

Complexity in Cellular Automata

Anahí GAJARDO,

DIM - Universidad de Concepción

VI Escuela de Sistemas Complejos de Valparaíso,

7 al 11 de enero de 2008

A *configuration* is an assignation of colors to each cell, i. e., an element of S^R , where R is the rectangle of cells.

At each iteration, the applet computes a function $F : S^R \rightarrow S^R$, producing an infinite sequence of configurations : $c, F(c), F^2(c), F^3(c), \dots$. Such a sequence is called *trajectory* of c .

If some configuration appears twice in this sequence, then the next configurations repeats too.

A trajectory is *periodic* if there is an integer $p > 0$ such that $F^p(c) = c$. If $p = 1$ the trajectory is called a *fixed point*.

A trajectory is *eventually periodic* if there are two integers $t, p > 0$ such that $F^{t+p}(c) = F^t(c)$, p is called the period and t is the transient.

We can investigate the existence of fixed and periodic points, and to characterize them.

Since S^R is a finite set, every sequence is eventually periodic.

Nevertheless, it is not useful to study the system over a rectangle. The rule can be defined over the infinite grid : \mathbb{Z}^2 .

In this case there can exist trajectories which are neither periodic nor eventually periodic.

The Limit Set

A configuration c is in the *limit set* if for every iteration i there exists a configuration c' such that $c = F^i(c')$.

Does a configuration c “converges” “somewhere” ?

- “Somewhere” may be a fixed point, a periodic point or, more generally, a set of configurations.
- The word “converges” have a meaning only if we have a notion of *distance*.

A classical metric is defined as follows. Given two infinite configurations c and c' , then

$$d(c, c') = 2^{-n}, \text{ where } n = \min\{\max\{i, j\} | c(i, j) \neq c'(i, j)\},$$

i. e., n is the distance from $(0,0)$ to the nearest different cell between c and c' .

Sensitivity Classification

The CA rules can be classified into four classes :

- Equicontinuous
- Almost equicontinuous (but not equicontinuous)
- Sensitive (but not expansive)
- Expansive

Informally, a rule is *sensitive* if for every configuration c , and every integer d , there exists a “perturbation” that affect cells at distance d from the “perturbation”, where “perturbation” can be a modification of the state of even an infinite set of cells.

If a rule is not sensitive, there are “patterns” that “resist” any perturbation. But there may be configurations that allow some perturbations to propagate through arbitrarily long distances. These rules are called *almost equicontinuous*.

If perturbations can only affect cells inside fixed radius (from the perturbation), we say that the rule is *equicontinuous*.

Finally, a rule is said to be *Expansive* if every perturbation “diffuses” over every configuration.

Formally, a rule is

expansive if

$$(\exists \delta > 0)(\forall c \in S^{\mathbb{Z}^2})(\forall c' \in S^{\mathbb{Z}^2})(\exists t \in \mathbb{N} \cup \{0\}) d(F^t(c), F^t(c')) > \delta,$$

sensitive if

$$(\exists \delta > 0)(\forall c \in S^{\mathbb{Z}^2})(\forall \epsilon > 0)(\exists c' \in B_\epsilon(c))(\exists t \in \mathbb{N}) d(F^t(c), F^t(c')) > \delta,$$

equicontinuous if

$$(\forall c \in S^{\mathbb{Z}^2})(\forall \epsilon > 0)(\exists \delta > 0)(\forall t \in \mathbb{N})(\forall c' \in B_\delta(c))d(F^t(c), F^t(c')) < \epsilon,$$

almost equicontinuous if the set

$$\{c \mid (\forall t \in \mathbb{N})(\forall \epsilon > 0)(\exists \delta > 0)(\forall c' \in B_\delta(c))d(F^t(c), F^t(c')) < \epsilon\}$$
 is an

intersection of *dense and open* sets.

Does a given perturbation will affect a given configuration ?

If the rule is sensitive, this may be a difficult question.

How the difficulty of a problem can be evaluated ?

A problem is easy if there is a method that solve it "quickly".

Problem : to compute $23(321+45)$.

Method 1

- to *add* 321 to 45
- to *multiply* the result by 23
- the result of the multiplication is the answer

Problem : to compute $23(321+45)$.

Method 1

- to *add* 321 to 45
- to *multiply* the result by 23
- to answer the result of the last operation

Method 2

- to answer 8418

→ this problem is very easy.

