Cellular Automata, Universality of

Jérôme Durand-Lose[*]

November 7, 2007

Laboratoire d'Informatique Fondamentale d'Orléans,
Université d'Orléans,
B.P. 6759, F-45067 Orléans Cedex 2.

Contents

Glossary

Cellular automata

They are dynamical systems that are continuous, local, parallel, synchronous and space and time uniform. Cellular automata are used to model phenomena where the space can be regularly partitioned and where the same rules are used everywhere, for example: flow dynamics or percolation in physics, systolic arrays in computer science, epidemics in biology...

The configurations are infinite arrays of cells. Each cell has a state chosen inside a finite set. The dynamics is given by replacing the state of each cell according to it and the states of the cells at a bounded distance. Since there are finitely many neighboring cells, there are finitely many state patterns / inputs. The mapping to the new state is

[*] http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose, Jerome.Durand-Lose@univ-orleans.fr

called the local function. The same local function is used for all the cells. They are all updated simultaneously.

Computational universality

Computability is defined by Turing machine, μ-recursive functions or λ-calculus. All these approaches (and many more) end up defining the same set of functions over $\mathbb{N}$ (or on words, *i.e.* finite sequences over a finite alphabet): the computable functions. They defined, according to the Church-Turing thesis, what can be computed by any reasonable device.

A machine is computation universal if it is able to compute any computable function (indicated as a part of the entry). This corresponds also to the common approach of computer, the hardware is universal and the program to be executed is stored in main memory (like the data to process) and is part of the input as far as the hardware / operating system is concerned.

Intrinsic universality

It is the capability to simulate any machine in a class of machine. If one think of Turing machines or an equivalent model of computation, this folds back to the classical computational universality. The interest of this notion lays with machines that are not equivalent to Turing machines. This is the case of cellular automata: they update infinite configurations, there are uncountably many possible configurations thus they cannot be encoded in a countable set, say $\mathbb{N}$.

An intrinsically universal CA "represents" all the CA since it can exhibit any phenomenon any other one can.

1 Definition

Cellular automata (CA) and the subject are briefly defined before two kinds of universality are considered: computational universality and intrinsic universality. A more involving section on advanced topics ends this chapter.

Computational universality deals with the capability to carry out any computation as defined by Turing machines (in computability Theory) while *intrinsic universality* deals with the capability to simulate any other machine of the same class (here cellular automata). This distinction is fundamental here because while computational universality refers to finite inputs and relates to our understanding of computing with computers, intrinsic universality encompasses infinite configurations and relates to our understanding of the physical world. These universalities are presented as simply as possible and an example of universal CA is presented in each case. The last section is devoted to the history and advanced topics such as various definitions of simulation between CA, restriction to reversible CA and different underlying lattices.

2 Introduction

A cellular automaton is a discrete dynamical system composed of regularly displayed cells which satisfies the following properties:

1. *local finiteness*: the set of states available for any cell is finite and there is finitely many cells in any bounded region of space;
2. *locality of computation*: the next state of a cell only depends on the cells around it. There is no global data nor unbounded effect;
3. *uniformity* in both space and time: the dynamics of the cells are identical (space) and never change (time);
4. *parallelism*: all cells are updated at each iteration; and
5. *synchrony*: all cells are iterated at the same instant.

Local finiteness and locality of computation ensure that the next state of any cell can be computed with finite information. This and uniformity ensure that a finite description exists. Parallelism and synchrony ensure that the system is deterministic.

Locality of computation also means that a finite part of a configuration can be isolated in order to see how its central part evolves. It also means that the system is continuous according to the product topology [Hedlund, 1969, Richardson, 1972]. Uniformity also means that the date is not relevant and that if, starting from some local pattern, a phenomenon appends, when the same pattern appears again, the same phenomenon appends, whatever the iteration and location.

Universality. Being universal is somehow to represent all, to be capable to achieve anything possible. This notion is twofold: on the one hand it can be *absolute*, not related to anything in particular, and on the other hand, it can be *relative* to a specific domain. In the case of cellular automata, these two cases are: computational universality and intrinsic universality.

The first one deals with the capability to compute any computable function and relates to simulating Turing machines. Computability Theory and the Church-Turing thesis tell that the set of computable functions does not rely on one specific computing system. On the many textbooks on the subject, [Sipser, 1997, Part 2] is among the best ones for a computer scientist approach.

The second one deals with the capability to simulate any other CA starting from any initial configuration. The distinction is real because since CA do not halt, the notion of the result of a computation is quite meaningless and because CA handle infinite configurations. Even if Turing machines with infinite entries are considered, they only update a limited part of the tape in finite time while CA update the whole (infinite) configuration at each iteration! Since there are computation universal CA in dimension one and simulation is expected to preserve this property, all intrinsically universal CA are also computation universal.

For more information on cellular automata and universality, the reader might be interested in the following surveys and books: [Kari, 2005], [Wolfram, 2002], [Ilachinski, 2001], [Čulik II et al., 1990], [Gutowitz, 1991] and [Toffoli and Margolus, 1987].

Outline of this chapter

Section 3 deals with computational universality and Section 4 deals with intrinsic universality. In each of these sections, definition and results as well as the construction of an example of a universal CA are provided.

Section 5 starts with some history on the subject and then presents advanced and more involving results on existing definitions of simulations between CA, results on the restriction to the reversible CA and on variations on different underlying lattices. Section 6 presents some insight on future researches.

Formal definition of a cellular automaton

A configuration is a made of cells regularly displayed on $\mathbb{Z}^d$; d is called the *dimension*. Each cell is in a state chosen among a finite set of state Q. A *configuration* is then an element of the set:

$$\mathcal{C} = Q^{\mathbb{Z}^d}$$

which is referred to as the *set of configuration*. The computation is local, each cell evolves according to the states of the cells whose coordinates differ by at most r (*radius*) on any coordinate. The local evolution is given by a *local transition function* f that maps the states of the cell and the neighboring ones to the new state of the cell. The *global transition function*, $\mathcal{G}$, maps configurations into configurations; each state is replaced by the image of the states of the cell and the neighboring ones by the local transition function. The subset of $\mathbb{Z}^d$, $\mathcal{N} = [\![-r, r]\!]^d$ is called the *(complete) neighborhood*. It represents the relative positions of neighboring cells. The neighborhood is not necessarily of this form, in fact it can be any finite subset of $\mathbb{Z}^d$. The local and global functions are defined by:

$$\begin{array}{ccc} & f & \\ Q^{\mathcal{N}} & \rightarrow & Q \\ & & \end{array} \qquad \begin{array}{ccl} & \mathcal{G} & \\ \mathcal{C} & \rightarrow & \mathcal{C} \\ c & \mapsto & \mathcal{G}(c) \text{ s.t. } \forall x \in \mathbb{Z}^d,\ \left(\mathcal{G}(c)\right)_x = f\left(c|_{x+\mathcal{N}}\right) \end{array} \tag{1}$$

where $c|_{x+\mathcal{N}}$ denotes the restriction of the configuration c to the positions in $x+\mathcal{N}$.

Definition 1 A *cellular automaton* (CA) is designed by: (d, Q, r, f). When the (finite) neighborhood does not correspond to some $[\![-r, r]\!]^d$, this is emphasized by using $\mathcal{N}$ instead of r. The *space-time diagram*, $\mathcal{D} : \mathbb{Z}^d \times \mathbb{N} \rightarrow Q$, or orbit of a CA is just the infinite sequence of the configurations as the CA is iterated.

Examples of space-time diagrams are provided on figures 2 (left) and 4. For 1-dimensional CA, the space-time diagram can be seen as a tiling of the plane. In dimension 2, each configuration can be considered as a tiling. This approach leads to many undecidability results. It is developed in the Chapter *The Tiling Problem and Undecidability is Cellular Automata.*

Definition 2 If any, the *quiescent state* of a CA, $q_\#$, is a distinguished state such that the uniform configuration $\overline{q_\#}$ is map onto itself (which is equivalent to $f(q_\#, q_\#, \ldots q_\#) = q_\#$). A configuration is *finite* if only finitely many cells are not in the quiescent state.

