

R.8612



Tesis
I-18

UNIVERSIDAD AUTÓNOMA MADRID REGISTRO GENERAL
Entrada 01 Nº. 200400005207 10/02/04 15:29:27



Universidad Autónoma de Madrid



2630797

Ph.D. THESIS

Genetic Evolution and equivalence of some complex systems:

Fractals, Cellular Automata and Lindenmayer Systems

AUTHOR

Abdelatif Abu Dalhoum

Supervisors

Dr. Manuel Alfonseca

Dr. Alfonso Ortega

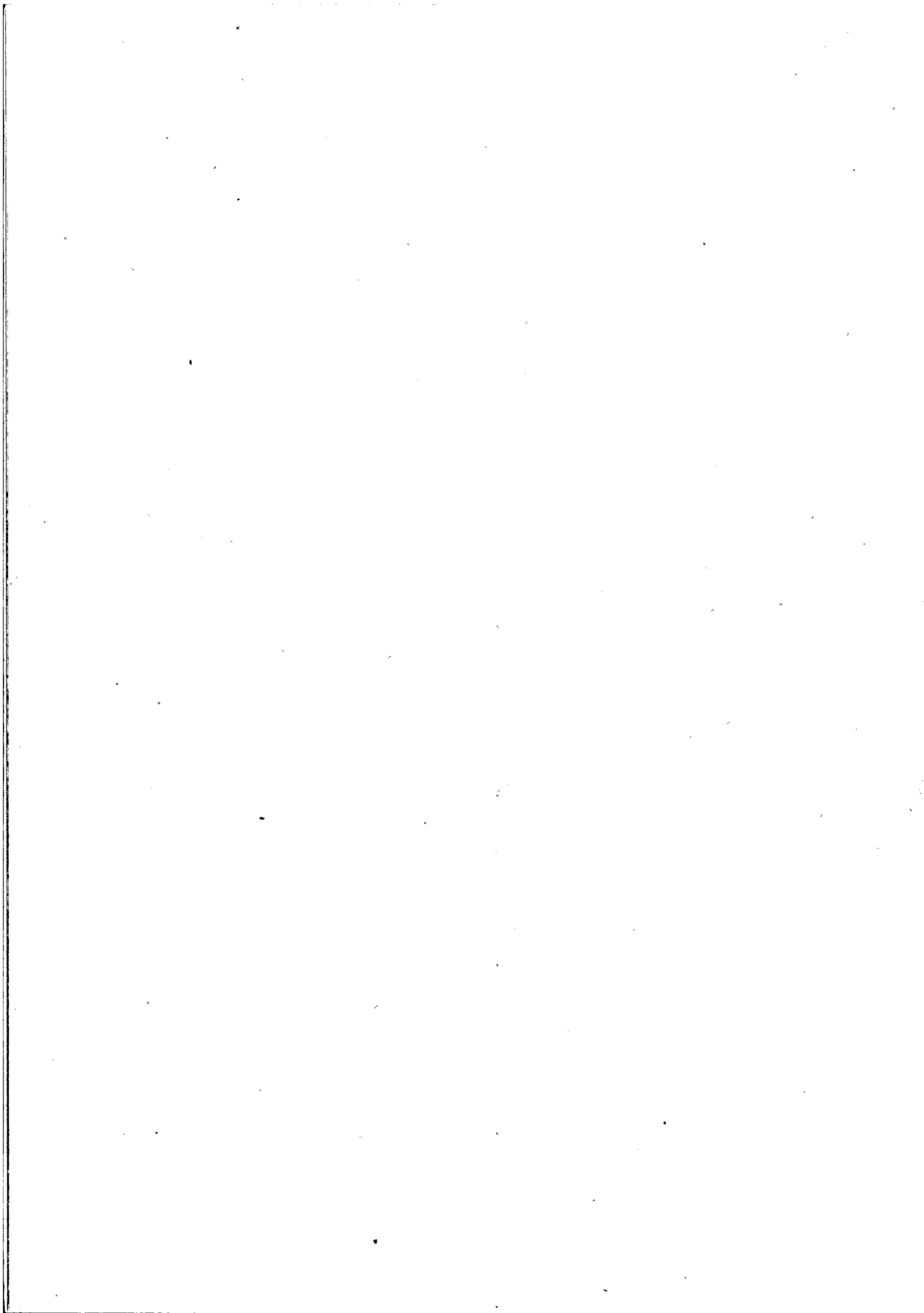


*Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid*

INF-DON-146-2

February 2004

*Dissertation submitted in partial fulfillment of the requirements for the degree of Ph.D. in
Computer Science*

