Phd Thesis

Davide Spataro

September 4, 2017

Contents

List of Figures 1

List of Tables 1

1 Cellular Automata **3**

1.1 Introduction 3

1.2 Informal Definition 5

1.2.1 Cellular space dimension and geometry 5

1.2.2 Neighborhood 6

1.2.3 Transition Function 7

1.3 Formal Definition 8

1.3.1 Finite State Automaton 8

1.4 Homogeneous Cellular Automata 11

1.5 Theories and studies 12

1.5.1 Elementary cellular automata 12

Wolfram's code 13

1.5.2 Wolfram's classification 14

1.5.3 At the edge of Chaos 15

1.5.4 Game of life 19

Game of life - brief definition 20

Game of life as Turing machine 21

1.6 Extension of the Cellular automata model 22
1.6.1 Probabilistic CA 22
Cellular automata as Markov process 23

2 Finite Difference Method **25**

2.1 Introduction 25
2.2 Finite difference formulas 26
2.3 Heat Equation 28

3 Tracking Particles **33**

3.0.1 Subsection Heading Here 37
3.1 Conclusion 37

Chapter 1

Cellular Automata

1.1 Introduction

Nowadays most of the basic natural phenomena are well described and known, thanks to the effort of lots of scientist that studied the basic physic's laws for centuries; for example, the freezing of water or the conduction that are well know and qualitative analyzed. Natural system are usually composed by many parts that interact in a complex net of causes/consequences that is at least very difficult but most of the times impossible to track and so to describe. Even if the single component are each very simple, extremely complex behavior emerge naturally due to the cooperative effect of many components. Much has been discovered about the nature of the components in natural systems, but little is known about the interactions that these components have in order to give the overall complexity observed. Classical theoretical investigations of physical system have been based on mathematical models, like differential equations, that use calculus as tool to solve them, which are able to describe and allow to understand the phenomena, in particular for those that can be described by which

are linear differential equation[1] that are easy solvable with calculus. Problems arise when non-linear differential equations come out from the modellation of the phenomena, like fluid turbulence[2]. Classical approach usually fails to handle these kind of equations due to the too high number of components, that make the problem intractable even for a computer based numerical approach. Another approach to describe such systems is to distill only the fundamental and essential mathematical mechanism that yield to the complex behavior and at the same time capture the essence of each component process. Cellular automata are a candidate class of such systems and are well suitable for the modellation and simulation of a wide class of systems, in particular those ones constructed from **many identical** components, each (ideally) simple, but together capable of complex behaviour[33][31]. In literature there are lots applications of cellular automata in a wide rage of class problems from gas[9] and fluid turbulence[28] simulation to macroscopic phenomena[14] like epidemic spread[29], snowflakes and lava flow[5][30]. CA were first investigated by S. Ulam working on growth of crystals using lattice network and at the same time by Von Neumann in order to study self-reproduction[22]; it was not very popular until the 1970 and the famous Conway's game of life[3], then was widely studied on the theoretical viewpoint, computational universality were

Figure 1.1: 3D cellular automata with toroidal cellular space.

[1] Some electro-magnetism phenomena, for instance, can be described by linear differential equations.

[2] Conventionally described by Navier-Strokes differential equation.

proved[3][32] and then mainly utilised, after 1980's, as parallel model due to its intrinsically parallel nature implemented on parallel computers[20].

1.2 Informal Definition

Informally a *cellular automata* (CA) is a mathematical model that consists of a discrete lattice of sites and a value, the state, that is updated in a sequence of discrete timestamps (steps) according to some logical rules that depend on a neighbor sites of the cell. Hence CA describe systems whose the overall behavior and evolution of the system may be exclusively described on the basis of local interactions[38], property also called centrism. The most stringent and typical characteristic of the CA-model is the restriction that the local function does not depend on the time t or the place i: a cellular automaton has homogeneous space/time behavior. It is for this reason that ca are sometimes referred to as *shift-dynamical* or *translation invariant* systems. From another point of view we can say that in each lattice site resides a finite state automaton[4] that take as input only the states of the cells in its neighborhood (see figure 1.4).

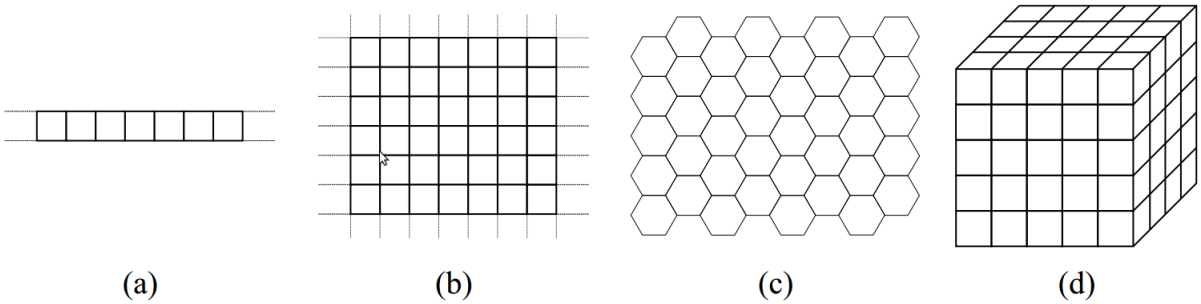
1.2.1 Cellular space dimension and geometry

The cellular space is a *discrete* d-dimensional lattice of sites (see figure 1.2). For 1-D automaton the only way to discretize the space is in a one-dimensional grid. For automaton with dimensionality higher than 1 the shape of each cell can be different than squared. In 2D tessellation for example each cell can be hexagonal or triangular instead of squared. Each tessellation present advantages and disadvantages. For instance the squared one does not give any graphical

[3] CA is capable of simulating a Turing machine, so, in theory is capable of computing every computable problems(Church-Turing thesis). Game of life, for example was proved to be capable of simulating logical gates (with special patterns as *gliders* and *guns*)

[4] A simple and well know computational model. It has inputs, outputs and a finite number of states (hence a finite amount of memory); An automata changes state at regular time-steps.

Figure 1.2: Examples of cellular spaces. (a) 1-D, (b) 2-D squared cells, (c) 2-D hexagonal cells, (d) 3-D cubic cells.



representation problem[5], but present problems of anisotropy for some kind of simulations[6][9]. An Hexagonal tessellation can solve the anisotropy problem[37] but presents obvious graphical issues. Often, to avoid complications due to a boundary, periodic boundary conditions are used, so that a two-dimensional grid is the surface of a torus (see picture 2.3).

