The Quest for Small Universal Cellular Automata

Nicolas Ollinger*

LIP, École Normale Supérieure de Lyon, 46, allée d'Italie
69 364 Lyon Cedex 07, France

Abstract. We formalize the idea of intrinsically universal cellular automata, which is strictly stronger than classical computational universality. Thanks to this uniform notion, we construct a new one-dimensional universal automaton with von Neumann neighborhood and only 6 states, thus improving the best known lower bound both for computational and intrinsic universality.

1 Why Study Small Universal Machines ?

Designing very small universal machines is an old and fascinating challenge, introduced by Shannon in 1956 [10], which usually involves tricky encodings. This problem has been also explored for other computational machines like Post machines [4]. As an abstract computing model, cellular automata provide the same concerns. Because of its uniformity – there is no separation between control and data – and parallelism, the cellular automata model also provides a kind of intrinsic universality.

An intrinsically universal cellular automaton can simulate, using macro-cells to encode single cells and a linear time slowdown, any given cellular automaton. Understanding how to construct very small automata of this kind involves understanding the way to structure data in both space and time, the way complex computation can be transfered from the local rule to the global one.

In his pioneer study of self-reproduction, Von Neumann [12] introduced a computationally universal (unformally, able to simulate any Turing machine) automaton – in fact, it can be proved intrinsically universal. As far as we know, the question to find very small universal automata was first studied by Smith III [11] and Banks [2] was the first to consider intrinsic universality. Banks closed the problem in dimension 2 and higher. In the case of one-dimensional cellular automata, the problem is more difficult. Until now, there was a gap between the smallest computationally universal automaton and the smallest intrinsically universal one. The different results are reviewed on Table 1.

In the present paper, we fill the gap by exhibiting an intrinsically universal cellular automaton with first-neighbors neighborhood and only 6 states. More material, like simulation programs and big colorfull space-time diagrams, is available on the Web at the url `http://www.ens-lyon.fr/~nollinge/6st/`.

* `Nicolas.Ollinger@ens-lyon.fr`

year	author	d	ν	states	universality
1966	von Neumann [12]	2	5	29	intrinsic
1968	Codd [3]	2	5	8	intrinsic
1970	Banks [2]	**2**	**5**	**2**	**intrinsic**
		1	3	18	intrinsic
1971	Smith III [11]	2	7	7	computation
		1	3	18	computation
1987	Albert and Čulik II [1]	1	3	14	intrinsic
1990	Lindgren and Nordhal [5]	**1**	**3**	**7**	**computation**

here d is the dimension of the cellular automaton and ν its neighborhood size: 5 corresponds to the von Neumann neighborhood and 3 to the first-neighbors neighborhood.

Table 1. Some previously known universal cellular automata

2 Definitions

A *cellular automaton* $\mathcal{A}$ is a quadruple $\left(\mathbb{Z}^d, S, N, \delta\right)$ such that $\mathbb{Z}^d$ is the d-dimensional regular grid, S is a finite set of states, N is a finite set of ν vectors of $\mathbb{Z}^d$ called the neighborhood of $\mathcal{A}$ and δ is the local transition function of $\mathcal{A}$ which maps S^ν to S.

A *configuration* $\mathcal{C}$ of a cellular automaton $\mathcal{A}$ maps $\mathbb{Z}^d$ to the set of states of $\mathcal{A}$. The state of the i-th cell of $\mathcal{C}$ is denoted as $\mathcal{C}_i$. The local transition function δ of $\mathcal{A}$ is naturally extended to a *global transition function* $G_\mathcal{A}$ which maps a configuration $\mathcal{C}$ of $\mathcal{A}$ to a configuration $\mathcal{C}'$ of $\mathcal{A}$ satisfying, for each cell i, the equation $\mathcal{C}'_i = \delta\left(\mathcal{C}_{i+v_1}, \ldots, \mathcal{C}_{i+v_\nu}\right)$, where $\{v_1, \ldots, v_\nu\}$ is the neighborhood of $\mathcal{A}$. A *space-time diagram* of a cellular automaton $\mathcal{A}$ is an infinite sequence of configurations $\left(\mathcal{C}_t\right)_{t\in\mathbb{N}}$ such that, for every time t, $\mathcal{C}_{t+1} = G_\mathcal{A}\left(\mathcal{C}_t\right)$. The usual way to represent space-time diagrams is to draw the sequence of configurations successively, from bottom to top.