3 Computational Universality

In this section, cellular automaton are proved to be able to perform any computation in the acceptation of computability Theory and thus that there exist computation universal CA. Only useful concepts and definitions are presented.

The computable functions can be defined by μ-recursion, λ-calculus, Turing machines or any other equivalent model. The Church-Turing thesis asserts that one gets the same functions up to some encoding / representation with any reasonable mean of computation. This is important since, for example, recursive functions address functions over natural integers while Turing machines deal with computations over words (finite sequences over a finite set of symbols).A model of computation is *computation universal* if any computable function can be computed by an instance / machine of the model. A machine is *computation universal* if it can compute any computable function as long as it is provided with the description of the function to compute along with the corresponding input. Computability Theory guaranties that such a universal machine exists.

The typical techniques to prove that a model of computation, here cellular automata, is computation universal are by:

induction like recursion Theory. This would be proving that some functions over integer (*e.g.* $n \mapsto n{+}1$) are computable and then that the functions computable in the model are closed under some operations (*e.g.* composition);

simulation of a generic instance of a computation universal model of computation. This would be to prove that any, say, Turing machine could be simulated by a cellular automata; and

simulation of a computation universal instance of a computation universal model of computation.

These approaches are very different. The first one relies on defining functions but is not concerned by the way they are computed. The second one deals with effective means of computation (allowing the implementation of algorithms and the measure of complexity). The third one is the modern vision of the all-purposes computer: a laptop can do anything from picture manipulation to music playing going through text edition and programming. There is only one hardware and according to the need, one uses a program or another. This is the duality of data and code in modern computers.

Generally, the second case is more simple to tackle. There are mainly two cases when a specific universal machine is transformed:

- when the situation is so complex that using a specific machine with few instructions becomes easier; and
- to get a computation universal CA with specific properties or qualities, typically as few states as possible.

The latter is done by designing a special simulation with a good property and then applying it to a particularly well chosen instance.

The term "computation universal" is not synonymous of "Turing equivalent" which means that whatever computes the model can be computed by, say, a Turing machine. In the case of cellular automata, this is meaningless for two reasons:

- the output is not defined since as a dynamical system, a CA never stops;

- when considering infinite configurations, there is just no way to manipulate them with Turing machines (uncountably many configurations for countably many words).

The first reason leads to a notion of simulation in an not-ending computation rather than an input-output function definition approach. The second reason can be bypassed when considering restrictions of configurations: finite or periodic. Another canonical thing is to consider models of computation able to handle uncountably many different inputs... like cellular automata. This idea leads to intrinsic universality presented in Sect. 4.

3.1 A computation universal cellular automaton

Here Turing machines and their executions are defined and a simulation by cellular automata is provided.

A Turing machine is a very simple device: a finite automaton that can read and write on an unbounded memory (indexed by $\mathbb{N}$). The memory is organized as an infinite sequence of cells called the *tape*. Each cell has a value from a finite set of symbols (the alphabet). Only a finite part of the tape is not empty at any step of the computation. The automaton is equipped with a head that can read or write a single cell of the tape and move the head one position forward of backward on the tape.

Definition 3 A *Turing machine* is defined by $(\Sigma, \mathtt{\#}, Q, q_i, \delta)$, where

- Σ is a finite *alphabet*;
- $\mathtt{\#} \in \Sigma$ is a special symbol use to indicate *empty* part of the tape;
- Q is the *set of states* of the automaton;
- $q_i \in Q$ is the *initial state* and
- $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \rightarrow\}$ is the *transition function*.

The transition function works as follows: in a state q, reading a symbol $\mathtt{a}$, if $\delta(q, \mathtt{a}) = (r, \mathtt{b}, \rightarrow)$ then the new state is r, the symbol $\mathtt{b}$ is written instead of $\mathtt{a}$ and the head moves one step on the right / forward.

Definition 4 A *computation* of a TM starts with the input written on the tape (completed by infinitely many $\mathtt{\#}$) and the automaton in state q_i. The computation goes on as defined by the transition function δ. The computation ends when the head tries to leave the tape on the left. The result is what is written on the tape.

This is not the usual definition which involves an halting state, although it is correct. This formalisation stresses that stopping is somehow an incident from the dynamical system point of view. As far as cellular automata are concerned, they do not stop; an operator might stop a CA when some condition is fulfilled but this is external to the CA.

The Turing machine considered as an example is very simple: starting on a word on $\{\mathtt{a}, \mathtt{b}\}$, it replaces each $\mathtt{a}$ by a $\mathtt{b}$ and *vice-versa*. The only symbols on the tape are $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{\#}$. There are only two states: q_i and r. The transition function is given on the left of Fig. 1.

This Turing machine is simulated by a CA of radius 1 (*i.e.* only cells at distance at most 1 are taken into account for computing the next state of a cell). The set of states of the CA is $\Sigma \cup \Sigma \times Q$, that is, a tape symbol alone or together with a state of the TM. The CA has 9 states, the table of its local function has 729 (=9^3) entries! In the table on the right of Fig. 1 only the cases where the central cell does not change its value

δ	a	b	#
q_i	q_i, b, →	q_i, a, →	r, #, ←
r	r, a, ←	r, b, ←	r, #, ←

x	y	z	$f(x,y,z)$
α	q_i, a	β	b
α	q_i, b	β	a
q_i, a	α	β	q_i, α
q_i, b	α	β	q_i, α
α	q_i, #	β	#
α	β	q_i, #	r, β
α	r, β	γ	β
α	β	r, γ	r, β

$\alpha, \beta, \gamma \in \{$a, b, #$\}$

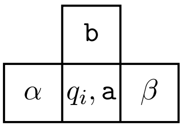
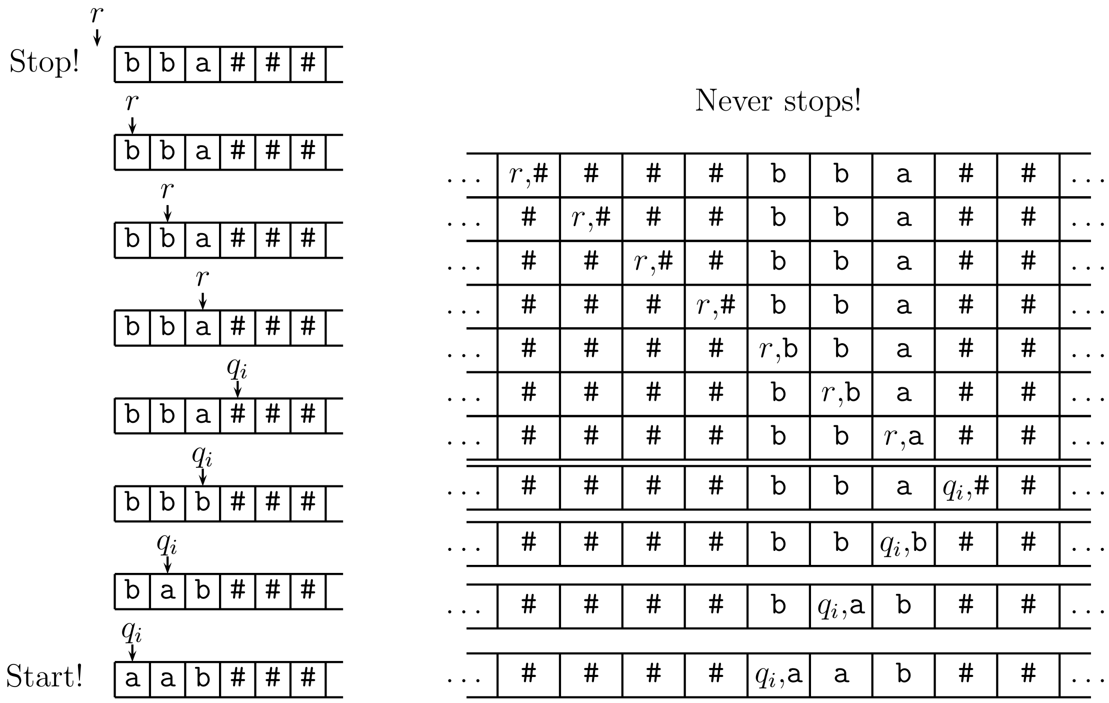


Figure 1: Transition function of a Turing machine and the simulating CA.

are indicated. On the far right, the first transition rule is represented as it appears on space-time diagrams: the bottom line represents the cell and its two closest neighbors and on top the next state of the cell (at the next iteration). In all the space-time diagrams, time is evolving upward.