Problem : given three integers x , y and z , to compute $x(y + z)$.

Method

- to *add* y to z
- to *multiply* the result by x
- to answer the result of the last operation

→ this problem is less easy.

Problem : given three integers x , y and z , to compute $x(y + z)$.

Method

- to *add* y to z

- to *multiply* the result by x

- to answer the result of the last operation

→ $m = \max\{|y|, |z|\} + 1$

iterations

→ $m|x|$ *iterations*

total : *less than* $|x|(|y| + |z|)$

iterations

→ it is “polynomial”.

Definition. *A problem is said to be polynomial if there is a method that solves it in polynomial time, i. e., a method that takes a number of iterations that is bounded by a polynomial in the size of the inputs.*

The class of polynomial problems is called **P**.

We are interested on the following problem :

(CA-VALUE(f)) Given the configuration of a finite square c of size $2t$ and a given state s , decide whether after $t-1$ steps, s will or will not be the state of the central cell of the square when the initial configuration is c and the CA rule is f .

We are interested on the following problem :

(CA-VALUE(f)) Given the configuration of a finite square c of size $2t$ and a given state s , decide whether after $t-1$ steps, s will or will not be the state of the central cell of the square, when the initial configuration is c and the CA rule is f .

→ it is polynomial.

Within **P** another class is distinguished, informally :

NC : the class of problems that can be solved by a parallel computer, with a polynomial number of processors, in “polylogarithmic” time.

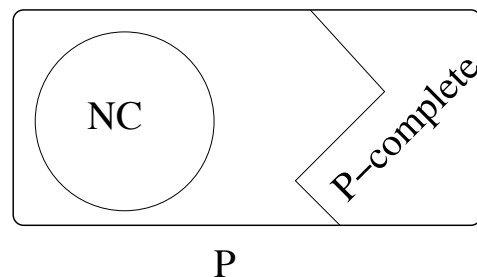
$$\mathbf{NC} \subseteq \mathbf{P}$$

The problem CA-VALUE(Addition) is in **NC**.

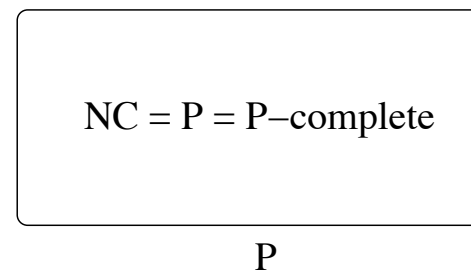
Within the class **P** there is a sub class of problems called **P-complete** problems.

This class is defined in such a way that if one of its problems belongs to **NC** then the whole class **P** is in **NC** (i.e., $P \subseteq NC$).

Then, only two cases are possible :



or



The question is : given a CA rule f , $CA\text{-VALUE}(f) \in \mathbf{NC}$ or it is **P-complete** ?

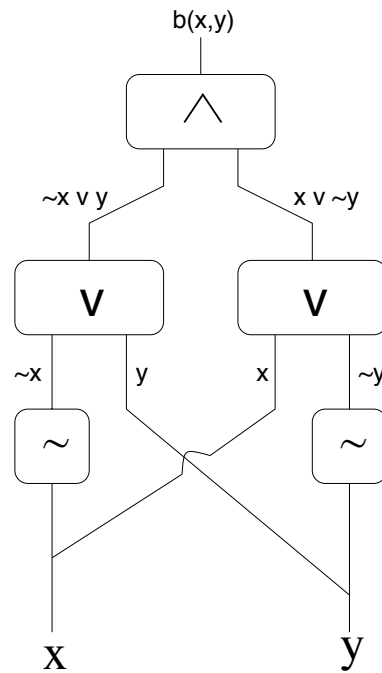
$CA\text{-VALUE}(\text{Game of Life}) \in \mathbf{P-complete}$ (Conway'82).

In fact, if we can solve $CA\text{-VALUE}(\text{Game of Life})$ we can also solve a very known **P-complete** problem :

(CIRCUIT-VALUE) Given a boolean formula, written with connectors AND, NOT and OR, and given the truth values of its variables, decide whether the formula is true or false.