A *sub-automaton*[1] of a cellular automaton corresponds to a stable restriction on the states set. A cellular automaton is a sub-automaton of another cellular automaton if (up to a renaming of states) the space-time diagrams of the first one are space-time diagrams of the second one. To compare cellular automata, we introduce a notion of rescaling space-time diagrams. To formalize this idea, we introduce the following notations:

σ^k. Let S be a finite state set and k be a vector of $\mathbb{Z}^d$. The shift σ^k is the bijective map from $S^{\mathbb{Z}^d}$ onto $S^{\mathbb{Z}^d}$ which maps a configuration $\mathcal{C}$ to the configuration $\mathcal{C}'$ such that, for each cell i, the equation $\mathcal{C}'_{i+k} = \mathcal{C}_i$ is satisfied.

o^m. Let S be a finite state set and $m = (m_1, \ldots, m_d)$ be a finite sequence of strictly positive integers. The *packing map* o^m is the bijective map from $S^{\mathbb{Z}^d}$

[1] The prefix *sub* emphasizes the fact that $(S'^{\mathbb{Z}}, G)$ is an (algebraic) sub-structure of $(S^{\mathbb{Z}}, G)$. One could have also used the terminology *divisor* as the set of space-time diagrams of one automaton is included into the one of the other.

onto $\left(S^{m_1 \cdots m_d}\right)^{\mathbb{Z}^d}$ which maps a configuration $\mathcal{C}$ to the configuration $\mathcal{C}'$ such that, for each cell i, the equation $\mathcal{C}'_i = \left(\mathcal{C}_{mi}, \ldots, \mathcal{C}_{m(i+1)-1}\right)$ is satisfied. The principle of $o^{(3,2)}$ is depicted on Fig. 1.

Fig. 1. The way $o^{(3,2)}$ cuts a two-dimensional configuration

Definition 1. *Let $\mathcal{A}$ be a d-dimensional cellular automaton with states set S. A* $\langle m, n, k\rangle$-rescaling *of $\mathcal{A}$ is a cellular automaton $\mathcal{A}^{\langle m,n,k\rangle}$ with states set $S^{m_1 \cdots m_d}$ and global transition function $G_{\mathcal{A}}^{\langle m,n,k\rangle} = \sigma^k \circ o^m \circ G_{\mathcal{A}}^n \circ o^{-m}$.*

Definition 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be two cellular automata. Then $\mathcal{B}$* simulates *$\mathcal{A}$ if there exists a rescaling of $\mathcal{A}$ which is a sub-automaton of a rescaling of $\mathcal{B}$.*

The relation of simulation is a quasi-order on cellular automata. It is a generalization of the order introduced by Mazoyer and Rapaport [6]. In [8], we motivate the introduction of this relation and discuss its main properties. In particular, it induces a maximal equivalence class which exactly corresponds to the set of intrinsically universal cellular automata as described by Banks [2] and Albert and Čulik II [1].

Definition 3. *A cellular automaton $\mathcal{A}$ is* intrinsically universal *if, for each cellular automaton $\mathcal{B}$, there exists a rescaling of $\mathcal{A}$ of which $\mathcal{B}$ is a sub-automaton.*

In the remaining part of this paper, we will especially consider cellular automata of dimension 1. As any cellular automaton can be simulated by a one-way cellular automaton, that is a cellular automaton with neighborhood $\{0, -1\}$, there exist intrinsically universal one-way cellular automata. Therefore, to prove that a particular cellular automaton is intrinsically universal, it is sufficient to prove that it can simulate any one-way cellular automaton.

3 A Simple 8 State Universal Cellular Automaton

Describing in details the behavior of a particular cellular automaton is not an easy task. The following text attempts to give a feeling of the way things work. To verify the correctness of the universality, one only needs the local transition function and the rules to encode a particular cellular automaton. These details will also be given.

We present the automaton in two steps. First, we describe the macroscopic idea behind the simulation. Secondly, we consider the microscopic encoding of those ideas into a cellular automaton.

3.1 Macroscopic Considerations

To simulate a cellular automaton, one has to iterate and compute in parallel a local transition function $\delta : S^N \to S$. Let Ξ be the binary alphabet $\{0, 1\}$ and $n = \lceil \log_2 |S| \rceil$. We can encode S on Ξ^n and decompose δ into n boolean functions $\delta_i : \Xi^{Nn} \to \Xi$. A sample cellular automaton, which will be used to illustrate our simulation, is presented on Fig. 2. As the single boolean operator `NAND`, denoted $|$, is a complete basis[2] for boolean functions computation, we can consider the functions (δ_i) as boolean circuits involving only the `NAND` operator. So, to simulate a cellular automaton, it is sufficient to compute n boolean circuits in parallel in constant time.

