the two IO's are "left" and "right"
- more generally, in $\mathbb{Z}^n$-c.a's, nodes have degree 2^n.

The transition function for all cells of the network is

$$< A, t+1 >= \delta(< A, t >, < v_1(A), t >, \ldots , < v_d(A), t >)$$

where $v_i(A)$ is the automaton connected to IO i of automaton A. If this IO is unused, the argument in δ is a symbol for emptyness, (for example $-$), or a special constant state similar to the border state for linear c.a's.

We may further imagine that wires have different lengthes, so that communication through them arrive with delays τ_i. The transition function will then be

$$< A, t+1 >= \delta(< A, t >, < v_1(A), t-\tau_1 >, \ldots , < v_d(A), t-\tau_d >)$$

and we then speak of networks with delays. The line of chapter 12 is the simplest possible example of such a network.

Let us mention that the set of states of the basic automaton always contains a quiescent state, such that any cell in quiescent state surrounded by quiescent neighbours remains quiescent.

As for the lines and $\mathbb{Z}^n$-c.a's of the preceding chapters, one particular cell in the network may receive an input from the outside, and produce an output, which are not to be confused with the inner IO's for interconnections. We may call this cell the origin cell (or cell 0).

13.2 An automaton to set up a spanning tree

13.2.1 Recalling definitions for graphs

For extensive explanations we recommend [8].

A *tree* is a non-oriented connected graph which contains no cycle. A *rooted tree* has a distinguished node, the root, through which it acquires a natural orientation, from the root to the nodes, from fathers to sons. Each node except the root has a father. The nodes which have no sons are the leaves.

A *spanning tree* for a graph, is a subset of the edges which is acyclic, and connects all the vertices.

Note that if the graph has degree d, the root of a spanning tree has at most d sons, and the other nodes of the tree have at most $d-1$ sons.

If the graph has n nodes, the spanning tree has the same n nodes. The number of edges of the tree is then $n-1$, because each edge is between a son and its father, and each node, except the root, has one, and only one, father. Each edge contributes for 2 in the total degree (sum of the degrees of all the nodes), which is thus $2n-2$.

13.2.2 Rules in the case with no delays

Our network has a distinguished cell, the origin. We are interested in finding a spanning tree rooted at the origin.

Let us observe that IO's of cells being labeled, a connection between two cells in the network is characterized by any of the 2 IO's that it connects, and is simultaneously designated and oriented if we give its sole origin IO.

Now we want to find states and transition rules for the automata at the nodes such that the network, starting with all automata quiescent except one, the origin cell, which is in some initial state, becomes, after some time, a tree rooted at the origin.

This means that some edges of the graph have been selected (and oriented). We can indicate that an edge is selected (and oriented) by marking its origin IO.

If at all nodes we know which IO's are origins of oriented edges of the tree, i.e. edges to the sons of the node, then we know the tree. Such information must be given by the states of the automata at each node. So we want states to be able to indicate which IO's are origins of selected edges. The states and transitions rules must also do the work of selecting the tree edges.

Here, the convenient states will have d components corresponding to the d IO's. On each component symbols a and x may appear : a is an infectious state and x marks the IO's through which it is exported, which will be the origins of the tree edges.

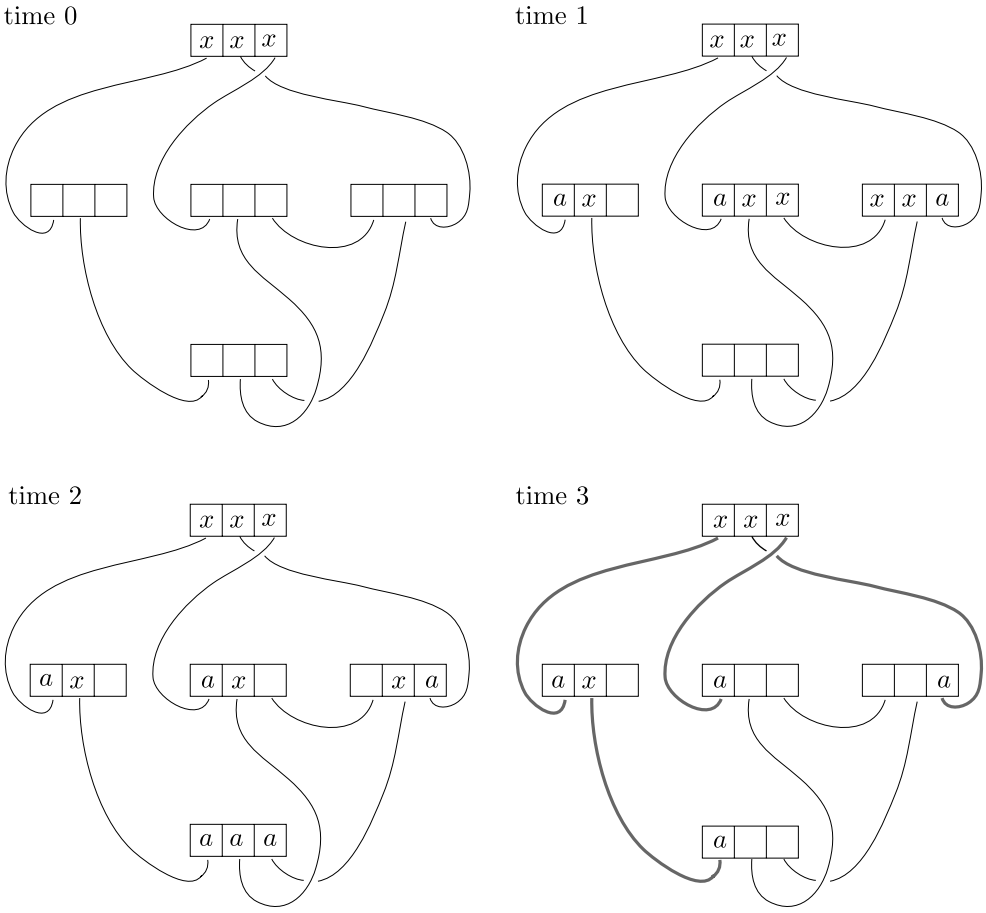


Fig.2a

In the initial configuration all the states are empty, except the state of the origin cell which has x marks at all used IO's. States change according to the following rules (see Figure 2a, corresponding to the network of Figure 1)