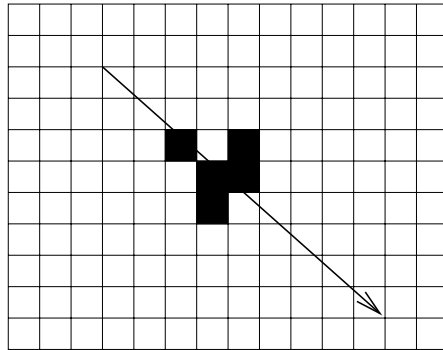
Theorem (Ladner'75) (*CIRCUIT-VALUE*) is ***P-complete***.

For example $b(x, y) = (x \text{ OR } (\text{NOT } y)) \text{ AND } ((\text{NOT } x) \text{ OR } y)$ is a boolean formula.

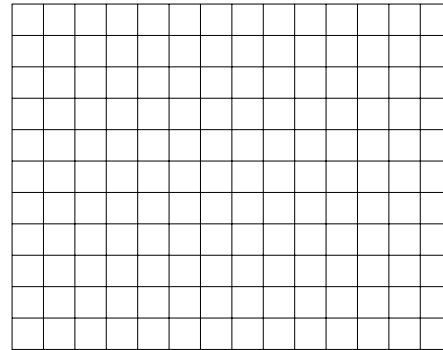


Conway computed circuits with the Game of Life by representing the logical value *True* by the presence of a glider, and the logical value *False* by its absence.