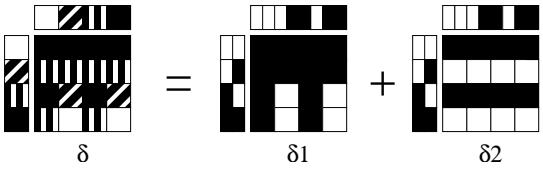


Fig. 2. A sample cellular automaton and its decomposition

Rather than usual circuits, we consider leveled circuits with two kind of gates: copy and `NAND`. The gates of a leveled boolean circuit are partitioned into a finite number of levels $L_0, L_1, \ldots, L_k$, such that every variable is on level L_0, the output is on level L_k and any gate on level L_{i+1} takes its inputs on level L_i. Sample representations of boolean functions by leveled circuits are given on Fig. 3. To simulate cellular automata, we want to simulate such circuits. In order to do so, we must flatten the circuit to encode it on some $S^{\mathbb{Z}}$. We introduce the idea of a boolean circuit cellular automata simulator.

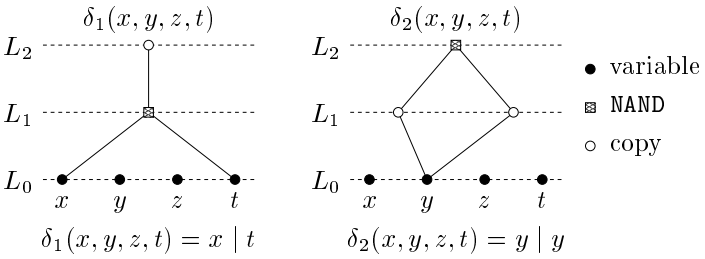
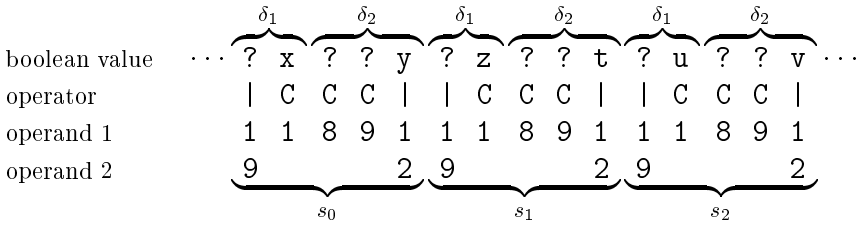


Fig. 3. Boolean functions and their leveled circuit representations

Definition 4. *A boolean circuit cellular automata simulator is a discrete dynamical model. Each cell of the model contains a stored boolean value which is updated at each time step according to some read-only information: an operator (copy or `NAND`) and the relative position of the operands.*

[2] A complete basis B for boolean functions is a set of boolean functions such that any boolean function can be obtained by circuits which doors compute functions from B. For an introduction on boolean functions, see [13].

As the operands can be arbitrarily far away, a boolean circuit cellular automata simulator is not a cellular automaton but it can simulate any cellular automaton, by local simulation of cellular automata cells and local transition function. We simply build a leveled circuit for each δ_i. Each circuit, padded with copy-only levels if necessary, must have the same height T. A cell s_j of the cellular automaton is encoded on the simulator by a macro-cell: a block of cells consisting of concatenation of the circuits of the (δ_i). Variables are extracted from the circuits outputs of the other macro-cells. In T time steps, each macro-cell s_j achieves exactly one transition of the simulated automaton. A sample encoding is depicted in Fig. 4.



In this example, the macro-cells s_j encode the cells of the sample cellular automaton. Only the encoding boolean values are given, `?` correspond to noise. For the operators encoding, `|` corresponds to `NAND` and `C` to copy. One step of the simulation is achieved by 2 time steps of the simulator (one step by level).

Fig. 4. Encoding cells using the boolean circuit cellular automata simulator

When simulating a cellular automaton, the simulator uses only finitely many different kind of cells. So, it accesses operands at a bounded distance m (an upper bound for m is n times the size of the biggest circuit over `NAND` for a boolean function with Nn inputs). To simulate the boolean circuit cellular automata simulator with a cellular automaton, the idea is to encode one circuit cell by a macro-cell consisting of circa m cells. Every circuit cell will send its boolean value to each m neighbors circuit cells on its right. When the i-th information cross the macro-cell, the macro-cell reads the i-th cell of its block and, depending on its contents, stores the value, applies an operand or ignores the value. When m values have crossed a macro-cell, the macro-cell has got its new boolean value and will be able to send it to its neighbors.