The dynamics is presented on Figure 2. On the left the whole computation of the Turing machine on the entry aab is given; the output of the function is bba as written on the tape when the machine stops. On the right, the corresponding iterations of the simulating CA are displayed. Since the CA works on a bi-infinite lattice, the configuration is completed with # on the left. As mentioned, a CA never stops so that at some point the computation has been carried out but the system goes on.



...	r,#	#	#	#	b	b	a	#	#	...
...	#	r,#	#	#	b	b	a	#	#	...
...	#	#	r,#	#	b	b	a	#	#	...
...	#	#	#	r,#	b	b	a	#	#	...
...	#	#	#	#	r,b	b	a	#	#	...
...	#	#	#	#	b	r,b	a	#	#	...
...	#	#	#	#	b	b	r,a	#	#	...
...	#	#	#	#	b	b	a	q_i,#	#	...
...	#	#	#	#	b	b	q_i,b	#	#	...
...	#	#	#	#	b	q_i,a	b	#	#	...
...	#	#	#	#	q_i,a	a	b	#	#	...

Figure 2: Iterations of a Turing machine and of the simulating CA.

A “halting” condition has to be provided, especially when one remembers that the halting of a Turing machine (the famous *Halting problem*) is not decidable. Usually, something very simple to test is chosen, for example that some state appears somewhere. Here this would be the first time the closest #-cell on the left is not in state #.

This example can easily be extended to a general method to simulate any Turing machine or specifically a computation universal one. Starting from a computation universal TM, a computation universal CA is generated.

3.2 Other ways to achieve computational universality

Among the various systems achieving computational universality that have been used to prove computational universality of CA, a brief classification can be made.

Machines. (Turing machines, counter automata and random access machines) These systems are very simple, an automaton together with a memory. In general, the whole evolution / orbit of the Turing machine is encoded inside the space-time diagram of the cellular automata like in the example. For counter automata [Minsky, 1967], there are finitely many counters but they can hold any natural integer, any counter can be accessed at any time. Random access machines are like counter automata but with infinitely many register and an indirect access mode which allows to access any register. The random access machines model is the closest to modern computer architecture, but to simulate it, one has to consider indirect addressing and infinitely many register. Counter automata are more simple to simulate: since there exists a computation universal 2-counter automaton, only 6 instructions have to be implemented.

Boolean circuits. The idea is to encode Boolean logic and then to say that any value can be encoded in binary and any function can then be computed with the binary encoding. For example, the transition function of a TM (resp. 2-counter automata) can be encoded and the value of the tape (resp. the counters) stored in an infinite memories. This is generally done by providing a way to encode bits, then logical gates and finally wiring to connect them. It is usually done in dimension 2, since ensuring a correct wire crossing is more complicated in dimension 1. A typical example of this is the computational universality of the Game of Life [Conway, 1970, Berlekamp et al., 1982]. This is developed in the Chapter *Gliders in Cellular Automata*.

Rewriting systems. These systems work on words by removing some sub-word and adding some other sub-word. Starting with a word encoding an entry, the system is expected to stop with the output encoded in the final word. Here are some examples of computation universal systems:

- type-0 grammar in Chomsky's hierarchy: one may replace sub-words by others according to rewriting rules; and
- tag-systems: a prefix of the word is removed and according to it and some rules, a suffix is added.

The proof of the universality of the 2-states 3-neighbors 1-dimension CA referred as the elementary cellular automaton 110 by Cook [Cook, 2004] is done with some tag-system. The construction relies on an intermediate level of simulation that ensures signal transmission and updating. It is too involving and lengthy to be presented here. Signals and particles / solitons ([Steiglitz et al., 1988] [Adamatzky, 2002]) are often used to carry bits / information around.

3.3 Consequences of computational universality

To have computation universality provides two things. On the positive side:

- any computation can be carried out, so that if something has to be computed, whether as a final result or to use in a broader scheme, it can be done;
- since any computation may take place, complex and interesting behaviours may appear;

and on negative side: many questions one may ask about the system become undecidable. The second point comes from the undecidability of the *Halting problem* (whether a given computation of a TM halts). Here are a few examples of this –many more can be found in, for example the Chapter *The Tiling Problem and Undecidability is Cellular Automata*–, given a CA and a finite initial configuration:

- will some state ever appear?
- will the CA ever enter a stable configuration?
- will the configuration grow infinitely?
- will some given configuration be reached?

3.4 "Quality" of a computation universal Cellular Automata

Computability Theory is rather disappointing: one the one side what is computing and computable becomes clear but on the other side it mostly provides negative results like this and that are not computable. In this subsection slight differences on computability / simulation are addressed as well as complexity (of computing) issues.

For Turing machines, the written part of the tape is always finite (it is completed by infinitely many empty state #). The tape is *potentially infinite*, it can be extended as much as needed, it is never infinite. For computing with a CA, it can be assumed that only a finite part of the space is used for computing since otherwise it would take an infinite time for states to interact (it takes $\frac{l}{2r}$ for cells at distance l to interact). There is a definition of finiteness for CA (Def. 2): outside a finite part is it the quiescent state $q_{\#}$ (which plays the same role as #). When an infinite configuration (even it is periodic after some point) is used to achieve computational universality, one talks about *weak computational universality*; more insights on the weakness notion on Turing machines can be found in [Woods and Neary, 2007].

Another important criterion is the one of the efficiency of computation. For example, the simulating a Turing machine by a 2-counter automaton suffers an exponential slowdown so that one may not be interested in such a computing device or any device where computational universality derives from a 2-counter automaton. For example, the proof of the computational universality of rule 110 by Cook [Cook, 2004] has an exponential slowdown but Neary and Wood [Neary and Woods, 2006] proved that in fact that it can be done with a polynomial slowdown. Polynomial slowdown is the classic simulation mode between reasonable models of computation (and this one of the key to the definition –and stability– of the complexity class P –polynomial time solvable problems).

Another thing worth mentioning is that CA are inherently parallel, the universality proofs more or less directly leads to the Turing machines which is the canonical sequential model. Various approaches have been made to consider CA as a computing system on its own, independently of any other model of computation, for example:

- *iterative automaton* where the input is given symbol by symbol to a distinguished cell;
- as word recognizers with only one cell per symbol [Terrier, 1996] (this cannot be computation universal because the memory is bounded);
- if finitely many cells are considered but they are all equipped with a stack, the any computation can be made [Kutrib, 2001];
- a algorithmic on CA based on signals has been developed [Delorme and Mazoyer, 2002, Mazoyer, 1996, Mazoyer and Terrier, 1999]; and
- Martin devised an intrinsically universal CA together $S-m-n$ theorem for CA (as computing devices over infinite configurations) to provide a *acceptable programming system* point of view [Martin, 1994].

This directly links to the notion of universality developed in the next section.