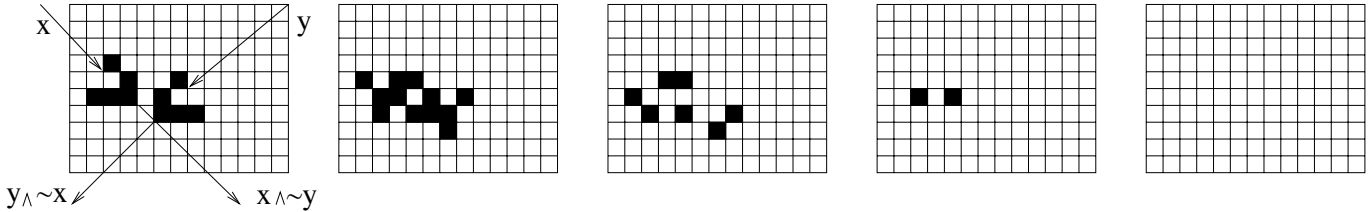
T



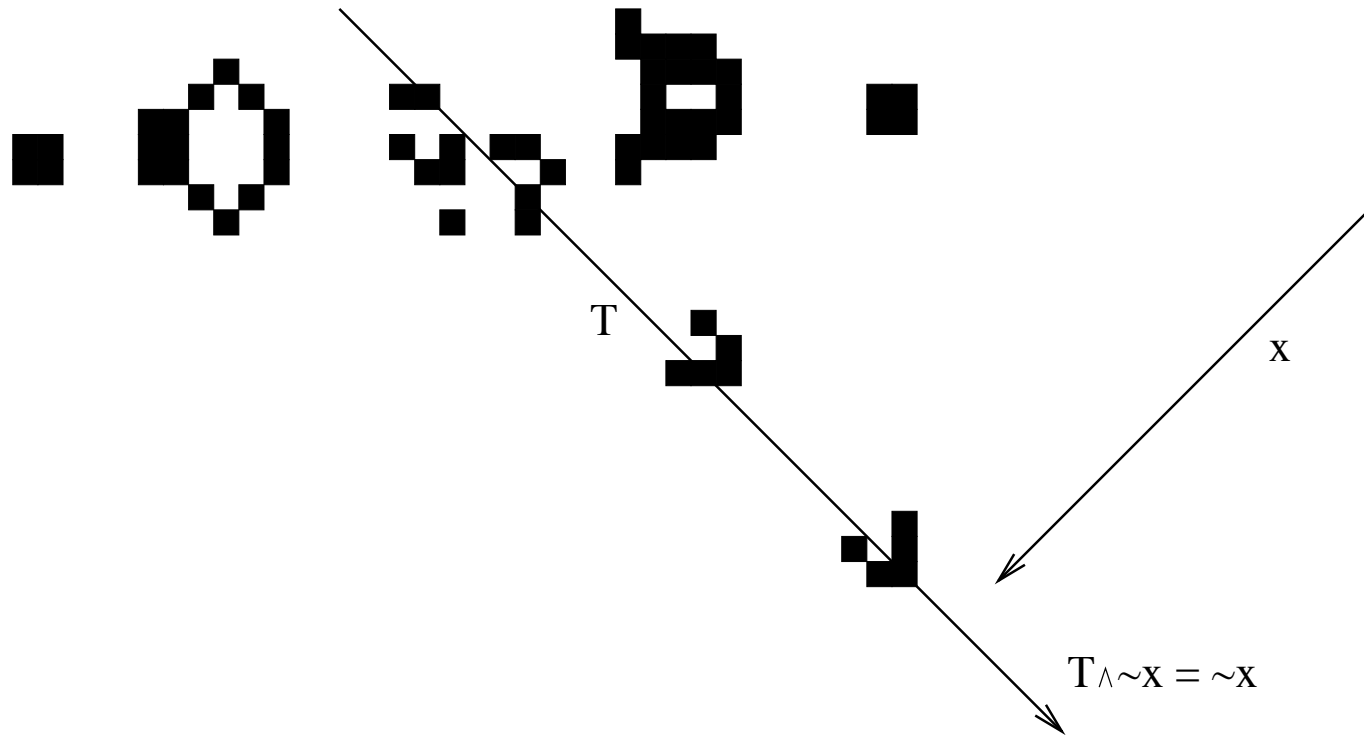
F



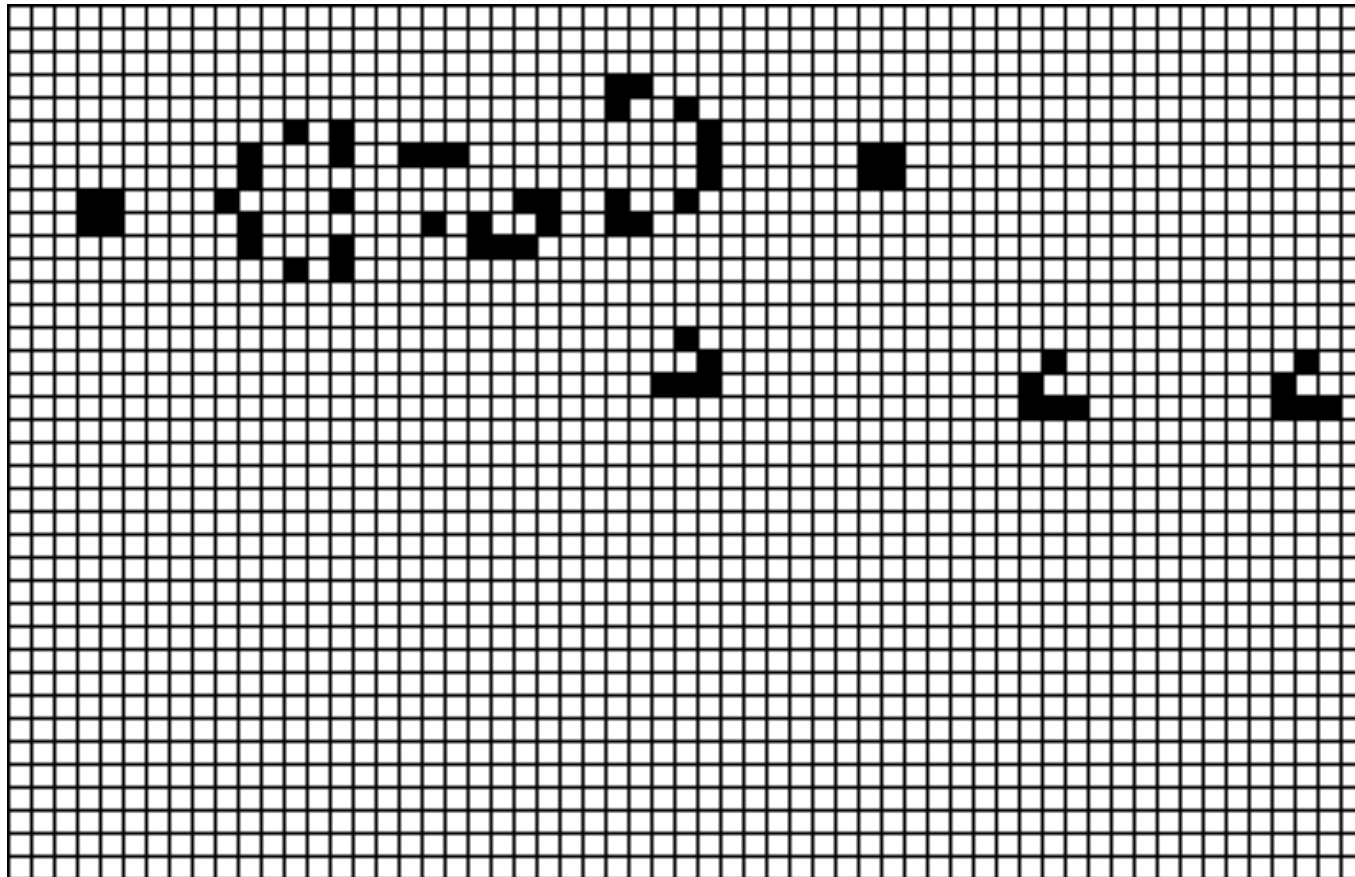
If two gliders meet together, they annihilate each other.
 This give us the logical gate ($x \text{ AND } (\text{NOT } y)$) :