3.2 Microscopic Encoding

Background layer. We choose to represent a circuit cell by $m + 2$ cells of our universal automaton, where m is the maximal distance to look at. We use three kinds of states. The first kind, Op state, concerns m cells and memorizes the way to handle incoming information, as described above. One state cell is of the second kind Val and is used to store the boolean value of the macro-cell. The last cell state is of the third kind Sig and transmits a boolean value from macro-cell

to macro-cell. A computation is done each time the three kinds of state meet. The details of the interactions are depicted on Fig. 5.

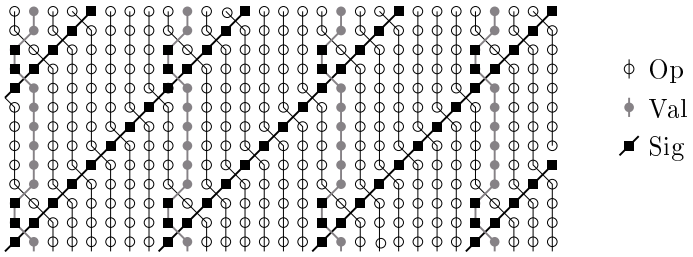


Fig. 5. Three kinds of states to move the information

On the figure, time goes from bottom to top. The Val states are the ones which stay on a same column. The Sig states are the ones which go at real time to the right. The Op states go slowly to the left when crossing Sig cells. We just defined a cellular automaton with von Neumann neighborhood and 3 states. The local transition function is given on Table 2.

s_l	s_m	s_r	$\delta(s_l, s_m, s_r)$	s_l	s_m	s_r	$\delta(s_l, s_m, s_r)$
Op	Op	Op	Op	Sig	Op	Op	Sig
Op	Op	Sig	Op	Sig	Op	Sig	Val
Op	Op	Val	Op	Sig	Op	Val	Sig
Op	Sig	Op	Op	Sig	Sig	Op	Op
Op	Sig	Sig	Sig	Sig	Val	Op	Sig
Op	Sig	Val	Sig	Val	Op	Op	Op
Op	Val	Op	Val	Val	Op	Sig	Op

Table 2. Local transition function of the 3 state automaton

Computational layer. To achieve the construction, we split the three kinds of states into sub-kinds. The Sig kind is split in Sig_0 and Sig_1 boolean signals. The Val kind is also split in Val_0 and Val_1 boolean values. The Op kind is split in $\mathrm{Op}_\sharp$, Op_C, Op_F, and $\mathrm{Op}_|$ to encode respectively, the end of a macro cell, the copy of the incoming signal, the absence of operation, and a `NAND` between incoming and stored values. We obtain an 8 state intrinsically universal cellular automaton, whose transition function is completely described on Table 3.

Encoding of a circuit cell. The exact encoding of a circuit cell, starting from a macro-cell of the boolean circuit cellular automata simulator, is the following:

Let m be the maximum relative position of an operand used in the macro-cell operators (for example $m = 9$ on Fig. 4). The basic shape for cells-encoding is

$$\omega(x, d_1, \ldots, d_{m-1}) = \mathrm{Val}_x\, \mathrm{Op}_{d_1} \cdots \mathrm{Op}_{d_{m-4}}\, \mathrm{Sig}_x\, \mathrm{Op}_{d_{m-3}}\, \mathrm{Op}_{d_{m-2}}\, \mathrm{Op}_{d_{m-1}}\, \mathrm{Op}_\sharp\,.$$

s_l	s_m	s_r	$\delta(s_l, s_m, s_r)$
Op_i	Op_j	Op_k	Op_j
Op_i	Op_j	Sig_v	Op_j
Op_i	Op_j	Val_v	Op_j
Op_i	Sig_v	Op_j	Op_j
Op_i	Sig_v	$\mathrm{Sig}_{v'}$	Sig_v
Op_j	Sig_v	$\mathrm{Val}_{v'}$	$\mathrm{Sig}_{v'}$
Op_i	Val_v	Op_j	Val_v