1.2.2 Neighborhood

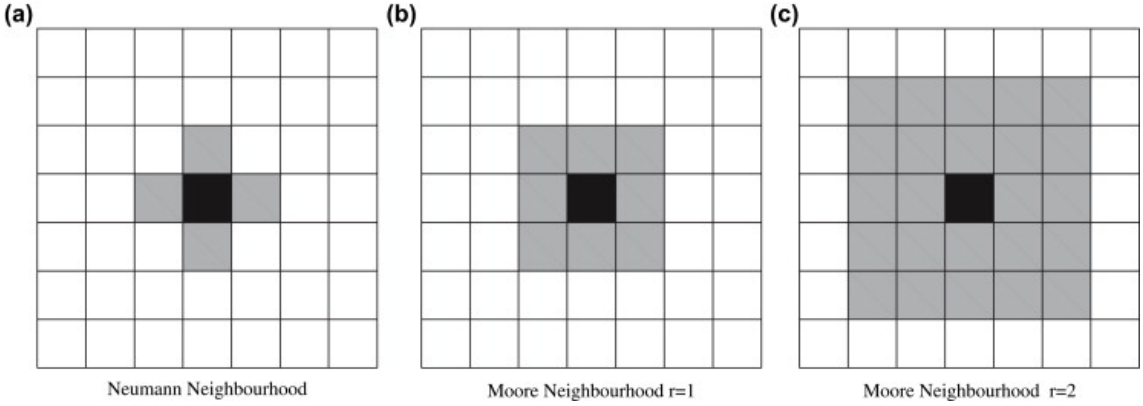
The evolution of a cell's state is function of the states of the neighborhood's cells. The geometry and the number of cells that are part of the neighborhood depends on the tessellation type, but it has to have three fundamental properties:

1. **Locality**. It should involve only a 'limited' number of cells.
2. **Invariance**. It should not be changed during the evolution.
3. **Homogeneity**. It has to be the same for each cell of the automaton.

Typically neighborhood "surrounds" the central cell. For 1-D cellular automata its borders are identified with a number r called *radius*[36]. A $r = 2$ identify $n = 2r + 1$ cells in a 1D lattice: the central cell plus the right and left cells.

[5] Each cell could be easily mapped onto a pixel.
[6] The HPP model for fluid simulation was highly anisotropyc due to the squared tessellation.

Figure 1.3: Examples of different kind of neighborhood with different radius values.



Typical 2D cellular space neighborhood are the those of Moore and von Neumann neighborhood. The number of cells in the Moore neighborhood of range r is the odd squares $(2r+1)^2$, the first few of which are 1, 9, 25, 49, 81, and so on as r is increased. Von Neumann's one consist of the central cell plus the cell at north, south, east, and west of the central cell itself. Moore's ($r = 1$) one add the farther cells at north-east, south-east, south-west and north-west (see figure 1.3).

1.2.3 Transition Function

The evolution of the cell's state is decided by the transition function that is applied at the same time and on each cell. Usually that transition function is deterministic and defined by a *look-up* table only when the total number of state for each cell is small[7] otherwise is defined by an algorithmic procedure. It may be probabilistic, in the case of stochastic cellular automata.

[7]Otherwise the dimension of that table would be enormous because the number of entries is exponential in the number of states.

1.3 Formal Definition

Cellular automata are dynamic models that are discrete in time, space and state. A simple cellular automaton A is defined by a lattice of cells each containing a finite state automaton so we briefly give its definition.

1.3.1 Finite State Automaton

Also known as deterministic finite automata (DFAs) or as deterministic finite state machines, are ones of the most studied and simple computational model known. It is a theoretical model of computation[8] that can be in a finite number of states, only one at a time, the current state. Its state can change in response of inputs taken by a transition function that describe the state change given the current state and the received input of the automata. Its a much more restrictive in its capabilities than a Turing machines,[9] but they are still capable to solve simpler problems, and hence to recognize simpler languages, like well parenthesized string; More in general they are capable to recognize the so called *Regular languages*[10], but they fail for example in parsing *context-free* languages.

δ	a	b	c	d	e
q_0	q_0	q_0	q_2	q_1	q_1
q_1	q_1	q_3	q_1	q_1	q_1
q_2	q_3	q_2	q_2	q_0	q_1
q_3	q_0	q_1	q_1	q_0	q_1

Table 1.1: Tabular representation of a DFM's next-state function

[8] Language recognition problem solvers.

[9] For example we can show that is not possible for an automaton to determine whether the input consist of a prime number of symbols.

[10] Languages defined by regular expressions and generated by regular grammar, Class 3 in Chomsky classification. We can prove that for each language L accepted by a DFA exists a grammar L_G s.t. $L = L_G$

More formally a DFA is a 5-tuple:

$$M =< Q, \Sigma, \delta, q_0, F >$$

- Q is a finite, nonempty, set of states.
- Σ is the alphabet
- $\delta : Q \times \Sigma \longmapsto Q$ is the transition function (also called next-state function, may be represented in tabular form (see table 1.1)
- q_0 is the initial (or starting) state : $q_0 \in Q$
- F is the set, possibly empty, of final states : $F \subseteq Q$

A run of DFA on a input string $u = a_0, a_1, \ldots, a_n$ is a sequence of states $q_0, q_1, \ldots, q_n$ s.t. $q_i \overset{a_i}{\longmapsto} q_{i+1}$, $0 \leq i \leq n$. It means that for each couple of state and input the transition function deterministically return the next DFA's state $q_i = \delta(q_{i-1}, a_i)$. For a given word $w \in \Sigma^*$ the DFA has a unique run (it is deterministic), and we say that it **accepts** w if the last state $q_n \in F$. A DFA recognizes the language L(M) consisting of all accepted strings.

Figure 1.4 is an example of DFA[11]. It accepts the language made up of strings with a number N s.t $N \; mod \; 3 = 0$

- $\Sigma = \{a, b\}$
- $Q = \{t_0, t_1, t_2\}$
- $q_0 = t_0$
- $F = \{t_0\}$

[11] Graph representation is the most common way to define and design DFA. Nodes are the states, and the labelled edges are the possible states transition from a state u to a state w given a certain input. Note that, because the automaton is deterministic is not possible for two edges to point to two different nodes if same labelled.

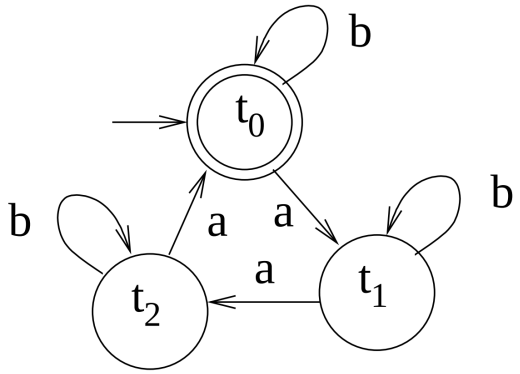


Figure 1.4: Graph representation of a DFA

If we execute the DFA on an input string S={aaabba} we can see that at time t=0 the DFA is in the initial state t_0 and the first symbol of S is read. The transition function is applied once per each symbol is S (i.e. $|S|$). The only rule that match the current state and input is $\delta = (t_0, a) = t_1$ hence the new state is t_1. The DFA accept the string only if there is not any input left and the current state is the final state q_f[12]. S is not accepted by the DFA defined in the example 1.4 because at the end of the computation the reached state is t_1 that is not a final state.