- an IO marked x transmits state a, if it arrives through an unmarked IO. Then a is inscribed in the place corresponding to the IO through which it has come. An x mark appears on each used IO through which no a arrives

(note that each connection marked x at its origin is thus marked with a at its end)

- a cannot be transmitted to a cell already in state a. So if an input marked x leads to another input already marked x, both x marks are (symmetrically) destroyed. We shall express things more briefly by saying that if x marks appear on 2 connected IO's, they disappear at next time-step

as a consequence each x IO will lead to an IO with an a state

- if a cell contains more than one a, at next time-step all but the first one are erased (so there always remains at least one), as well as the x marks at the origin of the connections that have exported them (a node in a tree can have only one father).

The above rules set up transition rules since the state of each cell is completely determined by the states at previous time-step of the cell and its neighbours.

13.2.3 The tree

Let us now study the graph formed by the edges marked x at one end and a at the other end, that we shall denote xa-edges

- any cell (other than the origin) sooner or later contains a, as a is transmitted by adjacency and the graph is connected

- each node has only one IO marked a, so that any path (cycles particularly) of xa-edges goes from node to node always from the x of one node to the a of the following node, in direction $x\vec{a}$ (or reversely always in direction $a\vec{x}$)

- but the a at the end of any $x\vec{a}$ edge has arrived one time-step after the x at its origin, so there can be no cycle with such edges

- starting from any cell (other than the origin) and repeatedly climbing from a to x, must end at the origin cell which has no a, because there are no cycles and the number of nodes is finite. Thus all nodes of the graph are connected by the xa edges.

Thus the xa-edges form a tree, as defined in 13.2.1.

This tree is a spanning tree because every node is connected to the origin cell.

At last we want to observe the orientation of the edges of this tree if it is rooted at the origin cell. The path going from the origin, which has no a and only x's, to any node, necessarily starts with an $x\vec{a}$-edge. It must then, as previously noted, continue with $x\vec{a}$ oriented edges. Thus for any node the IO's marked x go to the sons, while the a-IO leads to the father. The leaves of the tree are the nodes with no x-marks, only one a.

It is worth noting that in the tree as it is built here, the branch leading to a node is the shortest path from the origin to this node in the graph. Indeed state a takes place in a cell as soon as it arrives through the shortest path, and the extra a's that we suppress have arrived at the same time as the remaining one.

13.2.4 Timing in the ordinary case

Let us call "radius" of the graph and note R, the maximum distance from the origin cell to another cell, i.e. the length of the shortest path from the origin to the cell in case different paths exist.

State a needs time R to set in all nodes of the graph, then one more time-step is needed if some a states have to be erased. Thus the time needed to establish the tree is at most $R + 1$.

$$T_{tree} \leq R + 1 \leq n$$

because, if the graph has n cells, the radius is certainly at most $n - 1$.

13.2.5 Rules in the case with delays

The same automaton can be used in a network with delays, with minor adaptations

- we change the formulation of the second rule : if state a exported by an x mark arrives to an IO already marked x, then it simply destroys this x mark

- we note for the last rule that when extra a's are erased, a delay is needed to return to the cell where the corresponding x marks must also be erased

In figure 3a we have added indexes to symbols a and x indicating the time at which they have appeared, to help follow the process.

We can note here, that as in the ordinary case, and for the same simple reason, the delay from the root to a node of the tree is the shortest delay from the origin to this node in the graph (the tree is a *minimum-delay spanning tree*).

This explains that, though the graph in Figure 3a is the same as that in Figure 2a, the associated trees are different.