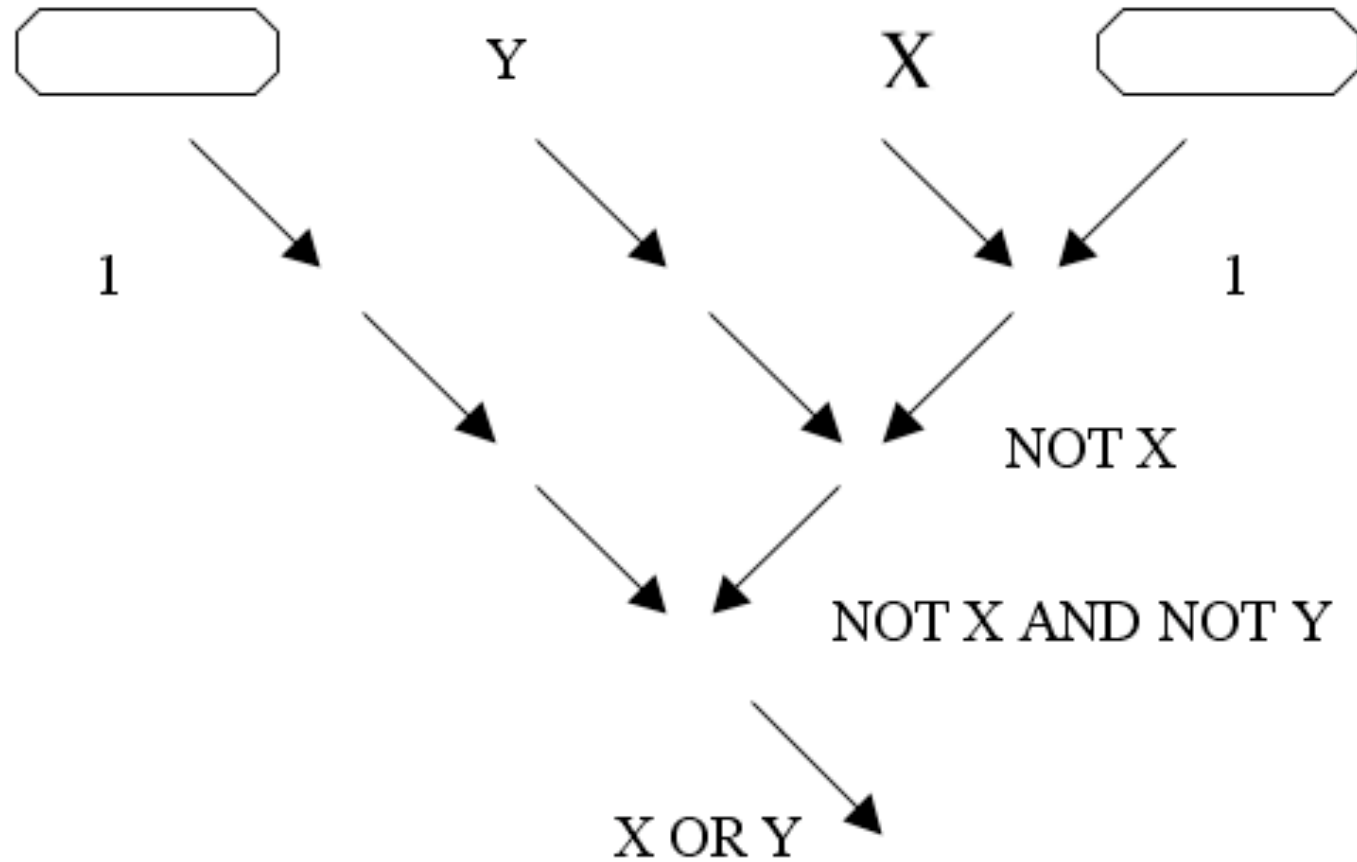
In order to obtain $NOT\ x$ from x we need a glider generator. Such a device produces the constant $True$.