s_l	s_m	s_r	$\delta(s_l, s_m, s_r)$
Sig_v	Op_i	Op_j	Sig_v
Sig_v	Op_i	$\mathrm{Sig}_{v'}$	$\mathrm{Val}_{\varphi(v,i,v')}$
Sig_v	Op_i	$\mathrm{Val}_{v'}$	Sig_v
Sig_v	$\mathrm{Sig}_{v'}$	Op_i	Op_i
Sig_v	$\mathrm{Val}_{v'}$	Op_i	$\mathrm{Sig}_{\psi(v,v',i)}$
Val_v	Op_i	Op_j	Op_i
Val_v	Op_i	$\mathrm{Sig}_{v'}$	Op_i

$$\text{where} \quad \varphi(v,i,v') = \begin{cases} v' & \text{if } i = C, \\ v \mid v' & \text{if } i = |, \\ v & \text{else,} \end{cases} \quad \text{and} \quad \psi(v,v',i) = \begin{cases} v' & \text{if } i = \sharp, \\ v & \text{else.} \end{cases}$$

Table 3. Local transition function of the 8 state universal automaton

Cells are encoded as instanciations of the operators of these basic shape :

$$\text{copy cell } \overset{\mathtt{C}}{\mathtt{1}}, \qquad \omega^{\mathtt{C}}_{\mathtt{1}}(x) = \omega(x, F, \ldots, F)$$

$$\text{copy cell } \overset{\mathtt{C}}{\mathtt{k}} \text{ with } k > 1, \quad \omega^{\mathtt{C}}_{\mathtt{k}}(x) = \omega(x, \underbrace{F, \ldots, F}_{k-2}, C, F, \ldots, F)$$

$$\texttt{NAND cell } \overset{\mid}{\underset{\mathtt{k}}{\mathtt{1}}}, \qquad \omega^{\mid}_{\underset{\mathtt{k}}{\mathtt{1}}}(x) = \omega(x, \underbrace{F, \ldots, F}_{k-2}, |, F, \ldots, F)$$

$$\texttt{NAND cell } \overset{\mid}{\underset{\mathtt{1}}{\mathtt{k}}} \text{ with } k > 1, \quad \omega^{\mid}_{\underset{\mathtt{1}}{\mathtt{k}}}(x) = \omega(x, \underbrace{F, \ldots, F}_{k-2}, C, \underbrace{F, \ldots, F}_{l-k-1}, |, F, \ldots, F)$$

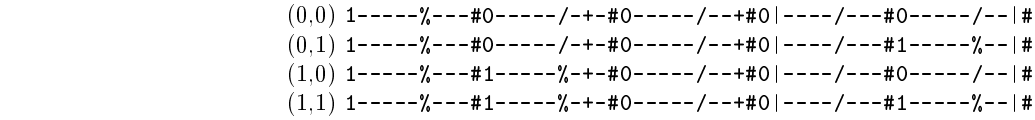
Let $\omega_1, \ldots, \omega_n$ be the sequence of encoding cells corresponding to the macro-cell we want to simulate. Let $v_1, \ldots, v_n$ be the initial boolean values to put into the circuit to encode a particular state. The macro-cell to use on the 8 state cellular automaton is then, shifting to ensure good synchronization:

$$\omega_2(v_1)\omega_3(v_2)\cdots\omega_n(v_{n-1})\omega_1(v_n).$$

One step of the simulation is then achieved in $m(m+2)T$ steps of the simulator where T is the number of levels of the circuit. This transformation was applied to the configuration on Fig. 4 to obtain the configuration on Fig. 6.

4 Tuning the Number of States

In the previous section, an 8 state universal cellular automaton was built. Now, by a careful analysis of the previous automaton, we first show that only 7 of the 8 states are really necessary to achieve universality. Secondly, we briefly explain how the number of Op states can be reduced to 2 sub-kinds, leading to a 6 state universal cellular automaton.



The symbols `-`, `+`, `#`, and `|` respectively encode the operators Op_F, Op_C, $\mathrm{Op}_\sharp$, and $\mathrm{Op}_|$. The symbols `0` and `1` encode the values Val_0 and Val_1. The symbols `/` and `%` encode the signals Sig_0 and Sig_1. One step of the simulation is achieved in 198 time steps of the simulator.