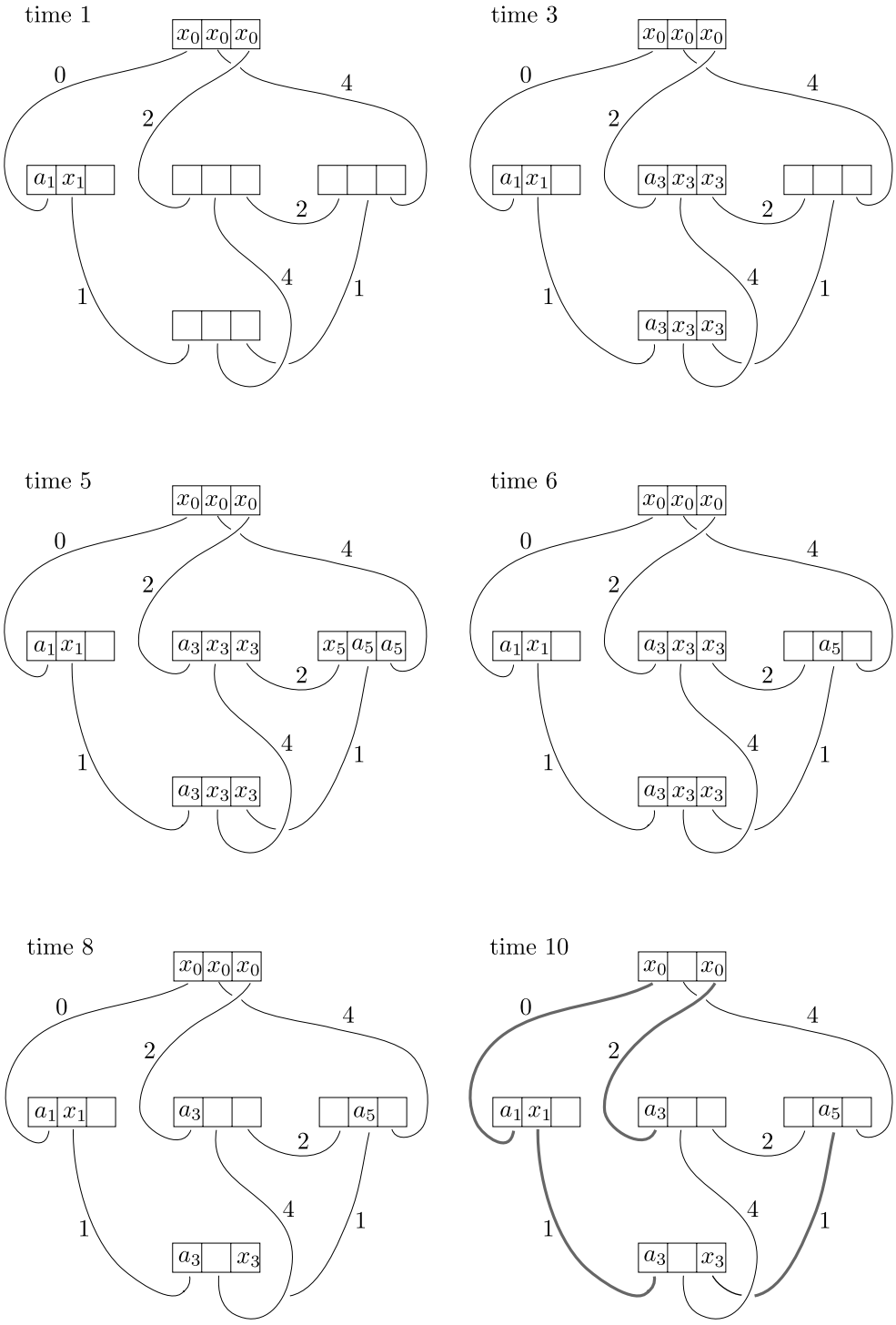


Fig.3a

13.2.6 Timing

We recall that, if the delay on a connection is τ, the time needed for a state to travel through the connection and set in the arrival cell, ready to depart through another connection, is $\delta = \tau + 1$, so that the delay to go through a path where connections have delays τ_1, τ_2, ... , τ_p and set on the last cell, is $\delta_1 + \delta_2 + \ldots + \delta_p$.

Jiang [30] calls "delay-radius", and we shall denote DR, the maximum delay between the origin and any other node. The latter being the shortest delay between the origin and this node if several paths lead from the one to the other. He denotes τ_{max} the maximum delay of a connection, and we shall denote $\delta_{max} = \tau_{max} + 1$ what we have called the corresponding reaction delay. Certainly $DR \leq (n-1)\delta_{max}$.

The time needed for state a to set on all cells, up to the furthest cell, is DR, then possibly extra a states must be erased as well as the corresponding x marks, which needs returning back to fathers (on connections not belonging to the tree edges), which takes a time at most δ_{max}. If we denote TD (total delay) the sum of all delays δ_i in the graph, we have

$$T_{tree} \leq DR + \delta_{max} \leq TD$$

13.3 Improving/Completing the automaton

Is it possible for the origin cell to be advised that the process of determining the spanning tree is accomplished ?

This tree is entirely determined if every node of the graph has received state a and the last interconnected x marks have been erased. The leaves of the tree then contain one single a and no x.

Now, if some cell has sons which are all leaves, the subtree under this cell is entirely determined. If these leaves inform their father that they are leaves, the cell knows that its subtree is determined. Likewise, if a cell knows that its sons's subtrees are all determined, then it knows that its proper subtree is determined. We shall thus carry up such information, from sons to fathers, from the leaves to the root.

The automaton to do this will be the same without or with delays.

To the preceding automaton determining the tree we add a new symbol, x with a cercle around it, this cercle denoting that the subtree under the corresponding son is completed (see Figures 2b and 3b)

And the rule : if the state of a cell contains a but no x, (or only cercled x's), the x mark at the IO of the father of this cell is cercled at next time-step/or after the delay to return to the father. In case the state contains more than one a, only the x mark to the first a will be circled, the extra a's are erased, and the x marks leading to them also.

When all the x marks of the root are cercled, the origin cell knows that the tree is completed. Figures 2b and 3b are the continuation of figures 2a and 3a with the automaton as we have just completed it.