4 Intrinsic Universality

4.1 Definition

Previous section deals with universality as the capability to perform any computation as defined in computability Theory. This theory deals with numbers / words and there exist only countably many configurations. But there are uncountably many configurations for any CA (unless, of course, if there is only one state). It is worth inquiring about another kind of universality that would take this into account and not involve any other model of computation. (It can thus be used for any class of dynamical systems.) It is some kind of *inner-universality*. The idea behind *intrinsic universality* is somehow the counterpart of universality for a Turing machine: being able to simulate any other CA (of the same dimension) on any (infinite) configuration.

Definition 5 A cellular automaton $\mathcal{A}$ *simulates* another CA $\mathcal{B}$ if there is an injective (one-to-one) function, ι, from the configurations of $\mathcal{B}$ to the ones of $\mathcal{A}$ and an integer, τ, such that the following diagram commutes:

$$\begin{tikzcd}
\mathcal{C}_{\mathcal{B}} \arrow[r, "\iota"] \arrow[d, "\mathcal{G}_{\mathcal{B}}"'] & \mathcal{C}_{\mathcal{A}} \arrow[d, "\mathcal{G}_{\mathcal{A}}^{\tau}"] \\
\mathcal{C}_{\mathcal{B}} \arrow[r, "\iota"] & \mathcal{C}_{\mathcal{A}}
\end{tikzcd}$$

A cellular automaton is *intrinsically universal* if it can simulate any other CA (of the same dimension).

The injectivity ensures that $\mathcal{B}$-configurations can be distinguished when mapped into $\mathcal{A}$-configurations. From this definition, it directly comes that:

$$\forall n \in \mathbb{N},\ \iota \circ \mathcal{G}_{\mathcal{B}}^{n} \;=\; \mathcal{G}_{\mathcal{A}}^{n.\tau} \circ \iota \ .$$

In an infinite run, $\mathcal{A}$ generates one iteration of $\mathcal{B}$ every τ iterations. Since the composition of injective functions is injective, the simulation relation is transitive. The definition is illustrated by the construction of an example in the rest of this section.

Other definitions of simulation can be found in Subsection 5.2. In every case, a different intrinsic universality is generated.

4.2 The way it usually works

It is assumed that the simulated CA has radius 1 and is 1-dimensional (how to deal with higher dimension is explained at the end of this section). If it is not the case, it is easy to simulate it by one of radius 1 (and then to use transitivity). Let $\mathcal{A} = (1, Q, r, f)$ be such that the radius is greater than 1. The idea is to group cells r by r and define $\mathcal{B} = (1, Q^r, 1, f_{\mathcal{B}})$ such that $f_{\mathcal{B}}$ is:

$$\begin{aligned} & \forall (c_1, c_2, \ldots c_r), (c_{r+1}, c_{r+2}, \ldots c_{2r}), (c_{2r+1}, c_{2r+2}, \ldots c_{3r}) \in (Q^r)^3 , \\ & \quad f_{\mathcal{B}}\left((c_1, c_2, \ldots c_r), (c_{r+1}, c_{r+2}, \ldots c_{2r}), (c_{2r+1}, c_{2r+2}, \ldots c_{3r})\right) \\ = & \quad \left(f(c_1, c_2, \ldots c_{2r+1}), f(c_2, c_3, \ldots c_{2r+2}), \ldots f(c_r, c_{r+1}, \ldots c_{3r})\right) \ . \end{aligned}$$

The injection ι is the canonical injection where states are grouped r by r:

$$\forall c \in \mathcal{C},\ \forall i \in \mathbb{Z},\ \left(\iota(c)\right)_i \ = \ \left(c_{ir}, c_{ir+1}, \ldots c_{(i+1)r-1}\right) \ .$$

This simulation is just a rescaling of space. From now on, only radius 1 CA are considered. The construction of an intrinsically universal CA uses two scales:

- *meta-cells scale* to manipulate the states and the transition function of the simulated CA; and
- *bit scale* to implement the meta-cells.

A meta-cell gathers copies of the states of its two closest neighbors. Then it has to use the transition function to compute the new state. The transition function is tabulated in the form of a sequence of blocks $(\mathtt{a}, \mathtt{b}, \mathtt{c}, f(\mathtt{a}, \mathtt{b}, \mathtt{c}))$. (It could also have been be represented by, *e.g.*, a Boolean circuit with states binary encoded as in [Ollinger, 2002].)

Meta-cells scale

Each meta-cell holds one block / entry of the transition table. The entries are infinitely repeating on both side and are endlessly shifting on the left so that in one period, the block corresponding to the update eventually appears. When one period has passed a new cycle starts. The architecture of meta-cells is presented on Fig. 3.

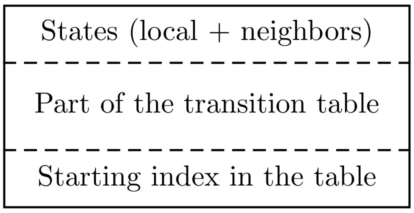
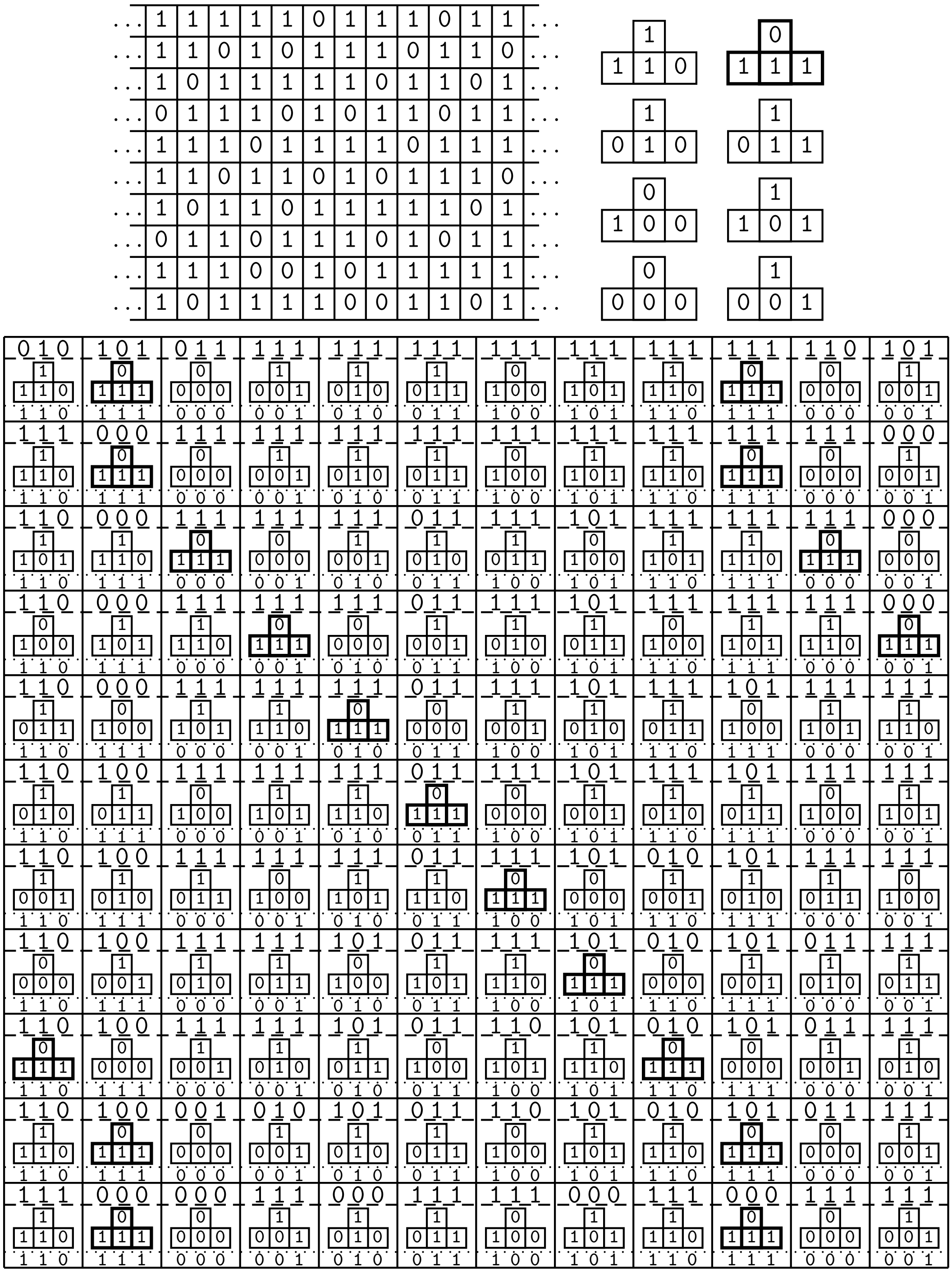

States (local + neighbors)
Part of the transition table
Starting index in the table

Figure 3: Meta-cell.