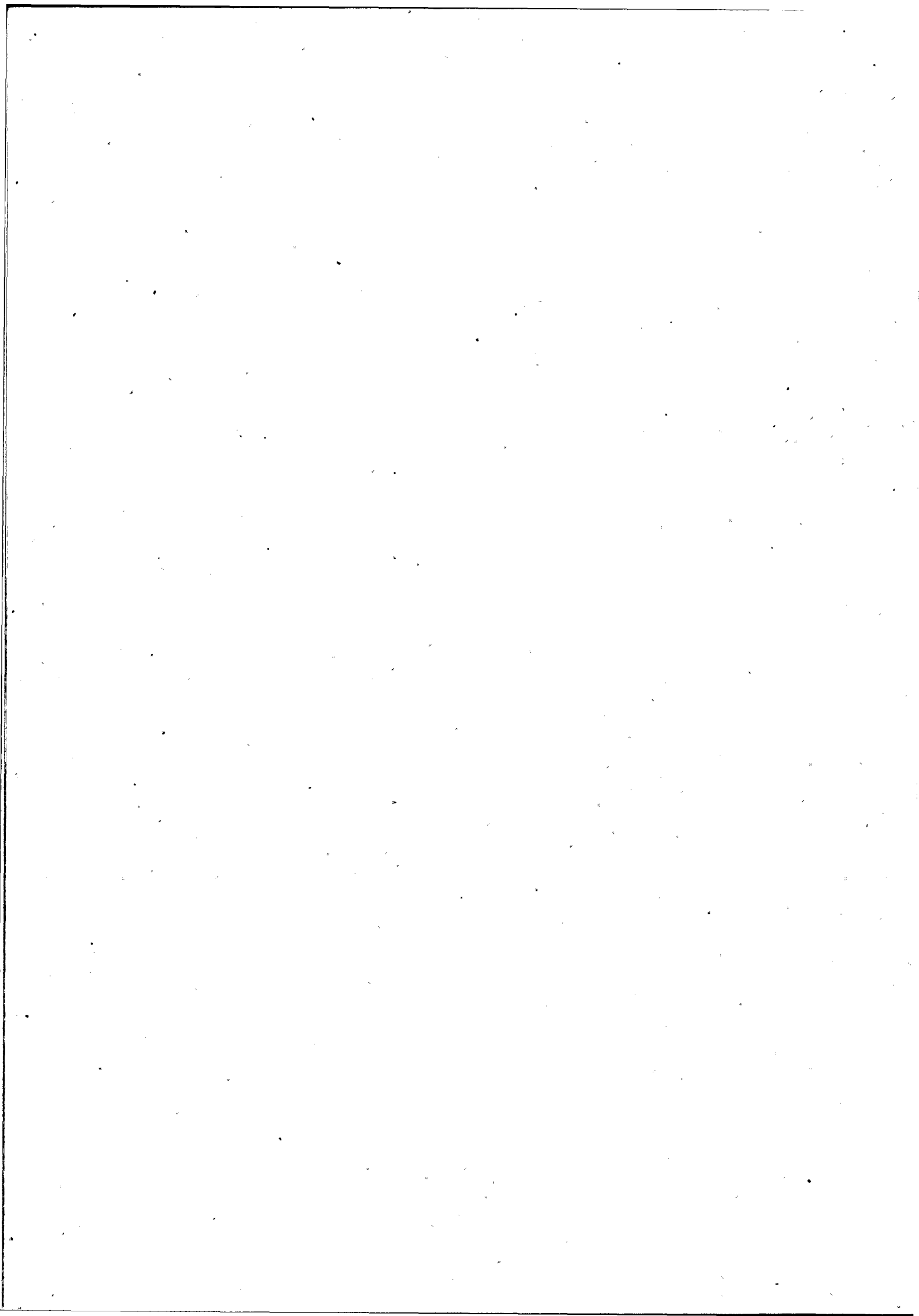


الإهداء

إلى روح والدي الغالية شفيقة العطاوي
حُبًا ووفاءً

Dedication

To the memory of my mother



Acknowledgments

First of all, I would like to thank my two supervisors Dr. Alfonso Ortega and Dr. Manuel Alfonseca, without whom I would have never obtained this dissertation; I am really wordless to thank them.

Also I would also like to thank the AECI (Agencia Española de Cooperación Internacional) for their scholarship for my first four years, and to thank the GHIA group in our department for their scholarship for the last two years.

Also I would like to thank all my friends Pablo, Leila, Pedro, German, Abraham, Manuel, Jose, Fran, Diana, Alejandro, Pablo, Miguel, Rosa, Alvaro, Ruth, Maria, Estrella, Boumedian, Ibrahiem, Hussien, Basher...etc, and specially Enrique how helped me while writing the thesis correcting my grammatical errors, for the fantastic years I spent with them thank you all.

Finally, and not less importantly, I would like thank all the members of my family: Farida, Lotfy, Jamal, Mohammad, Ali, Raed, Iman, Rashed and Mohammad, for their engorgements.

And I am especially grateful to my wife Hiba and to my daughter Shafieka, the light of my life, for whom I live, and without them there is no meaning for my life.

Table of contents

Acknowledgments.....	i
List of tables.....	vii