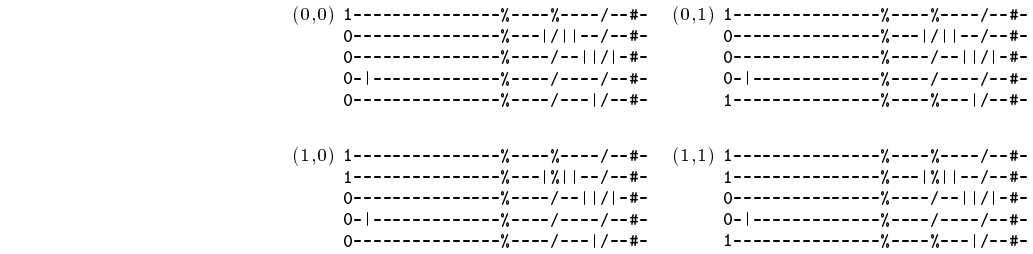
Fig. 6. Encoding of the sample automaton on the 8 state universal automaton

4.1 Emulate copy: from 8 to 7 states

When used with constants, the `NAND` operator acts as follows: $x \mid 0 = 1$ and $x \mid 1 = \neg x$. In the previous simulation, we replace each signal Sig_x by a triple of signals Sig_1, Sig_x, Sig_0. We can then use these constants to emulate Op_C because of the equality

$$1 \mid (x \mid (0 \mid y)) = x \ .$$

The Op_C cells are replaced by the sequence $\mathrm{Op}_|$, $\mathrm{Op}_|$, $\mathrm{Op}_|$ and the three other sub-kinds of Op cells are kept but guarded by an Op_F on the left and on the right to ignore the constant signals. This transformation was applied to the configurations on Fig. 6 to obtain the configurations on Fig. 7. When applying these transformation, the time to simulate one time step on the simulator grows from $m(m+2)T$ to $m(3m+4)T$ time steps of the simulator.



The symbols `-`, `#`, and `|` respectively encode the operators Op_F, $\mathrm{Op}_\sharp$, and $\mathrm{Op}_|$. The symbols `0` and `1` encode the values Val_0 and Val_1. The symbols `/` and `%` encode the signals Sig_0 and Sig_1. One step of the simulation is achieved in 558 time steps of the simulator.

Fig. 7. Encoding of the sample automaton on the 7 state universal automaton

So, we obtain a 7 state intrinsically universal cellular automaton by removing the Op_C state from the previous 8 state intrinsically universal automaton.

4.2 Split the operators: from 7 to 6 states

To obtain a 6 state intrinsically universal cellular automaton, we will now show how to modify the preceding automaton to use only 2 Op sub-kinds.

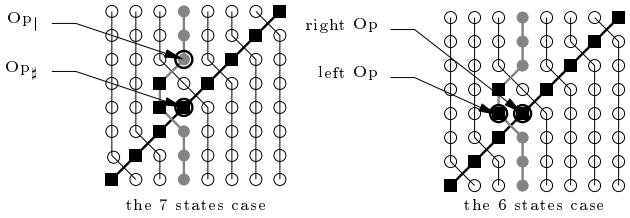


Fig. 8. Location of the computations in the universal cellular automata

The trick is displayed on Fig. 8. In the case of the 7 state automaton, the only active computations concern $\mathrm{Op}_\sharp$ and $\mathrm{Op}_|$ and are applied by looking at the Op on the right of the collision between Val and Sig. We now split the Op in a left Op and a right Op. The two sub-kinds of Op are the boolean values Op_0 and Op_1. The left Op controls the modification of the Val value: if it is equal to Op_1, a NAND between the Sig and the Val is applied and stored in the Val; if it is equal to Op_0, the Val keeps its value. The right Op controls the modification of the Sig value: if it is equal to Op_1, the Val value is stored in the Sig; if it is equal to Op_0, the Sig keeps it value. The 3 old Op are emulated as explained on Table 4.

old Op	left Op	right Op
Op_F	Op_0	Op_0
$\mathrm{Op}_\sharp$	Op_0	Op_1
$\mathrm{Op}_|$	Op_1	Op_0

Table 4. Emulation of the 7 state Op by the 6 state Op

The problem now is that we need to make the Val advance through the Op two by two. To achieve this, we add a new Sig between every two consecutive Sig. The purpose of this garbage signal is not to carry useful information, it only moves the Op.

By doing this, we introduce two new problems. First, we must take care that the garbage signals do not perturb the computation. Secondly, in order to preserve the injectivity of the simulation, we need to clean the information of the garbage signals at the end of every macro-cell transition.

$$\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\ldots\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\#}|_{\#}|_{\#}|_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\ldots\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\left(\mathtt{F}_{\#}|_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\right)\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\mathtt{F}_{\mathtt{F}}\ldots\mathtt{F}_{\mathtt{F}}\#_{|}\mathtt{F}_{\mathtt{F}}$$

Fig. 9. Encoding of a cell with in subscript the operations of the garbage signals

Perturbation of the computation. We first consider the problem of the perturbation of the computation. On Fig. 9 is represented the typical sequence of Op cells for an encoding using the 7 state universal cellular automaton. In subscript are displayed the operations performed by the garbage signals between the useful signals. The only case where the Val value is modified is by applying a

NAND at the end. This will erase the information. But, the time before, the value of the Val has been saved on a Sig signal. In fact, just after the $\mathrm{Op}_{\sharp}$ is executed, the next starting configuration is entirely contained in the sequence of useful signals. We will put everything back in the Val when cleaning the garbage.