In the upper part of Figure 4, a space-time diagram generated by some CA is given as well as the transition function of the CA in terms of a relation inside the space-time diagram between three states (at an iteration) and the state above (at the next iteration). In the lower part of Figure 4, the simulation, at the meta-cell scale is presented. Each meta-cell corresponds to the presentation in Fig. 3. The transition table is set inside the initial configuration. The starting index is used by the meta-cell to detect the end of the cycle.



...	1	1	1	1	1	0	1	1	1	0	1	1	...
...	1	1	0	1	0	1	1	1	0	1	1	0	...
...	1	0	1	1	1	1	1	0	1	1	0	1	...
...	0	1	1	1	0	1	0	1	1	0	1	1	...
...	1	1	1	0	1	1	1	1	0	1	1	1	...
...	1	1	0	1	1	0	1	0	1	1	1	0	...
...	1	0	1	1	0	1	1	1	1	1	0	1	...
...	0	1	1	0	1	1	1	0	1	0	1	1	...
...	1	1	1	0	0	1	0	1	1	1	1	1	...
...	1	0	1	1	1	1	0	0	1	1	0	1	...

Figure 4: Iteration of a CA and simulation at meta-cell scale of the first iteration.

As it can be seen on Fig. 4, on the upper part of each cell; at start (lowest row) there are three copies of the state of the simulated cell then, copies are exchanged with the neighbors (first simulating iteration). Then the transition table starts moving. As soon as the entry is found in the table, the three states are replaced by three copies of the new state. The meta-cell then waits for its index to appear again to start this cycle again. The synchrony and uniformity of CA ensures that all meta-cells stay synchronized.

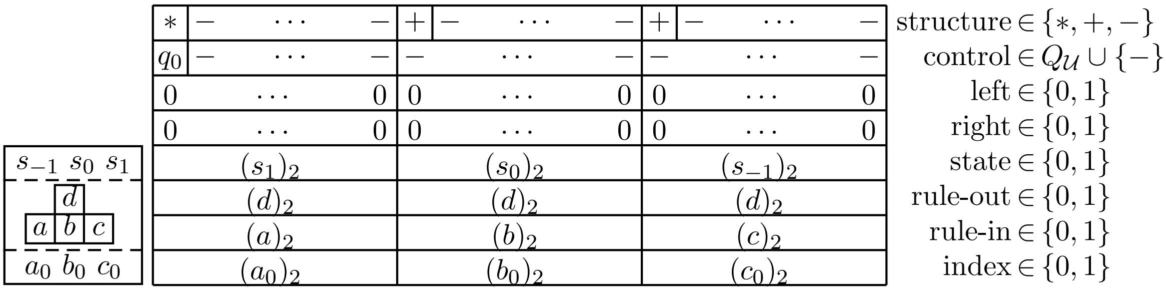
Bits scale

Since the encoding and simulation must work for any CA, it must be able to handle any set of states. The set of states of any CA if finite but unbounded, thus it is impossible to use a common set of states. A state a is thus binary encoded (denoted $(a)_2$). This way if the set of states is larger, then the meta-cell is larger, composed of more elementary cells.

On Figure 5, a meta-cell is given and the layers named. The set of states on each layer for elementary cells are also given. Some layers are added:

- structure: to delimit the different parts of the meta-cell,
- control: to drive the movement, copy and test of bits,
- left and right: to carry bits around (to exchange states and to shift the transition table).

The left and right layers carry the bits very simply: unless noted the new value in left (resp. right) is the one that on the left (resp. right) layer of the right (resp. left) cell ensuring a left (resp. right) shift on the layer. When the simulation starts, s_{-1} and s_1 are equal to s_0 and (a, b, c) is equal to (a_0, b_0, c_0).



$* \mid - \cdots -$	$+ \mid - \cdots -$	$+ \mid - \cdots -$	structure $\in \{*, +, -\}$
$q_0 \mid - \cdots -$	$- \cdots -$	$- \cdots -$	control $\in Q_{\mathcal{U}} \cup \{-\}$
$0 \cdots 0$	$0 \cdots 0$	$0 \cdots 0$	left $\in \{0, 1\}$
$0 \cdots 0$	$0 \cdots 0$	$0 \cdots 0$	right $\in \{0, 1\}$
$(s_1)_2$	$(s_0)_2$	$(s_{-1})_2$	state $\in \{0, 1\}$
$(d)_2$	$(d)_2$	$(d)_2$	rule-out $\in \{0, 1\}$
$(a)_2$	$(b)_2$	$(c)_2$	rule-in $\in \{0, 1\}$
$(a_0)_2$	$(b_0)_2$	$(c_0)_2$	index $\in \{0, 1\}$

Figure 5: Binary encoding of a meta-cell with elementary cells.

The meta-cell implementation works as follows: there is a single value in the control layer. It moves forth and back on the entire meta-cell like a signal. It manages bits from various layers and changes its state accordingly. The construction is only sketched; the states mentioned below are in an intermediate level between meta-cells and elementary cells. The universal CA is not detailed because although this is not complicated it would be quite lengthy and not very informative.

Starting in state q_0, a meta-cell first carries out the states exchange with the meta-cell on the left (the meta-cell on the right takes care of the other exchange) bit by bit using the left and right layers. In state q_1, the transition table is shifted by one entry. In state q_2, it moves through the entire meta-cell and checks whether the state and rule-in layers are identical. If the layers agree, the right transition is found and the state layer can be updated (the bits are copied) then it enters q_3 otherwise it restarts in q_1. In state q_3, the meta-cell has been updated; the transition table is still shifted until a full shift has been done. This is indicated by identical rule-in and index layers. In this the case it enters q_0 and the simulation of a new iteration starts.

Various technical things like delays are added to keep meta-cells synchronized. The number of iterations needed to simulate an iteration is roughly speaking the length of a cell multiplied by the length of the transition table:

$$\log(|Q|).|Q|^3 \ .$$

Higher dimension. It is treated exactly in the same way. One layer gathers the information, another one deals with the transition function. The transition table shifting is done on one dimension only. Extra dimensions can be used to accelerate the process or to design circuitry for a more efficient encoding of the transition function.

Intrinsically universal CA are computation universal by composition (as long as the definitions are robust enough). There is no notion of semi-weakly simulation since the whole configuration has to be used to encode an infinite configuration.

5 Advanced topics

This section gathers a brief history of the subject and various results on specific approaches. It is more involving and targeted to a learned reader.

5.1 A bit of history

Since there is a lot of places where the history of CA is presented (as well as other chapters of this encyclopedia, the reader might be interested in the following survey: [Sarkar, 2000]), only the universality part is developed.

Cellular automaton were introduced in the 50's by Ulam and von Neumann [Ulam, 1952] to study self-reproduction [Neumann, 1966, Arbib, 1966, Burks, 1970]. Computational universality was used just to prove that any pattern can be built. Universality was investigated and proved without any explicit distinction between computational and intrinsic universality before [Banks, 1970].