List of figures.....	ix
Introduction.....	1
Motivation and plan of the thesis.....	3
Motivation.....	3
Objectives of this thesis.....	7
Plan of the thesis.....	8
Basic Concepts.....	11
Chapter 1 Evolutionary Algorithms.....	13
1.1. Overview.....	13
1.2. Genetic Algorithms.....	13
1.2.1. Individuals and Initial Population.....	14
1.2.2. Fitness Function (Evaluation).....	14
1.2.3. Generating a New population: genetic operators.....	14
1.3. Genetic programming (GP).....	16
1.4. Evolutionary Strategies (ES) and Evolutionary Programming (EP).....	16
1.5. Variants of the classical GA/GP scheme.....	16
1.6. Grammatical evolution (GE).....	17
1.6.1. Genotype-Phenotype mapping.....	17
Chapter 2 Fractals.....	25
2.1. Overview.....	25
2.2. Categorizing Fractals.....	26
2.2.1. Initiator-Iterator Fractals.....	26
2.2.2. Random Fractals.....	28
2.2.3. Other ways of defining fractals: affine transformation fractals and iterated function systems.....	29
2.2.4. Iterative Dynamical System Fractals.....	30
2.3. How can we estimate the dimension of fractals.....	30
2.3.1. Hausdorff dimension.....	31
2.3.2. Richardson-Mandelbrot dimension estimation.....	31
2.3.3. Box Counting.....	32
2.4. Multi-fractals.....	33
2.5. Application of Fractals.....	33
Chapter 3 Lindenmayer Systems (L-Systems)	35
3.1. Overview.....	35
3.2. Classes of L-Systems.....	37