Cleaning of the garbage. The cleaning of the garbage is rather technical. The main trick is the partial configuration given on Fig. 10. After going through these configuration, the garbage signal of the Sig_0 and Sig_1 guards contain the value Sig_1, the garbage signal of the Sig_x signal contains $\mathrm{Sig}_{\bar{x}}$ and the value cell on its left contains Val_x. Thus, to avoid synchronization problems, the encoding algorithm goes as follows, starting from the 7 state macro-cell configuration.

signal		0	x	1	0	y	1	
main op		|	F	F	|	|	|	
garbage op	#	F	F	#	#	#	F	
cell value	?	1	1	1	1	$\bar{y}$	y	

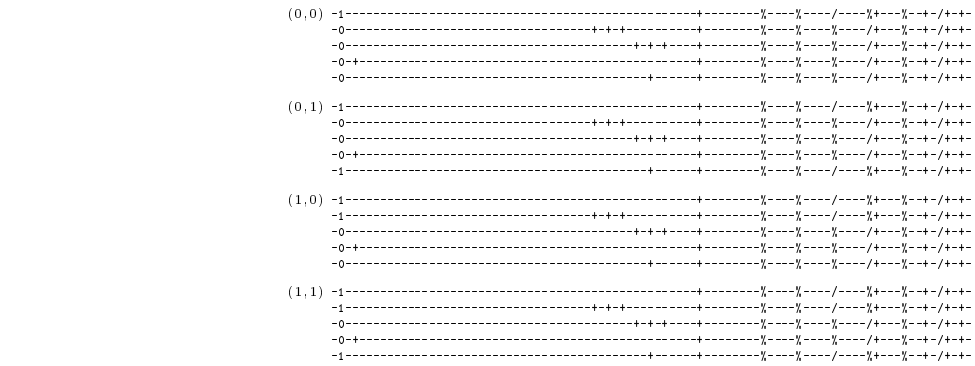
Fig. 10. Sequence of operations to clean the garbage

First, for each subword encoding a cell of the boolean circuit cellular automata simulator, which consists of $3m$ cells of the kind Op, concatenate $n-2$ times the configuration $\mathrm{Op}_F\,\mathrm{Op}_F\,\mathrm{Op}_F$ where n is the size of the boolean circuit cellular automata simulator macro-cell. Then, concatenate the cleaning pattern of Fig. 10, that is $\mathrm{Op}_|\,\mathrm{Op}_F\,\mathrm{Op}_F\,\mathrm{Op}_|\,\mathrm{Op}_|\,\mathrm{Op}_|$. Finally, apply the 7 states to 6 states conversion as described on Table 4. This will let enough space between signals to insert the garbage signals at a distance at least 4 after each coding signal. Finally, the time to simulate one time step on the simulator grows from $m(3m+4)T$ to $(m+n)(6(m+n)+7)T$ time steps of the simulator.

Once again, we apologize for the difficulty to read such statements. We hope it will at least help the interested reader to understand the ideas and path which led to this universal 6 state cellular automaton. For a formal proof, it is sufficient to extract the encoding of the meta-cells from the text and use the local transition function given in Table 5 to simulate another intrinsically universal cellular automaton.

5 The Quest just Begins

Usually the construction of very small computationally universal cellular automata involves very tricky encodings. The main problem is to encode, and be able to differentiate between both the control part of the sequential device and the passive data collection with as few states as possible. Therefore, the universal sequential device which is encoded into a cellular automaton must already be in some sense minimal, as in Lindgren and Nordhal [5] automaton. Those two levels of optimization lead to difficulties to understand how to effectively compute with the obtained cellular automata and the choice of the minimal sequential



The symbols `-` and `+` respectively encode the operators Op_0 and Op_1. The symbols `0` and `1` encode the values Val_0 and Val_1. The symbols `/` and `%` encode the signals Sig_0 and Sig_1. One step of the simulation is achieved in 2548 time steps of the simulator.