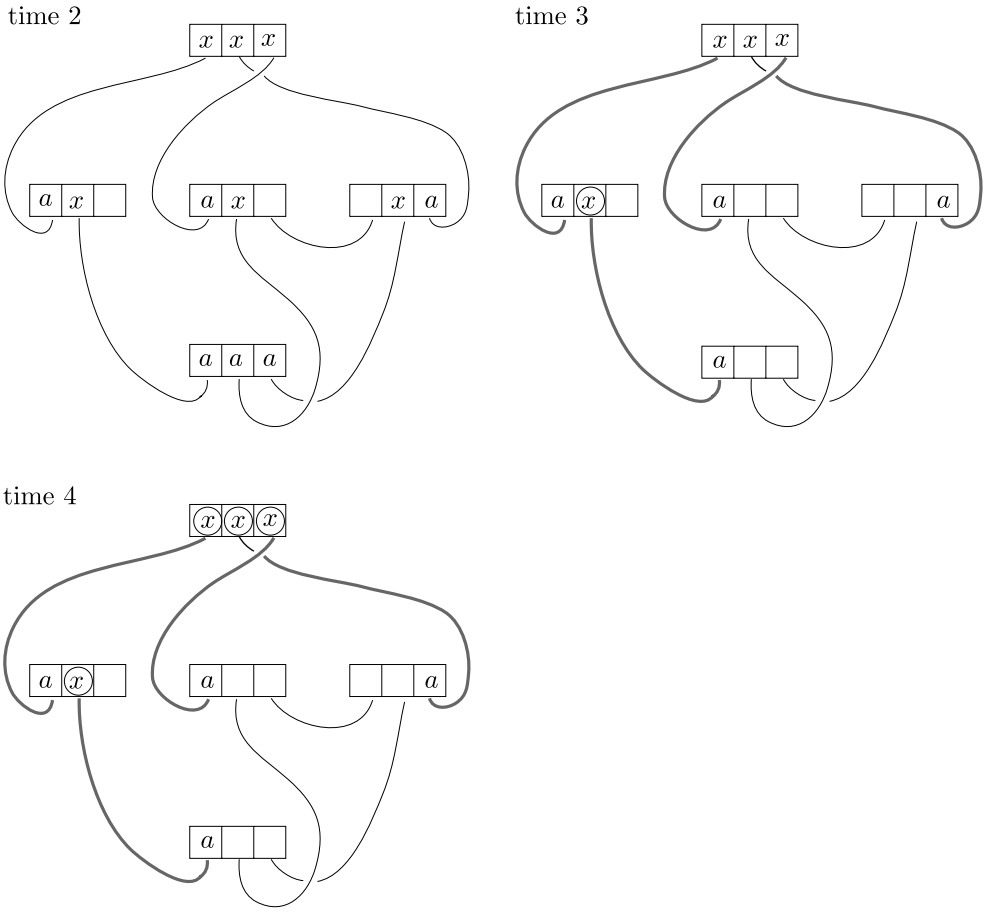


Fig.2b

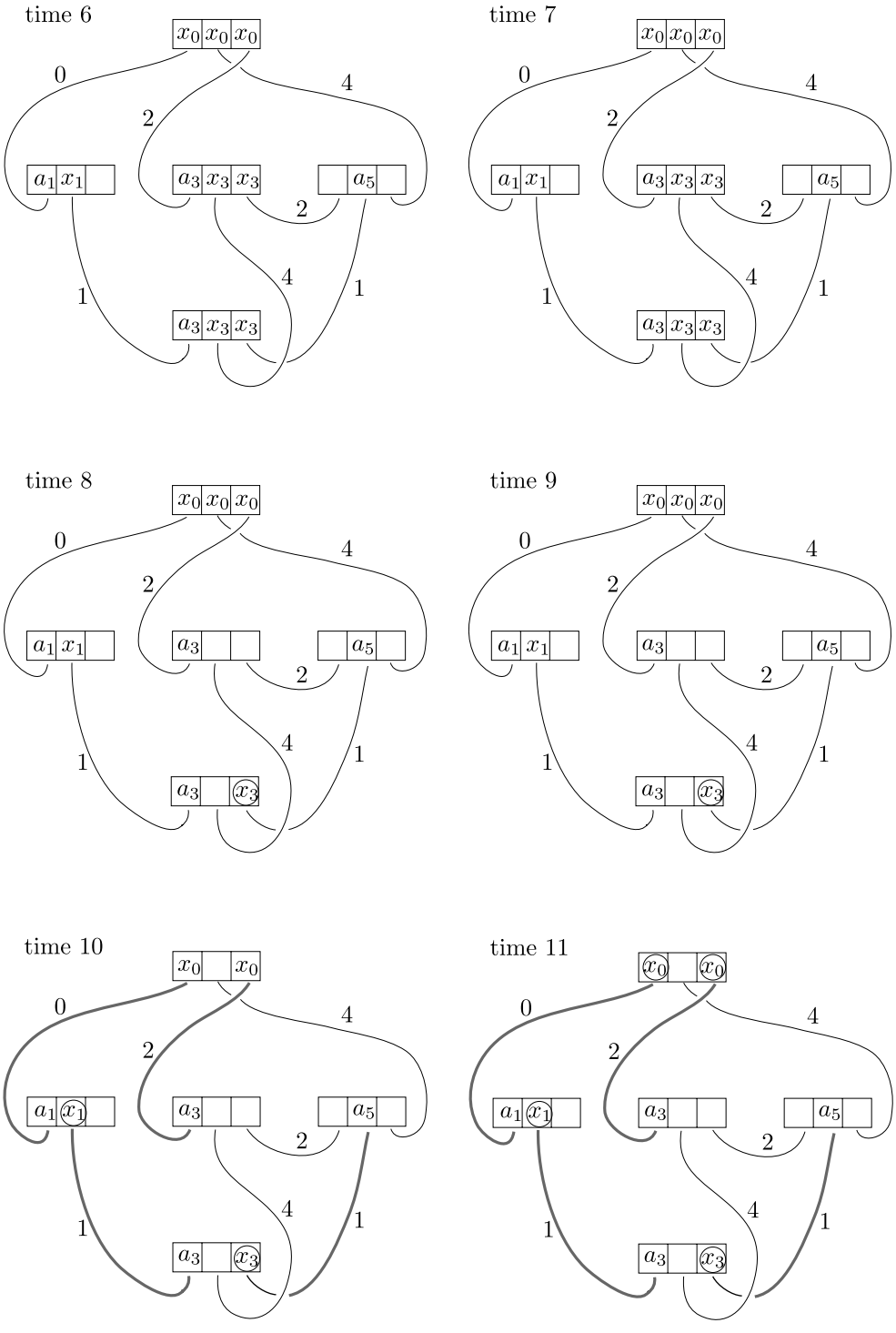


Fig.3b

The time needed to bring back this information to the origin (without waiting for the interconnected x's to be erased), is the same time as was needed for state a to go down to the furthest leave, R or DR . So that the total time to build the tree and warn the origin cell that this task is achieved is, in the case without delays

$$T_{tree-root} = 2R$$

and in the case with delays

$$T_{tree-root} = 2DR.$$

13.4 Automata solving graph problems

In the preceding section we have built a finite automaton such that, if we place a copy of this automaton at each node of any graph of maximum degree d, and connect them in any manner so that the wires follow the edges (IO's may be interchanged), and place the origin automaton in some determined initial state while all others are quiescent, after some finite time (depending on the number of nodes of the graph and the possible delays on the connections), the automata give us the solution of a graph problem.

Rosenstiehl had the idea of such automata since 1966 [47], and has applied it to various graph problems in many papers later on.

Using Rosenstiehl's idea we have built here a very simple solution for the problem that serves our present purpose. But we are not yet finished.

13.5 Making the tree into a line

The idea is to use a depth-first search of the tree (see Figure 4).

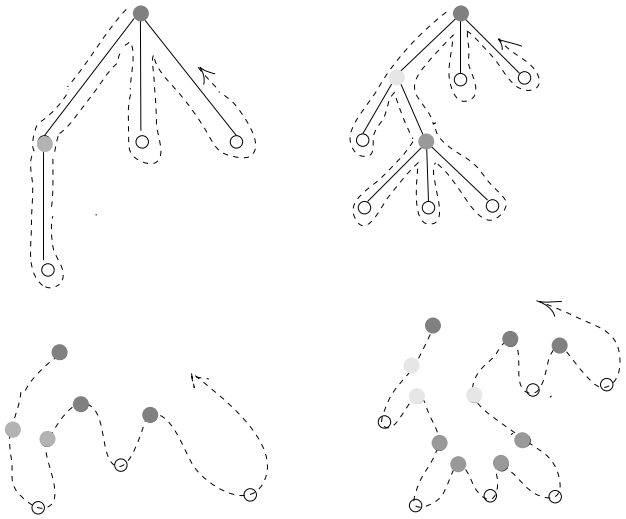


Fig.4

Let us first observe that in moving around a tree, each node will be met a number of times equal to its degree, because after first arriving to this node we come back to it after having quitted each of its sons, and the number of sons is the degree minus 1. The root indeed has no father, so has a number of sons equal to its degree, but we do not come back to the root. Thus the line will have a number of cells equal to the sum of the degrees of all the nodes of the tree, which is $2n - 2$.

As yet the tree is determined by the states on the automata at the nodes, which have d components corresponding to the d IO's, with one component a (except for the root) on the IO leading to the father, and several components x (except for the leaves) on the IO's leading to the sons, the other components being void.