3.2.1. 0L systems.....	37
3.2.2. $\langle k, I \rangle$ IL Systems.....	39
3.2.3. Systems with tables.....	40
3.2.4. Systems with extensions.....	40
3.2.5. Other combinations.....	40
3.2.6. Other L-System extensions.....	41
3.3. L-System Design.....	41
3.3.1. Ad-hoc solution.....	42
3.3.2. Genetic L-System programming.....	42
Chapter 4 Fractals and L-Systems.....	45
4.1. Overview.....	45
4.2. Turtle Graphics Method.....	45
4.3. Vector Graphics Method.....	46
4.4. Equivalence between the Turtle and Vector graphical representations of L-Systems.....	47
4.5. Computing the dimension of initial-iterated fractals.....	48
Chapter 5 Cellular automata.....	51
5.1. Overview.....	51
5.2. Description and types of Cellular automata.....	51
5.2.1. Lattice.....	52
5.2.2. Neighborhood.....	52
5.2.3. Local Transition Function.....	54
5.2.4. Classic Cellular Automata.....	55
5.2.5. Cellular automata variants.....	58
5.3. Cellular automata and the edge of chaos.....	58
5.4. Cellular Automata Programming.....	59
5.4.1. Task classification problem.....	59
5.4.2. GAs: How they work with CA.....	60
5.4.3. Signals.....	61
Chapter 6 L-Systems and Cellular Automata.....	63
6.1. Overview.....	63
6.2. L-Systems and Cellular automata.....	63
6.3. One-dimensional binary cellular automaton with three inputs that generates the Sierpinski gasket.....	64
6.4. IL-System equivalent to bi-dimensional CA that simulate ecosystem model.....	65
6.5. IL-System equivalent to three-dimensional CA that generates and propagates pulses.....	67
6.6. Equivalence between L-Systems and Cellular automata.....	68
Genetic Evolution & Equivalence between some Complex Systems.....	71
Chapter 7 Grammatical Evolution to Design Fractal Curves with a Given Dimension.....	73
7.1. Overview.....	73
7.2. The design of L-Systems that represent curves with a given fractal dimension.....	74
7.3. The developmental algorithm.....	75

7.4. The genetic algorithm.....	78
7.5. Parallels to biological evolution.....	79
7.6. Evolving the turtle angle.....	81
7.7. Results.....	82
7.8. Examples.....	87
Chapter 8 Evolving the game of life with a genetic algorithm.....	99
8.1. Overview.....	99
8.2. Concise representation of life-related cellular automata.....	99
8.3. A genetic algorithm that evolves into the game of Life.....	100
8.4. Experiments and results.....	101
Chapter 9 Cellular automata equivalent to PDOL systems.....	105
9.1. Overview.....	105
9.2. One-dimensional cellular automata.....	105
9.2.1. Informal description.....	106
9.2.2. Formal definitions.....	106
9.3. One-dimensional cellular automata n-equivalent to PDOL systems.....	107
9.3.1. Informal description.....	107
9.3.2. Formal description.....	111
9.3.2.a. Theorem.....	111
9.3.2.b. Proof.....	112
9.3.2.c. Example.....	115
Chapter 10 Cellular Automata equivalent to DOL Systems.....	117
10.1. Overview.....	117
10.2. One dimensional Cellular automata equivalent to DOL Systems.....	117
10.2.1. Informal description.....	117
10.2.2. Formal description.....	122
10.2.2.1. Theorem.....	122
10.2.2.2. Proof.....	122
10.3. Examples.....	126
Chapter 11 Cellular automata equivalent to DIL systems.....	131
11.1. Overview.....	131
11.2. One dimensional Cellular Automata equivalent to DIL Systems.....	131
11.2.1. Informal description.....	131
11.2.2. Formal description.....	135
11.2.2.1. Theorem.....	135
11.2.2.2. Proof.....	136
11.3. Examples.....	141
Conclusions and Future Work.....	149
Conclusions.....	151
Open lines of work.....	153
References and Bibliography	155