Fig. 11. Encoding of the sample automaton on the 6 state universal automaton

s_l	s_m	s_r	$\delta(s_l, s_m, s_r)$
Op_i	Op_j	Op_k	Op_j
Op_i	Op_j	Sig_v	Op_j
Op_i	Op_j	Val_v	Op_j
Op_i	Sig_v	Op_j	Op_j
Op_i	Sig_v	$\mathrm{Sig}_{v'}$	Sig_v
Op_j	Sig_v	$\mathrm{Val}_{v'}$	$\mathrm{Sig}_{\varphi(j,v,v')}$
Op_i	Val_v	Op_j	Val_v

s_l	s_m	s_r	$\delta(s_l, s_m, s_r)$
Sig_v	Op_i	Op_j	Sig_v
Sig_v	Op_i	$\mathrm{Sig}_{v'}$	Val_v
Sig_v	Op_i	$\mathrm{Val}_{v'}$	Sig_v
Sig_v	$\mathrm{Sig}_{v'}$	Op_i	Op_i
Sig_v	$\mathrm{Val}_{v'}$	Op_i	$\mathrm{Sig}_{\psi(v,v',i)}$
Val_v	Op_i	Op_j	Op_i
Val_v	Op_i	$\mathrm{Sig}_{v'}$	Op_i

where $\varphi(j, v, v') = \begin{cases} v' & \text{if } j = 0, \\ v \mid v' & \text{if } j = 1, \end{cases}$ and $\psi(v, v', i) = \begin{cases} v & \text{if } i = 0, \\ v' & \text{if } i = 1. \end{cases}$

Table 5. Local transition function of the 6 state universal automaton

device turns back to problems of non-uniformity like transfer between control and data, between letters and states.

If, as we believe, cellular automata must be considered as an acceptable programming system, that is as a clean computing model, universality investigations must be done inside the model itself and not through doubtful notions of extrinsic simulation. The notion of intrinsic universality try to respond to this necessity. As it is defined on top of a notion of intrinsic simulation which respects both the locality and the uniformity of cellular automata, the notion of intrinsically universal cellular automaton is formal (thus allowing to prove that some cellular automata are not universal which is impossible without a formal definition), uniform and involves full parallelism. This helped us to design an 8 states universal cellular automaton without any tricky encoding. The trick to go from 8 states to 7 states was also very simple. Thus, we were able to do as good as [5] without beginning to really encode. Our tricky encoding lies into the transformation from 7 to 6 states.

The problem to find the smallest intrinsically universal cellular automaton is still open. We think that, using very tricky encodings, it should be possible to transform our automaton into an intrinsically universal automaton with 4 states.

References

1. J. Albert and K. Čulik II, A simple universal cellular automaton and its one-way and totalistic version, *Complex Systems*, **1**(1987), no. 1, 1–16.
2. E. R. Banks, Universality in cellular automata, in *Conference Record of 1970 Eleventh Annual Symposium on Switching and Automata Theory*, pages 194–215, IEEE, 1970.
3. E. Codd, *Cellular Automata*, Academic Press, 1968.
4. M. Kudlek and Y. Rogozhin, New small universal circular Post machines, in R. Freivalds, editor, *Proceedings of FCT'2001*, volume 2138 of *LNCS*, pages 217–226, 2001.
5. K. Lindgren and M. G. Nordahl, Universal computation in simple one-dimensional cellular automata, *Complex Systems*, **4**(1990), 299–318.
6. J. Mazoyer and I. Rapaport, Inducing an order on cellular automata by a grouping operation, *Discrete Appl. Math.*, **218**(1999), 177–196.
7. M. Minsky, *Finite and Infinite Machines*, Prentice Hall, 1967.
8. N. Ollinger, Toward an algorithmic classification of cellular automata dynamics, 2001, LIP RR2001-10, `http://www.ens-lyon.fr/LIP`.
9. N. Ollinger, Two-states bilinear intrinsically universal cellular automata, in R. Freivalds, editor, *Proceedings of FCT'2001*, volume 2138 of *LNCS*, pages 396–399, 2001.
10. C. E. Shannon, A universal Turing machine with two internal states, *Ann. Math. Stud.*, **34**(1956), 157–165.
11. A. R. Smith III, Simple computation-universal cellular spaces, *J. ACM*, **18**(1971), no. 3, 339–353.
12. J. von Neumann, *Theory of Self-reproducing Automata*, University of Illinois Press, Chicago, 1966.
13. I. Wegener, *The complexity of boolean functions*, Wiley-Teubner, 1987.