In a first stage we just virtually split these states in pieces by considering that each a or x-component is a node of the line (see Figure 5), and make precise how the preceding and succeeding nodes of this node are to be found. We find them in accomplishing the depth-first search, and precising that each time we go through a node we go through its first/next x-component, or, when there are no more of these, through its a-component.

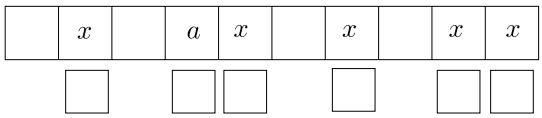


Fig.5

Some notations will be useful :

- nodes of the tree will be denoted R(the root), N, M, F ...
- the a-component of a node N is denoted $a(N)$
- if the state of a node N has k x-components, we denote them here

$$x_1(N), \ldots, x_i(N), \ldots, x_k(N)$$

in the order of the components, which is the order of the numbers of the IO's, and will also be the order of the sons of N

- the son under $x_i(N)$ will be denoted $\mathcal{S}(x_i(N))$
- each node $N \neq R$ has a father F such that $N = \mathcal{S}(x_j(F))$. We may also write $x_j(F) = \mathcal{F}(N)$.

We obtain the successor and the predecessor of a node simply by describing the well-known depth-first path through the tree. The i-th time we go through N we go through component $x_i(N)$.
This gives

$$succ(x_i(N)) = \begin{cases} x_1(\ \mathcal{S}(x_i(N))\ ) & \text{if } \mathcal{S}(x_i(N)) \text{ has an } x\text{-component} \\ a(\ \mathcal{S}(x_i(N))\ ) & \text{otherwise} \end{cases}$$

$$succ(a(N)) = \begin{cases} x_{j+1}(F) & \text{if } x_j(F) = \mathcal{F}(N) \text{ and } F \text{ has more than } j\ x\text{-components} \\ a(F) & \text{otherwise} \end{cases}$$

and

$$pred(a(N)) = \begin{cases} x_j(F) & \text{if } N \text{ has no } x\text{-component} \\ a(\ \mathcal{S}(x_k(N))\ ) & \text{if } x_k(N) \text{ is the last } x\text{-component of } N \end{cases}$$

$$pred(x_i(N)) = \begin{cases} x_j(F) & \text{if } i = 1 \text{ and } N = \mathcal{S}(x_j(F)) \\ a(\ \mathcal{S}(x_{i-1}(N))\ ) & \text{if } i > 1 \end{cases}$$

The last cell of the line is $a(\mathcal{S}(x_k(R))$, where $x_k(R)$ is the last x-component of the root.