The most famous CA is certainly Conway's Game of Life [Conway, 1970] from the early 70's. This 2-dimensional CA is both computation universal [Berlekamp et al., 1982] and intrinsically universal [Durand and Róka, 1998].

Quest for small universal CA

There have been an ongoing search for more than four decades for universal CA as small as possible. The interest is to know whether "small" CA are more simple and thus can be handled or there is no gap in complexity. Table 1 sums up results in this quest. Intrinsically universal CA are also computation universal but the converse is not true.

These results were achieved with various constructions. Some use quite different definitions of CA for commodity (but there are indeed CA):

- as partitioned CA: [Morita, 1992b],

Table 1: Historic bounds on computation and intrinsically universal CA.

Year	Reference	Dimension	$\lvert\mathcal{N}\rvert$	$\lvert Q\rvert$	Computation	Intrinsic
1966	[Neumann, 1966]	2	5	29	✗	✗
1968	[Codd, 1968]	2	5	8	✗	✗
1970	[Banks, 1970]	2	9	2	✗	✗
		1	3	18	✗	✗
		1	5	2	✗	✗
1971	[Smith III, 1971]	2	7	7	✗	
1987	[Albert and Čulik II, 1987]	1	3	14	✗	✗
1990	[Lindgren and Nordahl, 1990]	1	3	7	✗	
2002	[Ollinger, 2002]	1	3	6	✗	✗
2004	[Cook, 2004]	1	3	2	✗	
2006	[Richard, 2006]	1	3	4	✗	✗

- as CA with Margolus's neighborhood (or partitioning CA): [Toffoli and Margolus, 1987] and [Margolus, 2002] (billiards) and [Cordero et al., 1992] (spin model in Physics).

Computing universality can also be defined by the computational complexity of the sets of orbits of a CA as well as the reachability relation, relating to the Turing degrees of undecidability [Sutner, 2005].

5.2 Defining simulation among CA (for intrinsic universality)

The definition used in Sect. 4 is not the only existing one. Many papers prove results on simulation without providing a formalised definition of it, but by considering the construction anyone would say that it is a simulation, in an empirical fashion. There is no absolute definition commonly accepted and none contradicts the intuition of simulation. In Table 1, no distinction on the simulation is made and most of the time the construction fits more than one definition.

Most of the presented definitions can be amended in order to cover cases where not all iterations are covered, say for example only the one out of 3. In the following, $\mathcal{A} = (Q_{\mathcal{A}}, r_{\mathcal{A}}, f_{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, r_{\mathcal{B}}, f_{\mathcal{B}})$ denote cellular automata of the same dimension d.

Embedding of Hertling

The term *embedding* is to be understood as simulation. It was introduced in order to prove that CA that are not onto cannot be simulated by onto (and specially reversible) ones of the same dimension.

Definition 6 (Embedding [Hertling, 1998]) A mapping $\mu : Q_{\mathcal{A}}^{\mathbb{Z}^d} \to Q_{\mathcal{A}}^{\mathbb{Z}^d}$ is said to be a *morphism* if and only if for any shift of $\mathbb{Z}^d$, $\sigma_{\mathcal{A}}$, there exists a shift of $\mathbb{Z}^d$, $\sigma_{\mathcal{B}}$, such that: $\mu \circ \sigma_{\mathcal{A}} = \sigma_{\mathcal{B}} \circ \mu$.

$\mathcal{A}$ can be embedded in $\mathcal{B}$ if there are mapping $\mu : Q_{\mathcal{A}}^{\mathbb{Z}^d} \to Q_{\mathcal{A}}^{\mathbb{Z}^d}$ and $\nu : Q_{\mathcal{A}}^{\mathbb{Z}^d} \to Q_{\mathcal{A}}^{\mathbb{Z}^d}$ and an integer k such that:

$$\forall t \in \mathbb{N},\ \mathcal{G}_{\mathcal{A}}^{t} = \nu \circ \mathcal{G}_{\mathcal{B}}^{kt} \circ \mu\ . \tag{2}$$

The embedding is *strong* if μ is a continuous morphism, *weak* if it is a just a morphism and *set-theoretic* otherwise (*i.e.* not a morphism).

The interest of the morphism is to enforce the respect of the structure of $\mathbb{Z}^d$. Hertling proved, using the Axiom of Choice (hence the qualification) that any CA can be set-theoretically embedded in the one dimensional CA that does nothing but shift the value on the left. This is of course highly nonconstructive and contradict the intuition of what simulation could be.

Comparing to Def. 5, using ν to come back allows to have some garbage produced and discarded by ν.

Grouping relation of Mazoyer and Rappaport

This definition of a grouping relation was introduced to focus on the importance of space and time structure. It allows to consider iterations once in a while, periodically.

Definition 7 (Grouping [Mazoyer and Rapaport, 1998, Mazoyer and Rapaport, 1999]) A cellular automaton $\mathcal{A}$ is a *sub-automaton* of $\mathcal{B}$ (denoted $\mathcal{A} \subseteq \mathcal{B}$) if there is an injection $\phi : Q_\mathcal{A} \to Q_\mathcal{B}$ such that:

$$\overline{\phi} \circ \mathcal{G}_\mathcal{A} \;=\; \mathcal{G}_\mathcal{B} \circ \overline{\phi} \ ,$$

where $\overline{\phi}$ is the component-wise extension of ϕ to $\mathcal{A}$-configurations. The nth grouping of an automaton is defined by grouping the cells n by n and consider only every nth iteration, $\mathcal{A}^n = (d, Q^n_\mathcal{A}, r_\mathcal{A}, f^{(n)}_\mathcal{A})$. The function $\mathcal{G}_{\mathcal{A}^n}$ is the nth iterate of $\mathcal{G}_\mathcal{A}$. Since the cells are grouped by n, the radius is not changed. The grouping relation is defined by:

$$\mathcal{A} \leq_{groupping} \mathcal{B} \iff \exists n, m \in \mathbb{N},\ 0 < n, m,\ \mathcal{A}^n \subseteq \mathcal{B}^m \ .$$

This is a stronger form of simulation where the space-time diagrams should be included; up to some rescaling on both side, all the space-time diagrams of $\mathcal{A}$ should be exactly (up to an injection) generated. There is no shift involved and the space-time ratio should be preserved.

This relation is a pre-order. The authors proved that there is a bottom equivalence class for CA: the CA with only one state (and one configuration). The nilpotent ones (after a fixed number of iterations any configuration is turned into the same one: only quiescent state) are just above. They also proved that there is an unbounded infinite ascending chain, so that there is no top and thus no intrinsically universal CA for this definition.

Rescaling of Ollinger

The previous definition is on the one hand interesting because it relays on space-time diagrams and a natural operation, grouping, over them, but on the other hand, there is no intrinsically universal CA, which have been provided for other definition. The following definition is a weakening of the first one that allows intrinsically universal CA.

Definition 8 (Rescaling [Ollinger, 2001]) For any k in $\mathbb{Z}^d$, let σ^k denotes the *shift* by k over configurations. For any m in $\mathbb{Z}^d$, let o^m denotes the *packing* of cells into packs of size m; it is a mapping from $Q^{\mathbb{Z}^d}$ into $(Q^m)^{\mathbb{Z}^d}$ (o^{-m} is the inverse, the unpacking function).

For $n, k \in \mathbb{Z}^d$ and $n \in \mathbb{N}$, $0<n$, the $<m, n, k>$-rescaling of $\mathcal{A}$ is the cellular automaton $\mathcal{A}^{<m,n,k>}$ such that:

$$\mathcal{G}_{\mathcal{A}^{<m,n,k>}} \;=\; \sigma^k \circ o^m \circ \mathcal{G}^n_\mathcal{A} \circ o^{-m} \ .$$

A cellular automaton $\mathcal{A}$ is simulated by $\mathcal{B}$ if there exists a rescaling of $\mathcal{A}$ which is a sub-automaton of a rescaling of $\mathcal{B}$. (The sub-automaton relation is defined as in Def. 7.)