$$t_0 \overset{\delta(t_0,a)}{\longmapsto} t_1 \overset{\delta(t_1,a)}{\longmapsto} t_2 \overset{\delta(t_2,a)}{\longmapsto} t_0 \overset{\delta(t_0,b)}{\longmapsto} t_0 \overset{\delta(t_0,b)}{\longmapsto} t_0 \overset{\delta(t_0,a)}{\longmapsto} t_1$$

On the input $S^1 = \{ababab\}$ the DFA accept:

$$t_0 \overset{\delta(t_0,a)}{\longmapsto} t_1 \overset{\delta(t_1,b)}{\longmapsto} t_1 \overset{\delta(t_1,a)}{\longmapsto} t_2 \overset{\delta(t_2,b)}{\longmapsto} t_2 \overset{\delta(t_2,a)}{\longmapsto} t_0 \overset{\delta(t_0,b)}{\longmapsto} t_0 \overset{\delta(t_0,b)}{\longmapsto} \mathbf{t_0}$$

[12]Previously we stated that F was a set but we can assume that there is only one final state ($|F| = 1$), because it is easy prove that exist a DFA with only one final state given a generic DFA ($|F| \geq 1$). We add one more state q_f and for each final state $q_i \in F$ we define new rules of the type $\delta(q_i, *) = q_f, * \in I$.

1.4 Homogeneous Cellular Automata

Formally a CA A is a quadruple $A =< Z^d, X, Q, \sigma >$ where:

- $\mathbb{Z}^d = \{i = (i_1, i_1, \ldots, i_d) \mid i_k \in \mathbb{Z}, \forall k = 1, 2, \ldots, d\}$ is the set of cells of the d-dimensional Euclidean space.

- X is the neighborhood, or neighborhood template; a set of m d-dimensional vectors (one for each neighbor)

$$\xi_j = \{\xi_{j1}, \xi_{j2}, \ldots, \xi_{jd}\} \ , \ 1 \leq j \leq m$$

  that defines the set of the neighbors cells of a generic cell $i = (i_1, i_1, \ldots, i_d)$

$$N(X, i) = \{i + \xi_0, i + \xi_2, \ldots, i + \xi_d\}$$

  where ξ_0 is the null vector. It means that the cell i is always in its neighborhood and we refer to it cell as *central cell* (see example below).

- Q is the finite set of states of the elementary automaton EA.

- $\sigma = Q^m \to Q$ is the transition function of the EA. σ must specify $q_k \in Q$ as successor state of the central cell. If there are m cells in the neighborhood of the central cell including itself, then there are $|Q|^m$ possible neighborhood's state configuration. It means that there are $|Q|^{|Q|^m}$ possible transition functions. Plus we can see that the tabular definition of the next-state function is unsuitable for practical purpose. It should have $|\sigma| = |Q|^m$ entries, an exceedingly large number.

- $\tau = C \longrightarrow C \longmapsto \sigma(c(N(X, i)))$ where $C = *cc$: $Z^d \to Q$ is called the set of the possible configuration and $C(N(X, i)))$ is the set of states of the neighborhood of i.

For example consider a 2D cellular automata with Moore neighborhood and a generic cell c=(10,10) and $|Q| = 5$ possible state for each cell .

$$X = \{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7, \xi_8\} =$$

$$= \{(0,0), (-1,0), (0,-1), (1,0), (0,1), (-1,-1), (1,-1), (1,1), (-1,1)\}$$

Hence the set of the cells belonging to the neighborhood(defined by X) of c=(10,10) is: $V(X,c) = \{(0,0)+c, (-1,0)+c, (0,-1)+c, (1,0)+c, (0,1)+c, (-1,-1)+c, (1,-1)+c, (1,1)+c, (-1,1)+c\}$

$$= \{(10,10), (9,10), (10,9), (11,10), (10,11), (9,9), (11,9), (11,11), (9,11)\}$$

and the total number of entries for the tabular definition of the transition-function is $|Q|^{|X|} = 5^9 = 1953125$ and the total number of possible transition functions is $|Q|^{|Q|^{|X|}} = 5^{5^9} = 5^{1953125}$.

1.5 Theories and studies

1.5.1 Elementary cellular automata

The most simple AC we can imagine are the elementary cellular automata[36]. They are one-dimensional periodic N cells array $\{C_i \mid 1 \leq i \leq N, C_i \in \{0,1\}\}$ each with 2 possible state (0,1), and rules that depend only on nearest neighbor value hence a radius r=1 neighborhood with a total number of involved cell $2r + 1 = 2 \times 1 + 1 = 3$ (central, right and left cells). Since there are only $2 \times 2 \times 2\times = 2^{2r+1} = 2^3 = 8$ possible states for the neighborhood of a given cell there are a total of $2^{2^3} = 2^8 = 256$ possible elementary automata (each of which may be identified with a 8-bit binary number[39]).

Table 1.2: Encoding of a transition function for a generic elementary CA. On the right the instance 110.

$F(1,1,1) = \{0,1\}$		$F(1,1,1) = 0$
$F(1,1,0) = \{0,1\}$		$F(1,1,0) = 1$
$F(1,0,1) = \{0,1\}$		$F(1,0,1) = 1$
$F(1,0,0) = \{0,1\}$	$\overset{instance}{\longrightarrow}$	$F(1,0,0) = 0$
$F(0,1,1) = \{0,1\}$		$F(0,1,1) = 1$
$F(0,1,0) = \{0,1\}$		$F(0,1,0) = 1$
$F(0,0,1) = \{0,1\}$		$F(0,0,1) = 1$
$F(0,0,0) = \{0,1\}$		$F(0,0,0) = 0$

Wolfram's code

The transition function is $F(C_{i-1}, C_i, C_{i+1})$ is defined by a look-up table of the form stated in table 1.2, and an example of an instance of a function is given (rule 110, an important rule on which [4] proved universal computational power, as Wolfram had conjectured in 1985, and is arguably the simplest Turing complete system[39]) in table 1.2.

More generally Wolfram's code[36, 39] can be calculated Conventionally neighborhoods are sorted in non-decreasing order ,(111=7),(110=6),(101=5) etc., and the may be interpreted as a 8-digit number

$$01101110 = 2^0 \times 0 + 2^1 \times 1 + 2^2 \times 1 + 2^{\times} 1 + 2^4 \times 0 + 2^5 \times 1 + 2^6 \times 1 + 2^7 \times 0 = 110$$

1. List and sort in decreasing numerical (if interpreted as number) order all the possible configuration of the neighborhood of a given cell.

2. For each configuration, list the state which the given cell will have, according to this rule, on the next iteration.

3. Interprets the resulting list as binary number and convert it to decimal. That is the Wolfram's code.

Note that is not possible to understand from that code which is the size or

the shape of the neighborhood. Is tacit to suppose that those information are already known.