Now that this is established, let us explain more clearly how the line will work, that is how the states of the automata on the nodes of the graph simulate the behavior of the line. It is in fact quite simple : the preceding d-component states, which determine the tree and the line, remain as they are. The new states will be obtained by a second sequence of d components, that we can imagine under the first ones, where the states of the line-nodes take place (the places under the void places are not used).

$-$	x_1	$-$	x_2	a	x_3	$-$	$-$	x_4
$-$	q_1	$-$	q_2	a	q_3	$-$	$-$	q_4

The state q of each line-node of automaton A evolves according to the states of its predecessor and successor, which it recognizes among the components of the new states of automata which are connected to A.

We may conclude by underlining that no supplementary time has been needed to get hold of the line, when the tree is completed. So that the total time to have the line is, in the case without delays

$$T_{tree-line} = 2R$$

and in the case with delays

$$T_{tree-line} = 2DR.$$

What is the length of the line ? We already know that the line has $2n-2$ nodes. In the case with delays we may bound the total delay of the line

- by 2 times the sum of all the tree edges, which we may call the total delay of the tree TTD, but which we do not know

- by $(2n-3)\delta_{max}$, which is best if the delays are not too different and the graph has many edges

- by 2 times the total delay TD of the graph, which is best if the delays are very inequal and the graph has few edges

13.6 Synchronization of a network

13.6.1 The automaton

And now that our network has become a line, we want to synchronize this line as we know to do it

- in the ordinary case by Mazoyer's minimal time solution of chapter 2 which will take time

$$T_{sync-line} = 2(2n-2) - 2 = 4n - 6$$

- in the case with delays by Mazoyer's solution of chapter 12 which will take time less than

$$T_{sync-line} \leq 2(2TD)^2 = 8TD^2$$

$$T_{sync-line} \leq 2(2n\delta_{max})^2 = 8n^2\delta_{max}^2$$

The states on the automata of the graph will be as described just before, from the beginning. The second line of d components may carry quiescent states (the quiescent states of the synchronization of the line) as long as the tree is not completed and the root not advised of this. As soon as the root R knows

that the tree is completed, that is when all its x-components are cercled, more precisely at next time-step, state G, the general for the synchronization of the line, is set under the first x-component of R, which is the first cell of the line. Then the synchronization process of the line takes place. It ends with state fire appearing simultaneously under all x and a-components of all nodes.

13.6.2 Time for synchronizing in the ordinary case

$$T = T_{tree-line} + 1 + T_{sync-line} = 2R + 1 + 4n - 6 = 6n - 7$$

We may further observe that the 2 ends of the line may act as generals in opposite directions for 2 half lines stopping when they bump in each other. Then the synchronizing time for each of the two half-lines is $2n-4$ and the total time for synchronization is less than $2(n-1) + 1 + (2n-4) = 4n - 5$.

Rosenstiehl manages to realize synchronization in better time, even less than $2n$ in [48], but with much more strain.

13.6.3 Time for synchronizing in the case with delays

$$T = T_{tree-line} + 1 + T_{sync-line}$$

$$\leq 2DR + 1 + 8TD^2$$

or

$$\leq 2DR + 1 + 8n^2\delta_{max}^2.$$

Theorem 13.6.1 (Mazoyer-Rosenstiehl) *There exists an automaton which synchronizes any network of degree at most d and arbitrary non-uniform delays, in time bounded by*

$$\leq 2DR + 8TD^2 + 1$$

or

$$2DR + 8n^2\delta_{max}^2 + 1.$$

Jiang's result in [30] is in time

$$118DR^3 + 2DR + 2\delta_{max} + 3$$

The difference is in the process to synchronize the spanning tree. Unfolding the tree into a line as we have done it is clearly a bad method, while using subtrees seems adapted and natural, in case the graph is more like a tree than like a line.

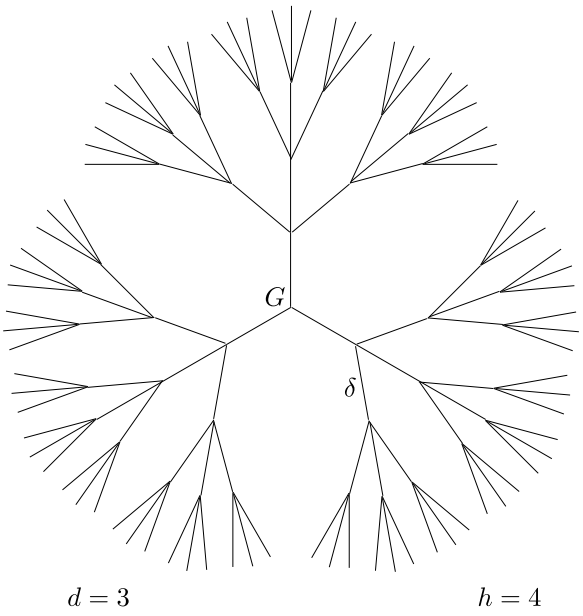


Fig.6

Indeed, for such an example as represented in Figure 6, where $DR = h\delta$ and $TD = d\,\frac{d^h-1}{d-1}\delta$, the rate between Mazoyer-Rosenstiehl and Jiang's times is about

$$\frac{1}{15}\,\frac{1}{\delta}\,(1+\frac{1}{d-1})^2\,\frac{d^{2h}}{h^3},$$

which gets worse as the height h of the tree grows.