Table list

2-1	The length of Von-Koch curve.....	27
3-1	Derivation steps that generate the Fibonacci sequence	39
5-1	Wolfram representation of elementary CA's	55
6-1	Cellular automata transition function 90	64
7-1	Results of experiments to get optimal values of genetic operation rates	79
7-2	Number of generations to reach the target in a set of tests of our Grammatical Evolution approach.	82
7-3	Different fractal curves sharing the same dimension, evolved by our method.	83
7-4	A few fractal curves evolved by our method	84
7-5	A ser of tests where the turtle angle was evolved too	87
7-6	Genetic algoritm execution for fractal dimension 1.58 with angle 60	87
7-7	Genetic algoritm execution for fractal dimension 1.58	92
8-1	Number of generations to reach the goal, fro different settings in the mutation rates and the initial conditions	102
8-2	Convergence speed as a function of grid size	
10-1	Transition function of the cellular automaton equivalent to a D0L system	124
11-1	Transition function of the cellular automaton equivalent to a D1L system	138

List of Figures

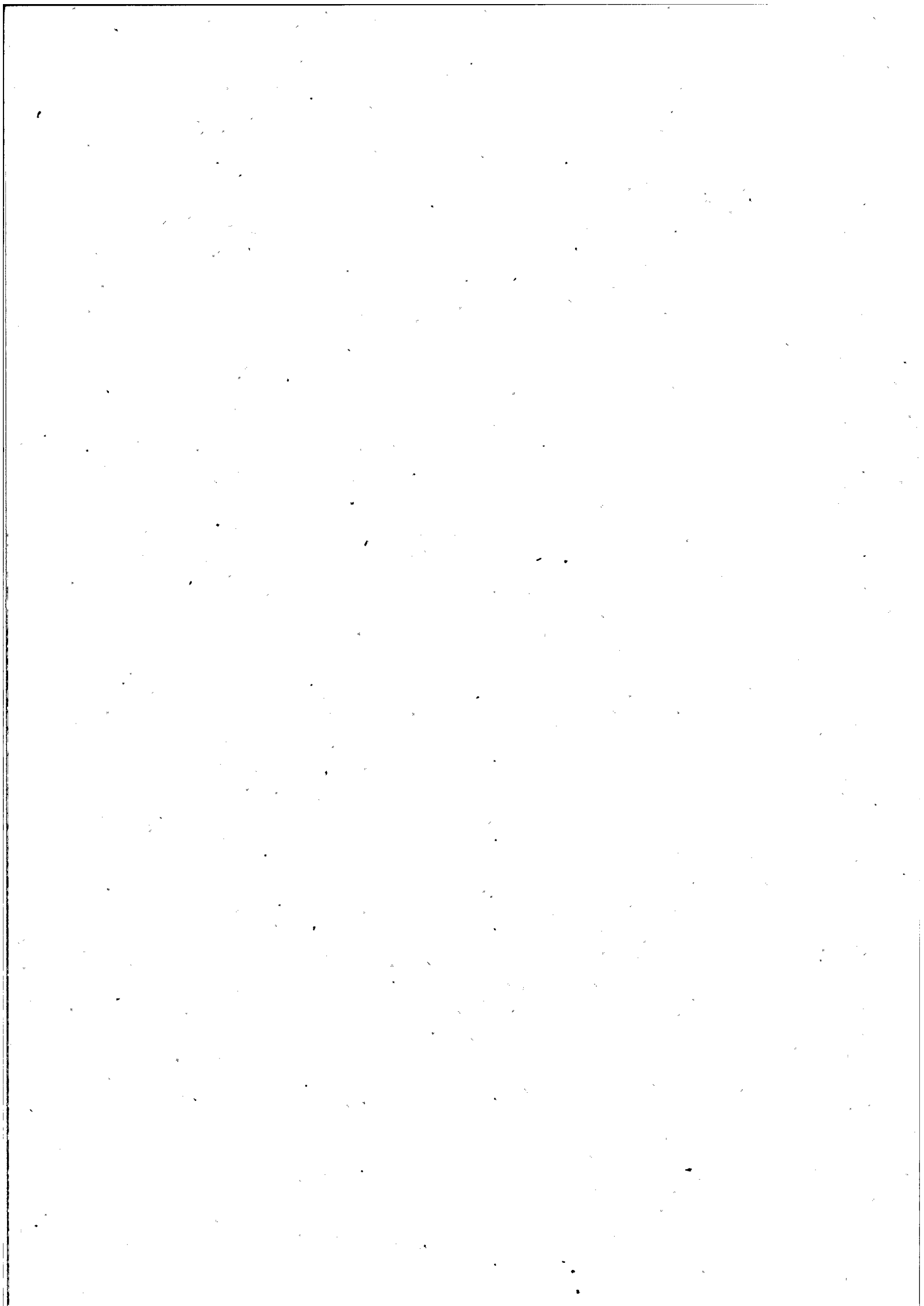
1-1	Pseudo-code of a standard genetic algorithm	14
2-1	The first five steps of Cantor set construction	26
2-2	Von Koch's curve	27
2-3	The iterator of Peano curve	28
2-4	Example of Brownian-motion fractal, from [Int].	29
2-5	Illustrates an example of a bacteria aggregation fractal [Int].	29
3-1	Development process of an artificial tree using a special type of L-systems called stochastic L-systems (from [Pru94])	35
3-2	A model of a date palm tree simulated by an L-system variant (from [Pru95]).	36
3-3	Found in [Pru95], illustrates the development of a plant	36
3-4	Plant growth simulation using Genetic Programming and L-systems, from [Jac94]	43
5-1	Von Neumann neighborhood in a bi-dimensional grid with a central position.	53