1.5.2 Wolfram's classification

Mathematical analysis of CA may be not so straightforward despite their simple definition. A first attempt to classify CA was attempted by Wolfram[39]. He proposed a set of four classes for CA classification that are the most popular method of CA classification, but they suffer from a degree of subjectivity. Classification is based only on visual valuations, that are obviously subjective. A more rigorous definition of these classes is given in [13][16]. Here the four Wolfram's classes.

1. these CA have the simplest behavior; almost all initial conditions result in the same uniform initial state (homogeneous state).

2. different initial conditions yield different final patterns, but these different patterns consist of an arrangement of a certain set of structures, which stays the same forever or repeats itself within a few steps(periodic structures).

3. behavior is more complicated and appears random, but some repeated patterns are usually present (often in the form of triangles)(chaotic pattern).

4. in some respects these are the most complicated class; these behave in a manner somewhere in between Class II and III, exhibiting sections both of predictable patterns and of randomness in their pattern formation(complex structures).

[13]They prove that decide the class(from the wolfram's four one) of membership of a generic CA is an undecidable problem. Is not possible to design an algorithm that solve this problem.

He observed that the behavior of a meaningful class of Cellular Automata by performing computer simulations of the evolution of the automata starting from random configurations. Wolfram suggested that the different behavior of automata in his classes seems to be related to the presence of different types of attractors.

In figures 1.5 and 1.6 some elementary automata divided in their classes.[14]

We can well see from these examples that automata from class 1 have all cells ending up very quickly with the same value, in a homogeneous state and automata from class 2 with a simple final periodic patterns. Class 3 appear to be chaotic and non-periodic and automata from class 4 have a mixed behaviour, complex-chaotic structures are locally propagated.

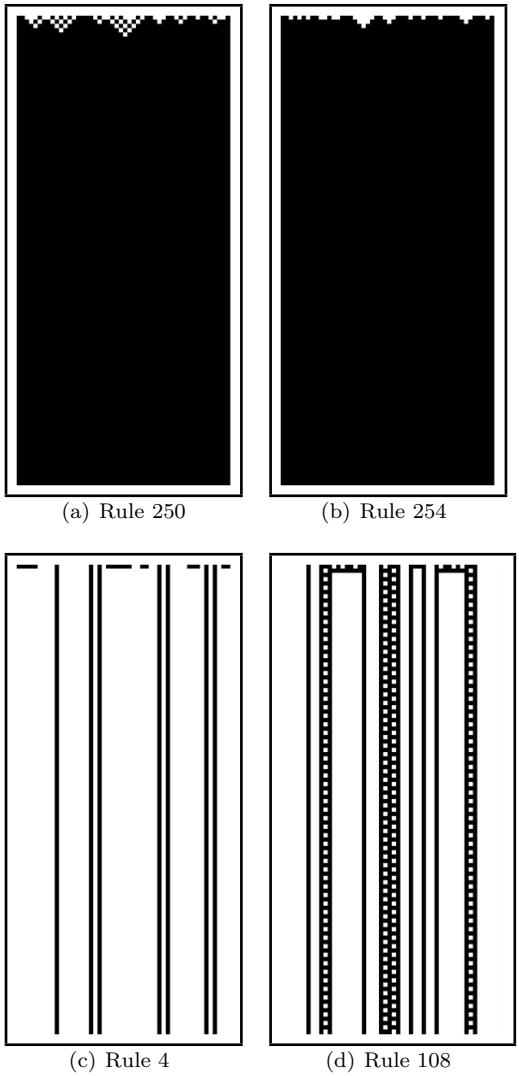
1.5.3 At the edge of Chaos

Class 4 automata are at *the edge of chaos* and give a good metaphor for the idea that the *interesting* complexity (like the one exhibit by biological entities and their interactions or analogous to the phase transition between solid and fluid state of the matter, is in equilibrium between stability and chaos[19].

> *Perhaps the most exciting implication (of CA representation of biological phenomena) is the possibility that life had its origin in the vicinity of a phase transition and that evolution reflects the process by which life has gained local control over a successively greater number of environmental parameters affecting its ability to maintain itself at a critical balance point between order and chaos.*
> *(**Chris Langton** - Computation at the edge of chaos. Phase transition and emergent computation - pag.13).*

[14] Images courtesy of http://plato.stanford.edu/entries/cellular-automata/

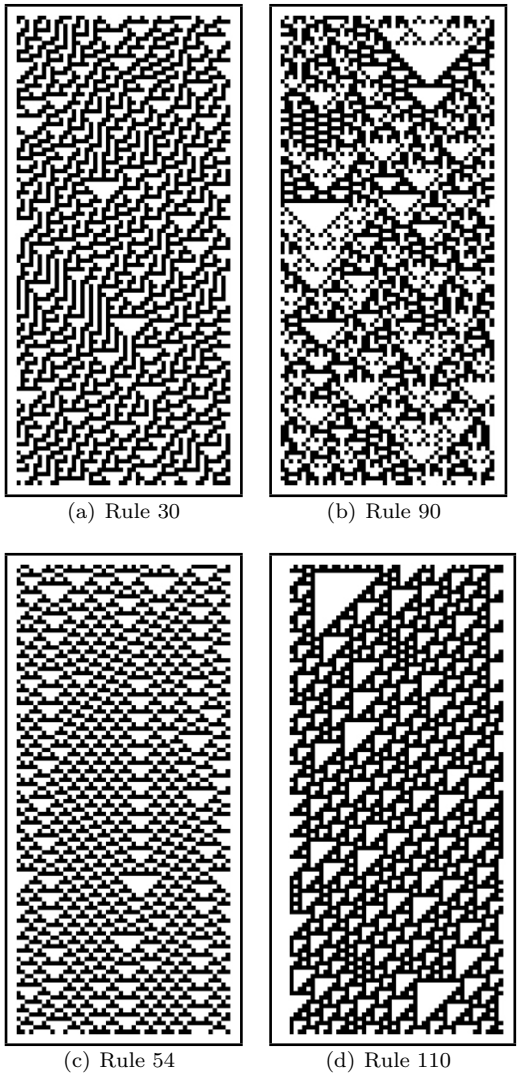
Figure 1.5: Class 1 (a,b) and 2 (c,d) elementary cellular automata



(a) Rule 250 (b) Rule 254

(c) Rule 4 (d) Rule 108

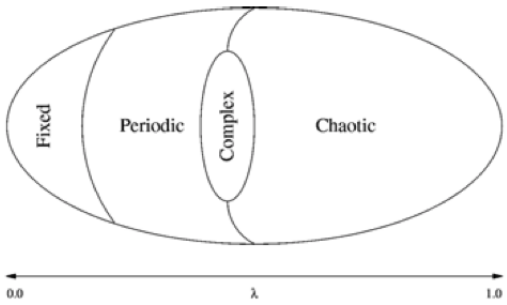
Figure 1.6: Class 3 (a,b) and 4 (c,d) elementary cellular automata



(a) Rule 30 (b) Rule 90

(c) Rule 54 (d) Rule 110

Figure 1.7: Relation between lambda parameter and the CA behaviors-Wolfram's classes.