Then we can compute $x \text{ AND } y$:



Then we can compute $x \text{ OR } y$:



Is this enough ?

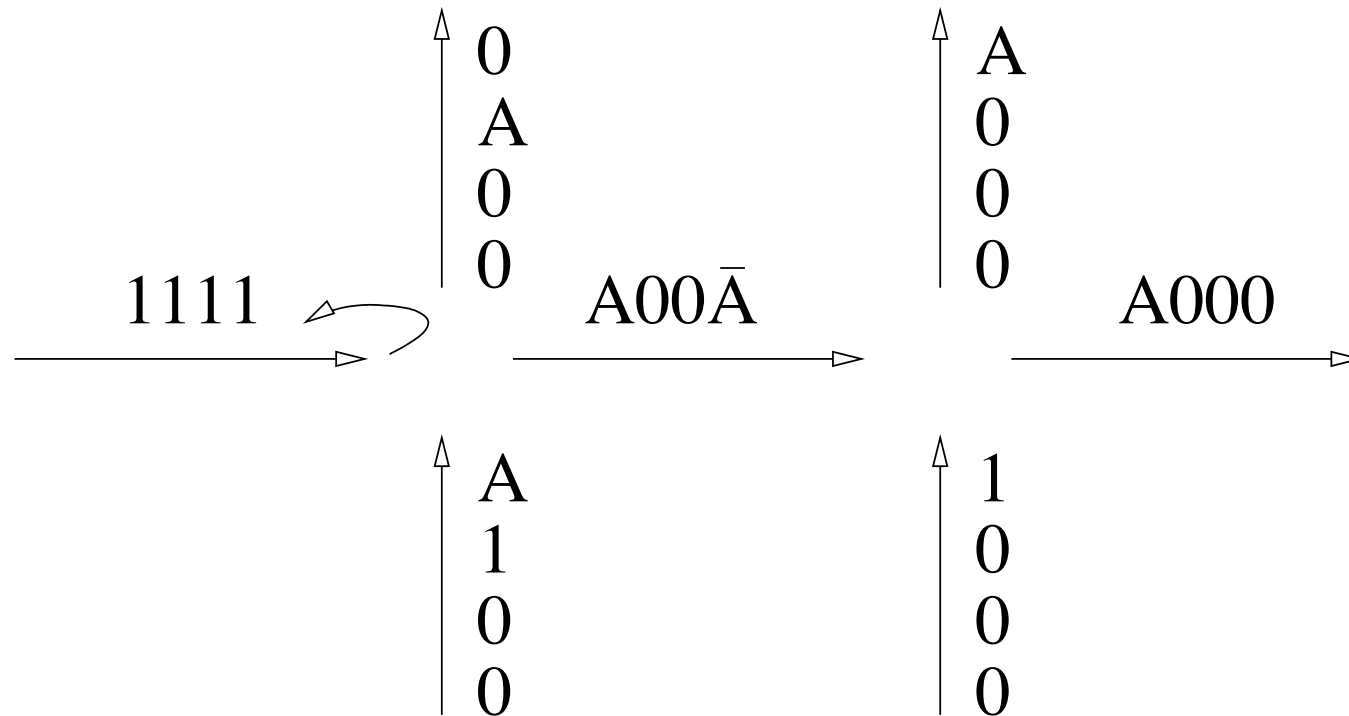
Is this enough ?

No, we need to cross and duplicate signals.

To cross signals is easy, but to duplicate them it is not. In fact there is no way to transform one glider into two gliders.

But we can duplicate the “effect” of one glider : two gliders can collide in several ways, if the relative positions are well choosed one glider can eliminate three gliders from a stream.

The following scheme* shows how to triplicate information :



*this construction was taken from Durand and Róka, in : Cellular Automata : a parallel model, Kluwer Academic Pub, 1999