This definition allows to include a shift and to treat independently the size of the blocks of cells and the iteration step. Comparing to the grouping definition (Def. 7), this allows to have enough time to locally mix information and compute the next state. Intrinsically universal CA exist (also with a meta-cell approach) and intrinsic universality of a 1-d CA is undecidable [Ollinger, 2003]. This result is still true on captive CA (the transition function may only output a state that is in the input) even though as the number of states grows larger, almost all captive CA is intrinsically universal [Theyssier, 2004, Theyssier, 2005].

5.3 Reversible case

The reversible subset of cellular automaton (CA such that the global function is invertible, its inverse is then the one of a CA) also contains computation universal CA [Morita and Harao, 1989, Morita, 1992b, Dubacq, 1995]. This topic is developed in the Chapter *Reversible Cellular Automata.*

It also contains intrinsically universal CA among reversible CA [Durand-Lose, 1995, Durand-Lose, 1997, Durand-Lose, 2002]. This means able to simulate any other reversible CA (of the same dimension) and not just any CA. It is a strong embedding (Def. 6) and does not contradict Hertling's results [Hertling, 1998] that non surjective CA cannot be simulated by reversible one.

Morita proved the any CA can be simulated over finite configurations by a reversible in [Morita, 1992a, Morita, 1995] but garbage is produced in order to ensure reversibility and the simulation time varies as the simulation goes on.

There is also a particular result [Durand-Lose, 2000] including the simulation of the non-reversible CA, but the simulation goes by a different definition. It is centered (like the usual topology) and the simulated iterated configurations are placed on parabolas. This twisting of space yields an infinite space to store the information for reversibility. It is not possible to recover a simulated iteration from finitely many simulating ones! The whole simulating space-time diagram is needed.

5.4 Variations on CA

Changing the underlying space

Other 2-dimensional spaces have been considered. There exist reversible computation universal CA both on triangular lattices [Imai and Morita, 1998] and hexagonal lattice [Morita et al., 1998].

Róka studied simulation between CA on different lattices in very general way: lattices are Cayley graph (it corresponds to a group, the arrows corresponds to generators). She proved the existence of simulations in the case of the existence of an homomorphism with a finite kernel and that all bi-dimensional planar structure are equivalent to $\mathbb{Z}^2$ [Róka, 1999].

There exist computation universal CA [Herrmann and Margenstern, 2003] and intrinsically universal CA [Margenstern, 2006] on the hyperbolic plane. This is more developed in the Chapter *Cellular Automata in Hyperbolic Space.*

Intrinsic universality among quantum cellular automaton

In the past decade, quantum computation theory has been tremendously developing. It relies on unitary gates which are of course reversible. The results on reversible CA has been "naturally" extended, for example, there is a 1-dimension Quantum CA which is intrinsically universal (among quantum CA) [Arrighi and Fargetton, 2007]. For more on the topic, please refer to the Chapter *Quantum Cellular Automata.*

Variable neighborhood

A new approach is to fix the states and the transition function and to have only the neighborhood (*i.e.* the relative localisation of the entries of the transition function) varying. Somehow it can be considered that the neighbourhood is not defined by the CA but by the configuration. Some simulation results exists [Nishio, 2007, Worsch and Nishio, 2007]. The transition function of simulated CA is not given inside the simulating configuration but by the simulating neighbourhood.

6 Future Directions

As mentioned just above, understanding the role played by the neighborhood in computing might be very enlightening.

It is known that 2 states and 2 neighbors is enough for computing with polynomial slowdown. It might be interesting to find constructions with limited slowdown and very concise encoding. Since CA are inherently parallel while Turing machines are sequential, a computing (and complexity) theory and algorithm that incorporate the parallelism of CA (local, uniform and synchronous) is worth enquiring.

As far as intrinsic universality is concerned, it relies on simulation between CA. The various definitions have to be linked and investigated.

Books and Reviews

[Adamatzky, 2002] Adamatzky, A., editor (2002). *Collision based computing.* Springer.

[Čulik II et al., 1990] Čulik II, K., Pachl, J., and Yu, S. (1990). Computation theoretic aspects of cellular automata. *Phys. D*, 45(1–3):357–378.

[Gutowitz, 1991] Gutowitz, H., editor (1991). *Cellular Automata, Theory and Experimentation.* MIT/North-Holland.

[Ilachinski, 2001] Ilachinski, A. (2001). *Cellular automata –a discrete universe.* World Scientific.

[Kari, 2005] Kari, J. (2005). Theory of cellular automata: a survey. *Theoret. Comp. Sci.*, 334:3–33.

[Sarkar, 2000] Sarkar, P. (2000). A brief history of cellular automata. *ACM Computing Surveys*, 32(1):80–107.

[Sipser, 1997] Sipser, M. (1997). *Introduction to the Theory of Computation.* PWS Publishing Co., Boston, Massachusetts.

[Toffoli and Margolus, 1987] Toffoli, T. and Margolus, N. (1987). *Cellular Automata Machine - A New Environment for Modeling.* MIT press, Cambridge, MA.

[Wolfram, 2002] Wolfram, S. (2002). *A New kind of Science.* Wolfram Media inc.

Primary Literature

[Albert and Čulik II, 1987] Albert, J. and Čulik II, K. (1987). A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1:1–16.

[Arbib, 1966] Arbib, M. A. (1966). Simple self-reproducing universal automata. *Information and Control*, 9(2):177–189.

[Arrighi and Fargetton, 2007] Arrighi, P. and Fargetton, R. (2007). Intrinsically universal one-dimensional quantum cellular automata. arXiv:0704.3961.

[Banks, 1970] Banks, E. R. (1970). Universality in cellular automata. In *Eleventh Annual Symposium on Switching and Automata Theory*, pages 194–215. IEEE.

[Berlekamp et al., 1982] Berlekamp, E., Conway, J., and Guy, R. (1982). *Winning Ways for your Mathematical Plays (games in particular)*, volume 2. Academic Press.

[Burks, 1970] Burks, A. (1970). *Essays on Cellular Automata.* Univ. of Illinois Press.

[Codd, 1968] Codd, E. (1968). *Cellular Automata.* Academic Press.

[Conway, 1970] Conway, J. (1970). Mathematical games. *Scientific American*, pages 120–123.

[Cook, 2004] Cook, M. (2004). Universality in elementary cellular automata. *Complex Systems*, 15:1–40.

[Cordero et al., 1992] Cordero, P., Goles, E., and Hernández, G. (1992). Q2R + Q2R as a universal billiard. *International Journal of Modern Physics C*, 3,2-2:251–266.

[Delorme and Mazoyer, 2002] Delorme, M. and Mazoyer, J. (2002). Signals on cellular automata. In Adamatzky, A., editor, *Collision-based computing*, pages 234–275. Springer.

[Dubacq, 1995] Dubacq, J.-C. (1995). How to simulate Turing machines by invertible 1d cellular automata. *International Journal of Foundations of Computer Science*, 6(4):395–402.

[Durand and Róka, 1998] Durand, B. and Róka, Zs. (1998). The game of life: universality revisited. In M., D. and J., M., editors, *Cellular Automata: a parallel model*, volume 460 of *Math. Appl.*, pages 51–74. Kluwer, Dordrecht.

[Durand-Lose, 1995] Durand-Lose, J. (1995). Reversible cellular automaton able to simulate any other reversible one using partitioning automata. In *LATIN '95*, number 911 in LNCS, pages 230–244. Springer.

[Durand-Lose, 1997] Durand-Lose, J. (1997). Intrinsic universality of a 1-dimensional reversible cellular automaton. In *STACS '97*, number 1200 in LNCS, pages 439–450. Springer.

[Durand-Lose, 2000] Durand-Lose, J. (2000). Reversible space-time simulation of cellular automata. *Theoret. Comp. Sci.*, 246(1–2):117–129.