Langton in his famous paper, *Computation at the edge of chaos: phase transition and emergent computation*[19],was able to identify, simply parametrizing the rule space, the various AC classes, the relation between them and to "couple" them with the classical complexity classes. He introduced the parameter λ[18]that, informally, is simply the fraction of the entries in the transition rule table that are mapped the not-quiescent state.

$$\lambda = \frac{K^N - n_q}{K^N}$$

where:

- K is the number of the cell states
- N the arity of the neighborhood
- n_q the number of rules mapped to the quiescent state q_q

Langton's major finding was that a simple measure such as it correlates with the system behavior: as it goes from 0 to $1 - \frac{1}{K}$(respectively the most homogeneous and the most heterogeneous rules table scenario), the average behavior of the system goes from freezing to periodic patterns to chaos and functions with an average value of λ (see [19] for a more general discussion) are being on *on the edge*(see figure 1.7).

He studied a entire family of totalistic CA with $k = 4$ and $N = 5$ with λ varying in $[0, 0.75]$. He was able to determine that values of $\lambda \approx 0.45$ raise up to class 4 cellular automata. Computational system must to provide fundamental properties if it is to support computation. Only CA *on the edge* show these properties on manipulating and store information data. Here the properties that a computational system as to provide:

Storage

Storage is the ability of the system of preserving information for arbitrarily long times

Transmission

Transmission is the propagation of the information in the form of signals over arbitrarily long distance

Modification

Stored and transmitted information is the mutual possible modification of two signals.

Storage is coupled with less entropy of the system, but transmission and modification are not. Few entropy is associated with CA of Class 1 and 2 and high entropy with class 3. Class 4 is something in between, the cells cooperate and are correlate each other, but not too much otherwise they would be overly dependent with one mimicking the other supporting computation in all its aspects and requirements. Moreover class 4 CA are very dependent from the initial configuration opening to the possibility to encode programs in it.

1.5.4 Game of life

CA are suitable for representing many physical, biological, social and other human phenomena. But they are a good tool to study under which condition

a physical system supports the basic operation constituting the capacity to support computation. Game of life is a famous 2D cellular automaton of '70s early studied (and perhaps proved) for its universal computation capacity.

Game of life - brief definition

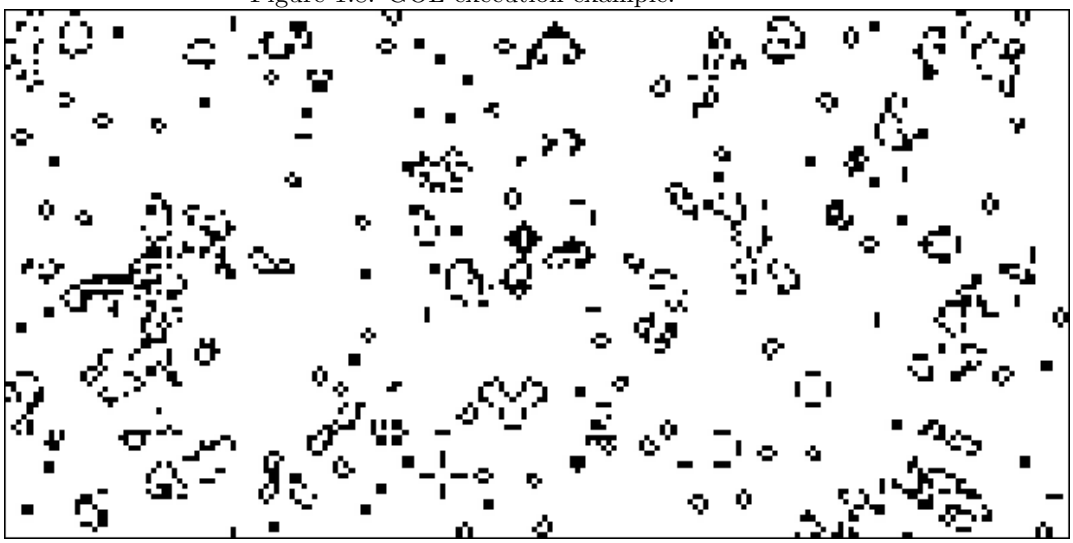
Game of Life (see figure 1.8) (GOL)[3] is a totalistic CA[15] defined by :

- a 2-D lattice of square cells in an orthogonal grid, ideally infinite
- $Q = \{0, 1\}$ 2 states, and we can picture 1 as meaning alive and 0 dead (those interpretation come from the behaviour of the next-state function).
- X is the Moore neighborhood template.
- σ is the transition function and can be summarized :
  - *Birth*: If the cell is in the state ***dead*** and the number of alive neighbors is ***3***, then the cell state becomes alive (1).
  - *Survival*: If the cell is in the state ***alive*** and the number of alive neighbors is ***2 or 3***, then the cell state is still alive (1).
  - *Dead*: If the cell is in the state ***alive*** and the number of alive neighbors is ***less than 2 or higher than 3***, then the cell state becomes dead (0).

GOL is a class 4 Wolfram's taxonomy, rich complex structures, stable blocks and moving patterns come into existence even starting from a completely random configuration.

[15] A totalistic cellular automaton is a cellular automata in which the rules depend only on the total (or equivalently, the average) of the values of the cells in a neighborhood.

Figure 1.8: GOL execution example.



A famous block for example is the *glider* (see picture 1.9) that is a 5-step-period pattern that is capable of moving into the cellular space.

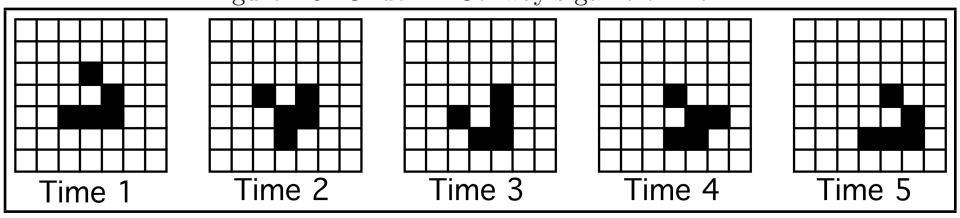
Game of life as Turing machine

Every CA con be considered a device capable of supporting computation and the initial configuration can encode an input string (a program for example). At some point the current configuration can be interpreted as the result of the computation and decoded in a output string. But as we stated before in section 1.3.1 not all the computational device have the same computational power. So which is the one of the game of life? Life was proved can compute everything a universal Turing machine can, and under Turing-Church's thesis, everything can be computed by a computer[1].

This raises a computational issue; given the *Halting Theorem*[16] the evolution of *Life* is unpredictable (as all the universal computational systems) so it

[16]There can not be any algorithm to decide whether, given an input, a Turing machine will accept or not.

Figure 1.9: Glider in Conway's game of life.



means that is not possible to use any algorithmically shortcuts to anticipate the resulting configuration given an initial input. The most efficient way is to let the system run.