But, as we have noted at the end of chapter 12, Mazoyer's time is much better for a line, $8\Delta^2$ versus $118\Delta^3$, if Δ is the total delay of the line. The advantage of Mazoyer's method is certainly in the techniques of time calculations between pairs and trios of connected automatas, which are much more complicated when subtrees are involved. Their implementation, extensively sketched by Jiang [30], outgrows the scope of the present work.

Bibliography

[1] ATRUBIN A.J.

A one-dimensional real-time iterative multiplier

IEEE Transactions on Electronic Computers EC-14 : pp394-399, 1965.

[2] BALZER Robert

An 8-state minimal-time solution for the firing squad synchronization problem

Information and Control 10 : pp22-42, 1967.

[3] BEYER W.T.

Recognition of topological invariants by iterative arrays

Ph.D Dissertation. MIT. Cambridge, Mass., 1969.

[4] BURKS Arthur W.

Essays on Cellular Automata

University of Illinois Press, Urbana, Illinois, 1970.

[5] CODD E.F.

Cellular automata

Academic Press, Inc. New York and London, 1968.

[6] COLE Stephen N.

Real-time computation by n-dimentional iterative arrays of finite-state machines

IEEE Transactions on Computers C-18, No 4 : pp349-365, 1969.

[7] BERLEKAMP E.R, CONWAY John, GUY Richard K.

Winning Ways for your mathematical plays, vol 1

A.K.Peters

[8] CORMEN Thomas, LEISERSON Charles, RIVEST Ronald

Introduction to Algorithms

MIT Press, Cambridge, Mass., 1990.

[9] CULIK II Karel

Weighted growth-functions of DOL-systems and growth-functions of parallel graph rewriting systems

Research Report CS-74-24, Dept. of Computer Science, University of Waterloo, Ontario, Canada, 1974.

[10] CULIK II Karel and YU Sheng

Undecidability of CA Classification Schemes

Complex Systems 2 : pp177-190, 1988.

[11] CULIK II Karel and CHOFFRUT Christian

On real-time Cellular Automata and Treillis Automata

Acta Informatica 21 : pp393-407, 1984

[12] DEHORNOY Patrick

Mathématiques de l'informatique

Collection Sciences Sup, Dunod, 2000.

[13] DUPRAT Jean

Proof of correctness of the Mazoyer's solution of the firing squad problem in Coq

ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR2002/RR2002-14.ps.Z

L.I.P, ENS de Lyon, 46 allée d'Italie 69364 Lyon, France, 1998.

[14] EILENBERG Samuel

Automata, languages and machines

Academic Press 1974

[15] FATES Nazim

Les Automates Cellulaires : vers une nouvelle épistémologie

Mémoire de D.E.A - Université Paris I Sorbonne , 2001

[16] FISCHER Patrick C.

Generation of primes by a one-dimensional real-time iterative array

Journal of the Assoc. Comput. Mach. 12(3) : pp388-394, 1965.

[17] GARDNER Martin

The fantastic combinations of John Conway's new solitaire game of "life".

Mathematical games Department. Scientific american 223 : 120-123, 1970.

[18] GARDNER Martin

On cellular automata, self-reproduction, the Garden of Eden and the game of "life".

Mathematical games Department. Scientific american 224 : 112-117, 1971.

[19] GINSBURG Seymour

The mathematical theory of context-free languages

Mac Graw-Hill New York 1966

[20] GINZBURG Abraham

Algebraic theory of automata

Academic Press 1968

[21] GOTO Eiichi

A minimum-time solution of the firing squad problem

course notes for applied mathematics, vol 298, Harvard University, Cambridge, MA, 1962

[22] GREFFENSTETTE John J.

Network Structure and the Firing Squad Synchronization Problem

Journal of Computer and System Sciences 26 : pp139-152, 1983.

[23] HEEN Olivier

Economie de ressources sur automates cellulaires

Thèse, Université Paris 7, 2 place Jussieu, Paris F-75005, 1996.

[24] HEEN Olivier

Linear Speed-Up for Cellular Automata Synchronizers and Applications

Theoretical Computer Science Volume 188, Numbers 1-2 : pp 45-57, 1997

[25] HEEN Olivier

Efficient constant speed-up for one dimensional cellular automata calculators

Parallel Computing 23 : pp1663-1671, 1997

[26] HILLIS Daniel

Un ordinateur parallèle : *la Connection Machine*

Pour la Science Août 1999.

[27] HOPCROFT J.E. and ULLMAN J.D.

Formal languages and their relation to automata

Addison-Wesley, Reading, Mass., 1969.

[28] IBARRA O.H.,KIM S.M., MORAN S.

Sequential machine characterization of treillis and cellular automata and application

SIAM J.Comput. 14.2 : p426, date ??

[29] JIANG Tao

The Synchronization of Non-Uniform Networks of Finite Automata

30th Symposium on Foundations of Computer Science 1989, pp376-381, 1989

[30] JIANG Tao

The synchronization of non-uniform networks of finite automata

Information and Computation 97(2) : pp234-261, 1992.

[31] KNUTH Donald

The art of computer programming : Fundamental algorithms

Addison-Wesley, Reading, Mass., 1997.

[32] KNUTH Donald

The art of computer programming : semi-numerical algorithms

Addison-Wesley, Reading, Mass., 1997.

[33] KOBAYASHI Kojiro

On the minimal firing time of the f.s.s.p. for polyautomata networks

Theoretical Computer Science 7 : pp149-167, 1978.