[Durand-Lose, 2002] Durand-Lose, J. (2002). Computing inside the billiard ball model. In Adamatzky, A., editor, *Collision-based computing*, pages 135–160. Springer.

[Hedlund, 1969] Hedlund, G. A. (1969). Endomorphism and automorphism of the shift dynamical system. *Math. System Theory*, 3:320–375.

[Herrmann and Margenstern, 2003] Herrmann, F. and Margenstern, M. (2003). A universal cellular automaton in the hyperbolic plane. *Theoret. Comp. Sci.*, 296:327–364.

[Hertling, 1998] Hertling, P. (1998). Embedding cellular automata into reversible ones. In Calude, C., Casti, J., and Dinneen, M., editors, *Unconventional Models of Computation.*

[Imai and Morita, 1998] Imai, K. and Morita, K. (1998). A computation-universal two-dimensional 8-state triangular reversible cellular automaton. In Margenstern, M., editor, *Universal Machines and Computations (UCM 98)*, volume 2, pages 90–99. Université de Metz.

[Kutrib, 2001] Kutrib, M. (2001). Efficient universal pushdown cellular automata and their application to complexity. In Margenstern, M. and Rogozhin, Y., editors, *Machines, Computations, and Universality (MCU '01), Chisinau, Moldavia, May 23-27, 2001*, volume 2055 of *LNCS*, pages 252–263. Springer.

[Lindgren and Nordahl, 1990] Lindgren, K. and Nordahl, M. G. (1990). Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4:299–318.

[Margenstern, 2006] Margenstern, M. (2006). An algorithm for buiding inrinsically universal automata in hyperbolic spaces. In Arabnia, H. R. and Burgin, M., editors, *International Conference on Foundations of Computer Science (FCS '06)*, pages 3–9.

[Margolus, 2002] Margolus, N. (2002). Universal cellular automata based on the collisions of soft spheres. In Adamatzky, A., editor, *Collision-based computing*, pages 107–134. Springer-Verlag.

[Martin, 1994] Martin, B. (1994). A universal cellular automaton in quasi-linear time and its S-n-m form. *Theoret. Comp. Sci.*, 123:199–237.

[Mazoyer, 1996] Mazoyer, J. (1996). Computations on one dimensional cellular automata. *Ann. Math. Artif. Intell.*, 16:285–309.

[Mazoyer and Rapaport, 1998] Mazoyer, J. and Rapaport, I. (1998). Inducing an order on cellular automata by a grouping operation. In *15th Annual Symposium on Theoretical Aspects of Computer Science (STACS '98)*, volume 1373 of *LNCS*, pages 116–127. Springer.

[Mazoyer and Rapaport, 1999] Mazoyer, J. and Rapaport, I. (1999). Inducing an order on cellular automata by a grouping operation. *Discete Appl. Math.*, 218:177–196.

[Mazoyer and Terrier, 1999] Mazoyer, J. and Terrier, V. (1999). Signals in one-dimensional cellular automata. *Theoret. Comp. Sci.*, 217(1):53–80.

[Minsky, 1967] Minsky, M. (1967). *Finite and infinite machines.* Prentice Hall.

[Morita, 1992a] Morita, K. (1992a). Any irreversible cellular automaton can be simulated by a reversible one having the same dimension. *Technical Report of the IEICE, Comp.*, 92-45 (1992-10):55–64.

[Morita, 1992b] Morita, K. (1992b). Computation-universality of one-dimensional one-way reversible cellular automata. *Inform. Process. Lett.*, 42:325–329.

[Morita, 1995] Morita, K. (1995). Reversible simulation of one-dimensional irreversible cellular automata. *Theoret. Comp. Sci.*, 148:157–163.

[Morita and Harao, 1989] Morita, K. and Harao, M. (1989). Computation universality of one-dimensional reversible (injective) cellular automata. *Transactions of the IEICE*, E 72(6):758–762.

[Morita et al., 1998] Morita, K., Margenstern, M., and Imai, K. (1998). Universality of reversible hexagonal cellular automata. In *MFCS'98 Satellite Workshop on Frontiers between Decidability and Undecidability.*

[Neary and Woods, 2006] Neary, T. and Woods, D. (2006). P-completeness of cellular automaton rule 110. In Bugliesi, M., Preneel, B., Sassone, V., and Wegener, I., editors, *International Colloquium on Automata Languages and Programming (ICALP '06)*, number 4051(1) in LNCS, pages 132–143. Springer.

[Neumann, 1966] Neumann, J. v. (1966). *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, Ill. USA.

[Nishio, 2007] Nishio, H. (2007). Changing the neighborhood of cellular automata. In Durand-Lose, J. and Margenstern, M., editors, *Machine, Computations and Universality (MCU '07)*, number 4664 in LNCS, pages 255–266. Springer.

[Ollinger, 2001] Ollinger, N. (2001). Two-states bilinear intrinsically universal cellular automata. In *FCT '01*, number 2138 in LNCS, pages 369–399. Springer.

[Ollinger, 2002] Ollinger, N. (2002). The quest for small universal cellular automata. In *ICALP '02*, number 2380 in LNCS, pages 318–329. Springer.

[Ollinger, 2003] Ollinger, N. (2003). The intrinsic universality problem of one-dimensional cellular automata. In *STACS '03*, number 2607 in LNCS, pages 632–641. Springer.

[Richard, 2006] Richard, G. (2006). A particular universal cellular automaton. oai:hal.ccsd.cnrs.fr:ccsd-00095821_v1.

[Richardson, 1972] Richardson, D. (1972). Tessellations with local transformations. *J. Comput. System Sci.*, 6:373–388.

[Róka, 1999] Róka, Zs. (1999). Simulation between cellular automata on cayley graphs. *Theoret. Comp. Sci.*, 225:81–111.

[Smith III, 1971] Smith III, A. R. (1971). Simple computation-universal cellular spaces. *J. ACM*, 18(3):339–353.

[Steiglitz et al., 1988] Steiglitz, K., Kamal, I., and Watson, A. (1988). Embedding computation in one-dimensional automata by phase coding solitons. *IEEE Transactions on Computers*, 37(2):138–145.

[Sutner, 2005] Sutner, K. (2005). Universality and cellular automata. In Margenstern, M., editor, *Machines, Computations, and Universality (MCU '04)*, number 3354 in LNCS, pages 50–59. Springer.

[Terrier, 1996] Terrier, V. (1996). Language not recognizable in real time by one-way cellular automata. *Theoret. Comp. Sci.*, 156:281–287.

[Theyssier, 2004] Theyssier, G. (2004). Captive cellular automata. In Fiala, J., Koubek, V., and Kratochvíl, J., editors, *Mathematical Foundations of Computer Science (MFCS '04), 29th International Symposium, Prague, Czech Republic, August 22-27*, volume 3153 of *LNCS*, pages 427–438. Springer.

[Theyssier, 2005] Theyssier, G. (2005). How common can be universality for cellular automata? In Diekert, V. and Durand, B., editors, *22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS '05, Stuttgart, Germany, February 24-26*, volume 3404 of *Lecture Notes in Computer Science*, pages 121–132. Springer.

[Ulam, 1952] Ulam, S. (1952). Random processes and transformations. In *International Congress of Mathematics 1950*, number 2, pages 264–275.

[Woods and Neary, 2007] Woods, D. and Neary, T. (2007). Small semi-waekly universal turing machines. In Durand-Lolse, J. and Margenstern, M., editors, *Machines, Computations and Universality (MCA '07)*, number 4664 in LNCS, pages 246–257. Springer.

[Worsch and Nishio, 2007] Worsch, T. and Nishio, H. (2007). Variations on neighborhoods in ca. In Moreno-Diaz, R., editor, *EUROCAST '07*, number 4739 in LNCS, pages 581–588. Springer.