> *Life, like all computationally universal systems, defines the most efficient simulation of its own behavior[15]*

1.6 Extension of the Cellular automata model

It is possible to relax some of the assumptions in the general characterization of CA provided in the ordinary CA definitions and get interesting results. Asynchronous updating of the cell, non homogenous lattice with different neighborhood or transition functions.

1.6.1 Probabilistic CA

Probabilist CA is are an extension of the common CA paradigm. They share all the basic concept of an ordinary homogeneous CA with an important difference in the transition function. σ is a stochastic-function that choose the next-state according to some probability distributions. They are used in a wide class of problems like in modelling ferromagnetism, statistical mechanics [10] or the cellular Potts model[17]

[17]Is a computational lattice-based model to simulate the collective behavior of cellular structures.

Cellular automata as Markov process

Another approach in studying CA, even if it is probably not a practical way to study the CA is to see CA as a Markov process[18]. A Markov process, is a stochastic process that exhibits memorylessness [19] and it means that the future state is conditionally independent[20] of the past. This property of the process means that future probabilities of an event may be determined from the probabilities of events at the current time. More formally if a process has this property following equation holds:

$$\begin{aligned} P(X(t_n) = x \,|X\,(t_1) = x_1, X(t_2) = x_2, \ldots, X(t_{n-1}) = x_{n-1}) \\ = P(X(t_n) = x|X(t_{n-1} = x_{n-1}) \end{aligned}$$

In PCA analysis we are more interested in Markov chain because each cell has a discrete set of possible value for the status variable. In terms of chain a CA is a process that starts in one of these states and moves successively from one state to another. If the chain is currently in state s_i, than it evolve to state s_j at the next step with probability p_{ij}.The changes of state of the system are called transitions, and the probabilities associated with various state changes are called transition probabilities usually represented in the Markov chain transition matrix :

$$M = \begin{pmatrix} p_{11} & p_{12} & p_{13} & \cdots \\ p_{21} & p_{12} & p_{23} & \cdots \\ p_{31} & p_{32} & p_{33} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

This could seems to be a good way to analyze the probabilistic CA but, a

[18]Name for the Russian mathematician Andrey Markov best known for his work on stochastic processes.

[19]Also called Markov property.

[20]Two event A and B are independent if $P(AB) = P(A)P(B)$ or in other words that the conditional probability $P(A|B) = P(A)$.

small grid 10×10 (common models model use grid 100×100 or larger) identify $2^{10\times10}$possible states and the resulting matrix dimension of $2^{10\times10} \times 2^{10\times10}$ that is a very large number!

Chapter 2

Finite Difference Method

2.1 Introduction

The finite difference approximation for derivativatives is one of the simplest oldest method to solve differential equations numerically. It was used since 1768 by *L. Euler* to solve one dimensional problem and extended by *C. Runge* to two dimension in 1910 ca. Since the advent of computers in 1950 ca. FDM popularity skyrocketed and during the last five decades theoretical results have been obtained regarging stability, convergence and other of its properties.

The general idea behind FDM is that the differential operator is approximated by replacing the derivatived using difference quotients, which are linear combination of function values at the grid points). In order to do so the space and time domain are *partitioned* in a grid like fashion and approximated solutions are computed only for those discrete grid points. The error, between the numerical solution and the exact solution is determined by the difference formula utilized and is commonly refeered as *truncation*[1] or *discretization* error.

[1]The term truncation comes from the fact that a finite difference quotient is a truncation of the Taylor expansion.

Depending on how the derivatives are approximated, explicit or implicit FDM schemes are obtained. When forward difference formulas are considered, the resulting difference equation is generally expressed in terms of an explicit recurrence formula, while backward difference formulas generally lead to implicit recurrence formulas involving unknown values, and therefore require the solution of a linear system to obtain the new state of the system at each grid point.

2.2 Finite difference formulas

Difference formulas can be derived from Tailor's expansion of T_{j+1}^n in terms of T_j^n and its derivatives as:

$$T_{j+1}^n = T_j^n + \frac{\partial T(t,x)}{\partial x}\bigg|_j^n \Delta x + \frac{1}{2!}\frac{\partial^2 T(t,x)}{\partial x^2}\bigg|_j^n \Delta x^2 + \ldots + \frac{1}{k!}\frac{\partial^k T(t,x)}{\partial x^k}\bigg|_j^n \Delta x^k + \ldots \tag{2.1}$$

If series is truncated after the second term ($k = 1$) and solving for $\frac{\partial T(t,x)}{\partial x}$ the following is obtained:

$$\frac{\partial T(t,x)}{\partial x} = \frac{T_{j+1}^n - T_j^n}{\Delta x} \tag{2.2}$$

Equation 2.2 is called fist **forward** finite difference approximation. Other approximations are possible and are easilby obtainable by expanding different points of the grids and using more points from the expansion.

Backward finite difference quotient can be obtained from the Taylor's expansions of

$$T_{j-1}^n = T_j^n - \frac{\partial T(t,x)}{\partial x}\bigg|_j^n \Delta x + \frac{1}{2!}\frac{\partial^2 T(t,x)}{\partial x^2}\bigg|_j^n \Delta x^2 + \ldots + \frac{1}{k!}\frac{\partial^k T(t,x)}{\partial x^k}\bigg|_j^n \Delta x^k + \ldots \tag{2.3}$$

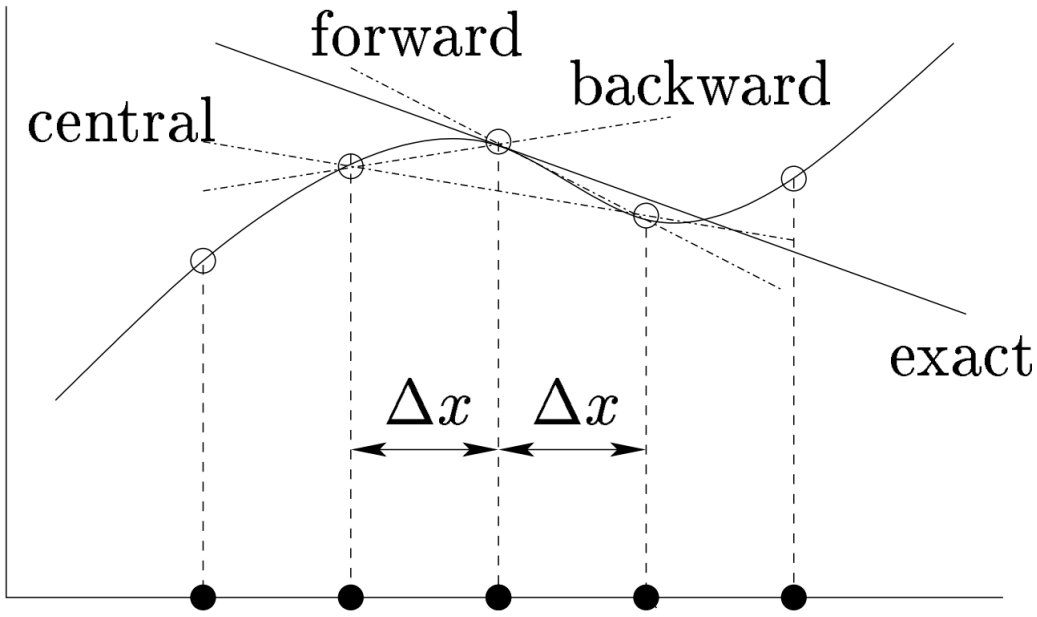


Figure 2.1: Explicit FDM discretization for the 1D heat conduction problem

which can be rearranged in the following manner

$$\frac{\partial T(t,x)}{\partial x} = \frac{T_j^n - T_{j-1}^n}{\Delta x} \tag{2.4}$$