[34] LANG Serge

Algebra

Addison-Wesley, Reading, Mass., 1997.

[35] MAZOYER Jacques

A six-state minimal time solution to the f.s.s.p.

Theoretical Computer Science 50 : pp183-237, 1987.

[36] MAZOYER Jacques

An overview of the FSSP

in Automata Networks

C.Choffrut Ed., Springer Verlag, Berlin, New York, pp82-93,1986

[37] MAZOYER Jacques

Synchronization of a line of finite automata with non uniform delays

L.I.P, ENS de Lyon, 46 allée d'Italie 69364 Lyon, France, 1986.

[38] MAZOYER Jacques

Synchronization of two finite automata

L.I.P Report : pp92-34, 1992.

[39] MINSKY M.

Finite and Infinite Machines

Prentice Hall, pp28-29, 1967

[40] MOORE Edward F.

Machine Models of self reproduction

Proceedings of Symposia in Applied Mathematics 14 : pp17-33, 1962

[41] MYHILL John

The converse of Moore's Garden-of-Eden theorem

Proc. AM. Math. Soc 14 : pp685-686, 196?

[42] MYHILL John

The abstract theory of self-reproduction

Views on General Systems Theory, pp106-118 Proceedings of the Second Symposium at Case Institute of Technology, 1964. (Essay 8 of Burks).

[43] POUNDSTONE William

The Recursive Universe

Oxford University Press, 1985

[44] RABIN M.O and SCOTT D.

Finite automata and their decision problems.

IBM Journal of research and developement vol. 3 : pp114-125, 1959.

[45] REIMEN Nicolas

Contribution à l'étude des automates cellulaires

Thèse, Université Paris 7, 2 place Jussieu, Paris F-75005, 1993.

[46] REIMEN Nicolas and MAZOYER Jacques

A Linear Speed-Up Theorem for Cellular Automata.

Theoretical Computer Science 101(1) : pp59-98, 1992

[47] ROSENSTIEHL P.

Existence d'automates finis capables de s'accorder bi-enqu'arbitrairement connectés et nombreux

Internat. Comp. Centre 5 pp245-261 (1966)

[48] ROSENSTIEHL P., FIKSEL J.R, HOLLIGER A.

Intelligent graphs : networks of finite automata capable of solving graph problems

Graph Theory and Computing (R.C.Read, Ed.), Academic Press, New York : pp.219-265, 1972

[49] SETTLE Amber and SIMON Janos

Smaller solutions for the firing squad

Theoretical Computer Science 276 : pp83-109,2002

[50] SMITHAlvy R.

Cellular automata complexity trade-offs

Information and Control 18(5) : pp466-482, 1971.

[51] SMITH ALVY R.

Real-time language recognition by one-dimensional cellular automata

Journal of Computation and System Sciences 6(3) : pp233-253, 1972.

[52] STERN Jacques

Fondements mathématiques de l'informatique

Mac Grawhill 1990

[53] TERRIER Véronique

Real time recognition with cellular automata : a meaningful exemple

L.I.P, ENS de Lyon, 46 allée d'Italie 69364 Lyon, France, 1990.

[54] TERRIER Véronique

Temps réel sur automates cellulaires

Thèse, Université Lyon 1, 1991.

[55] ULAM Stanislaw

On some mathematical problems connected with patterns of growth of figures.

Proceedings of Symposia in Applied Mathematics 14 : pp215-224,1962. (Essay 9 of Burks).

[56] UMEO Hiroshi

A note on firing squad synchronization algorithms

Proc. of Cellular Automata Workshop, Kutrieb & Worsch : pp65, 1966.

[57] UMEO Hiroshi, MAEDA Masashi, FUJIWARA Norio

An Efficient Mapping Scheme for Embedding Any One-Dimensional Firing Squad Synchronization Algorithm onto Two-Dimensional Arrays

ACRI 2002 (Cellular Automata for Research and Industry), Geneva, Stefania Bandini, Bastien Chopard, Marco Tomassini (Eds.) *Lecture Notes in Computer Science*2493 : pp69-81, Springer 2002

[58] VIVIEN Hélène

A quasi-optimal time for synchronizing two interacting finite automata

Journal of Algebra and Computation 6(2) : pp261-267, 1996

[59] VON NEUMANN John

Theory of Self-Reproducing Automata

University of Illinois Press. Edited and completed by Arthur W.Burks, 1966.

[60] WAKSMAN Abraham

An optimum solution to the firing squad synchronization problem

Information and Control 9(1) : pp66-78, 1966.

[61] WOLFRAM Stephen

Universality and Complexity in Cellular Automata

The Institute for Advanced Study , Princeton NJ 08540. 1983.

[62] WOLFRAM Stephen

Computation Theory of Cellular Automata

Communications in Mathematical Physics, 96, pp15-57, 1984

[63] WOLFRAM Stephen

Computer Software in science and mathematics

Scientific American 251(3) : pp188-203, 1984.

[64] WOLFRAM Stephen

Twenty Problems in the Theory of Cellular Automata

Physica Scripta T9, pp170-183, 1985.

[65] WOLFRAM Stephen

A New Kind of Science

Wolfram Media, Inc, 2002.

[66] YUNES Jean-Baptiste

Seven-state solutions to the firing squad synchronization problem

Theoretical Computer Science 127 : pp313-332, 1994.

[67] YUNES Jean-Baptiste

Synchronisation et automates cellulaires : la ligne de fusiliers

Thèse à l'Université Paris 7, L.I.T.P.-TH 93-01 1993

Index

Index