The same approach can be used to derive approximation for higher order derivatives. For example equation 2.5, known called **central difference formula**, is an approximation for the second order derivative and can be obtained retaining the firsts four terms in both equations 2.1 and 2.3 and adding the resulting expression:

$$\frac{\partial^2 T(t,x)}{\partial x^2} = \frac{T_{j+1}^n - 2T_j^n + T_{j-1}^n}{\Delta x^2} \tag{2.5}$$

Figure 2.2 show how finite difference formulas can be interpreted geometrically.

2.3 Heat Equation

As an example a simple FDM scheme for an initial-boundary condition problem for the heat conduction problem is derived.

$$\frac{\partial T(t,x)}{\partial t} = k\frac{\partial^2 T(t,x)}{\partial x^2} \tag{2.6}$$

where $0 \leq t \leq L$ and $0 \leq x \leq M$. In order to construct a FD approximation for equation 2.6

1. Both space and time domain are partitioned into a finite grid as follows:

$$x_j = j\Delta x,\ j = 0, 1, \ldots, M,\ \Delta x = \frac{1}{M} \tag{2.7}$$

$$t_n = n\Delta t,\ n = 0, 1, \ldots, L,\ \Delta t = \frac{1}{L} \tag{2.8}$$

2. First and second order derivative appearing in 2.6 are substituted by forward and central difference formulas respectively leading to (see Figure 2.3):

$$\frac{T_{j+1}^n - T_j^n}{\Delta x} = k\frac{T_{j+1}^n - 2T_j^n + T_{j-1}^n}{\Delta x^2} \tag{2.9}$$

3. Equation 2.6 is evaluated at grid point $(n\Delta t, j\Delta x)$

$h = \Delta x = 0.2$ and $\Delta t = 0.01$ with initial conditions $T(0, 0 < x < M) = 0$ and boundary conditions $T(0, x = 0) = 100\ , u(0, x = L) = 100$. Note that, assuming $k = 1$, the values assigned to the spatial and temporal steps satisfy the von Neumann stability analysis. In fact, $\frac{k\Delta t}{\Delta x^2} = 1/4 \leq 1/2$.

In this example, a one-dimensional cellular space R can be used (cf. Figure **??**a) to model the 10 grid points i $(i = 0, 1, ..., 9)$, while the values that the

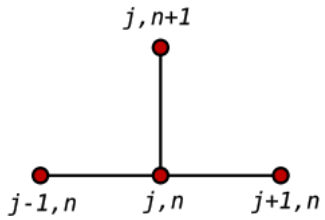


Figure 2.2: Explicit FDM discretization for the 1D heat conduction problem

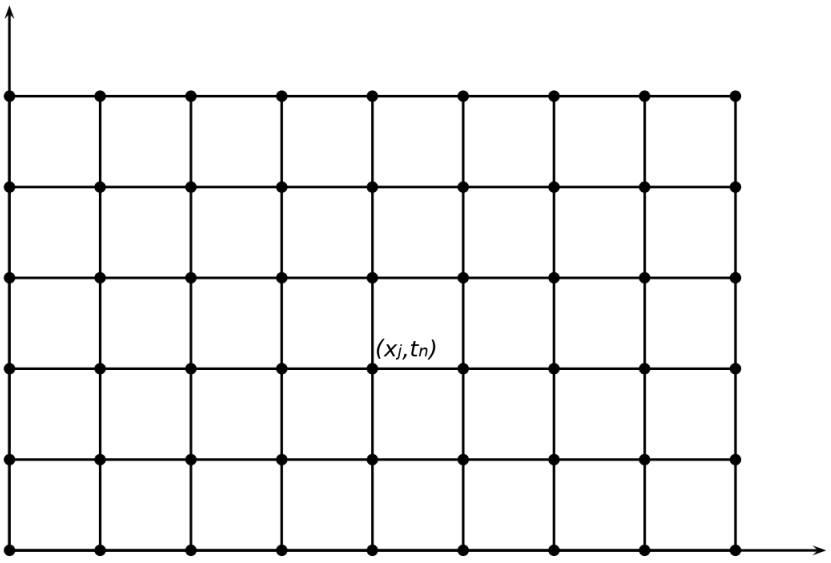


Figure 2.3: 1D heat equation FDM grid space and time partitioning.

generic cell can assume are simply represented by the $Q = [0, 100] \subset \mathbb{R}$ substate. By using the forward difference formula for the first order derivative and the central one for the second order, the following explicit recurrence scheme is obtained:

$$T_i^{t+1} = T_i^t + k\Delta t \frac{T_{i+1}^t - 2T_i^t + T_{i-1}^t}{\Delta x^2}$$

denoting that the value of the temperature at the grid point i at the time step $t+1$ is a function of the (known) values assumed by T at the $i-1$, i and $i+1$ grid points at the (previous) time step t. Thus, the one dimensional radious-one $X = \{-1, 0, 1\}$ neighburhood pattern can be adopted (cf. Figure **??**a) and an elementary process used to implement the above recurrence scheme. The explicit formulation of the FDM one-dimensional heat conduction model can be therefore expressed in terms of the XCA formalism as:

$$FDM_{heat} =< R, \Gamma, X, Q, P, \sigma, \gamma >$$

where:

- $R = \{0, 1, ..., 9\}$ is the one-dimensional cellular space, representing the integer coordinates of the cells of the computational domain (cf. Figure **??**a).
- $\Gamma = \{0, 9\} \subset R$ is the region identifying the boundary of the computational domain.
- $X = \{-1, 0, 1\}$ is the geometrical pattern defining the neighborhood relationship (cf. Figure **??**a).
- $Q = [0, 100] \subset \mathbb{R}$ is the set of cell's states used to express the temperature values at the grid points.

- $P = \{k = 1, \Delta t = 0.01, \Delta x = 0.2\}$ is the set of the FDM model parameters.
- $\sigma : Q^3 \to Q$ is the cell's transition function, which implements the explicit recurrence FDM scheme.
- $\gamma : Q^{|\Gamma|} \to Q^{|\Gamma|} \times \mathbb{R}$ is the global steering function, used to apply the boundary conditions at each computational step.

The initial conditions of the system are preliminarly defined at time $t = 0$ by assigning the temperature value 0 to each grid point, except for the boundaries where the value 100 is set. The global evolution of the system can therefore be obtained by applying the τ global transition function (which in turn applies σ to each grid point) and eventually the steering function γ at discete time steps.

XCA can be employed for both explicit and implicit schemes to represent FDM models in formal terms. In fact, in case of an explicit scheme, the computational domain can be represented by means of the R cellular space and the coordinates of the grid points involved in the recurrence formula defined by means of the X neighbourhood relationship. Moreover, the values of involved variables can be represented in terms of substates and the explicit recurrence formula easily expressed in terms of elementary processes. Instead, while dealing with a linear system resulting from an implicit FDM scheme, a steering function can be employed instead of elementary processes, together with an external linear algebra solver.

It is worth to recall that physically-based models laying on a XCA direct discrete approach (i.e., not going through the discretization of differential equations) can lead to the same discrete formulations achieved with the FDM, making these latter formulations a specific case of the general XCA approach. Mendicino et *al.* [**?**] proved that their direct discrete formulation of the Darcy equation for modelling unsaturated flow in a three-dimensional cubic cell system

is similar to the one achieved using an explicit FDM or finite volume scheme. However, the same discrete governing equation system would allow a greater level of convergence with respect to traditional methods if an irregular mesh is used and a not linear (e.g., quadratic) interpolation of the hydraulic head on the cells is adopted (e.g., Tonti proved it for the Finite Element Method [**?**]). This is a potential of the XCA approach that will be exploited in future versions of OpenCAL, where also non-regular grids will be allowed.

Chapter 3

Tracking Particles

Bibliography

[1] Conway J. H Berlekamp, E. R. What is life? *Winning Ways for Your Mathematical Plays, Games in Particular*, 2, 1982.

[2] Bastien Chopard and Pascal O. Luthi. Lattice boltzmann computations and applications to physics. *Theoretical Computer Science*, 217(1):115–130, March 1999.

[3] J. Conway. The game of life. *Scientific American*, 1970.

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

[5] Gino M. Crisci, Rocco Rongo, Salvatore Di Gregorio, and William Spataro. The simulation model sciara: the 1991 and 2001 lava flows at mount etna. *Journal of Volcanology and Geothermal Research*, 132(2-3):253–267, April 2004.

[6] Wells D. *The Penguin Dictionary of Curious and Interesting Numbers*. Penguin Books, 1986.

[7] Rocco Rongo William Spataro Giuseppe A. Trunfio Donato D'ambrosio, Giuseppe Filippone. Cellular automata and gpgpu:an application to lava flow modelling. *International journal of Grid computing*, 4:30–47, 2012.

[8] M. Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, C-21(9):948–960, Sept 1972.

[9] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the navier-stokes equation. *Phys. Rev. Lett.*, 56(14):1505–1508, April 1986.

[10] Vichniac G. Simulating physics with cellular automata. *Physica*, D 10:96–115, 1984.

[11] Ruxton G.D. Effect of the spatial and temporal ordering of events on the behavior of a simple cellular automaton. *Ecological Modelling*, 84:311,314, 1996.

[12] Pat Gelsinger. Intel spring forum. 2004.

[13] Moore Gordon. Cramming more components onto integrated circuits. *Electronics*, 38:114 ff., 1965.

[14] Salvatore Di Gregorio and Roberto Serra. An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. *Future Generation Computer Systems*, 16(23):259 – 271, 1999.

[15] Andrew Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific, Singapore, 2001.

[16] Sheng Yu Karel Culik. *Undecidability of CA Classification Schemes*. 1998.

[17] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[18] C.G. Langton. Computation at the edge of chaos. Master's thesis, Univeristy of Michigan, 1990.

[19] Chris G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. *Phys. D*, 42(1-3):12–37, June 1990.

[20] G. Vichniac N. Margolus, T. Toffoli. Cellular automata supercomputers for fluid-dynamics modelling. *Phys. Rev. Lett.*, 56:1694–1696, 1986.

[21] Robert J. Fowler Nathan Froyd, John M. Mellor-Crummey. Low-overhead call path profiling of unmodified, optimized code. *International Conference on Supercomputing*, 2005.

[22] Von Neumann. Theory of self reproducing automata. 1966.

[23] Nvidia. *CUDA C Best practices*. Nvidia, 2012.

[24] Nvidia. *CUDA C Dynamic Parallelism*. Nvidia, 2012.

[25] Nvidia. *CUDA C Programming Guide*. 2012.

[26] Seung Park and James D. Iversen. Dynamics of lava flow: Thickness growth characteristics of steady two-dimensional flow. *Geophysical Research Letters*, 11(7):641–644, 1984.

[27] Barbara Chapman Rungan Xu, Sunita Chandrasekaran. An openacc code for a c-based heat conduction code. *www.openacc.com*, 2012.

[28] F. Higuera S. Succi, R. Benzi. The lattice boltzmann equation: A new tool for computational fluid dynamics. *Physica*, 47:219–230, 1991.

[29] G.Ch Sirakoulis, I Karafyllidis, and A Thanailakis. A cellular automaton model for the effects of population movement and vaccination on epidemic propagation. *Ecological Modelling*, 133(3):209–223, September 2000.

[30] William Spataro, Maria V. Avolio, Valeria Lupiano, Giuseppe A. Trunfio, Rocco Rongo, and Donato DAmbrosio. The latest release of the lava flows simulation model sciara: First application to mt etna (italy) and solution of the anisotropic flow direction problem on an ideal surface. *Procedia Computer Science*, 1(1):17–26, May 2010.

[31] N. Margolus T. Toffoli. *Cellular Automata Machines: A New Environment for Modeling*. MIT Press, 1987.

[32] J.W. Thatcher. Universality in the von neumann cellular mode. *A. W. Burks (Ed.), Essays on Cellular Automata*, pages 103–131, 1970.

[33] Tommaso Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D: Nonlinear Phenomena*, 10(1-2):117–127, January 1984.

[34] Oreste Villa, Daniel Chavarra-mir, Vidhya Gurumoorthi, Andrs Mrquez, and Sriram Krishnamoorthy. Effects of floating-point non-associativity on numerical computations on massively multithreaded systems , 2013.

[35] Richard Walter and Thomas Worsch. *Efficient Simulation of CA with Few Activities*, volume 3305 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.

[36] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:471–526, 1983.

[37] S. Wolfram. Cellular automaton uids 1: Basics theory. *Journal of Statistical Physics*, 45:471–526, 1986.

[38] Stephen Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96(1):15–57, 1984.

[39] Stephen Wolfram. *A new kind of science*. Wolfram Medica, Inc, 2002.

$$\alpha = \sqrt{\beta} \tag{3.1}$$

3.0.1 Subsection Heading Here

Write your subsection text here.

3.1 Conclusion

Write your conclusion here.