5-2	Von Neumann's neighborhood in a tri-dimensional grid with a central position.	53
5-3	Moore's neighborhood	54
5-4	De Bruijn representation of rule 54 (from [Del98]).	55
5-5	a) One initial configuration for Conway's Life. b) The second generation .	57
6-1	The first 24 generations of the CA90 from 0...010...0	65
6-2	Three dimensional cellular automaton grid	67
7-1	Parallels between our Grammatical Evolution approach and biological evolution.	81
7-2	Four different fractal curves evolved for a target dimension of 1.255	83
7-3	A fractal curve with the same dimension as von Koch's snowflake	84
7-4	A fractal curve with approximate dimension 1.5	85
7-5	A fractal curve with approximate dimension 1.9	85
7-6	A fractal curve with approximate dimension 1.7	85
7-7	A fractal curve with approximate dimension 1.8	86
7-8	A fractal curve with approximate dimension 1.7	86
7-9	A fractal curve with approximate dimension 1.6	86
7-10	The initiator object	88
7-11	The iterator of Sierpinski	88
7-12	Sierpinski fractal after 9 iterations	88
7-13	The iterator object	89
7-14	The fractal after 6 iterations	89
7-15	The iterator object	89

7-16	The fractal after 6 iterations	90
7-17	The iterator object	90
7-18	The fractal after 4 iterations	90
7-19	The iterator object	91
7-20	The fractal after 5 iterations	91
7-21	The iterator object	91
7-22	The fractal after 9 iterations	92
7-23	The iterator object	92
7-24	The fractal after 5 iterations	93
7-25	The fractal iterator	93
7-26	The fractal after 6 iterations	93
7-27	The fractal iterator	94
7-28	The fractal after 6 iterations	94
7-29	The fractal iterator	94
7-30	The fractal after 6 iterations	95
7-31	The fractal iterator	95
7-32	The fractal after 4 iterations	95
7-33	The fractal iterator	96
7-34	The fractal after 3 iterations	96
7-35	The fractal iterator	96
7-36	The fractal after 4 iterations	97
7-37	The fractal iterator	97
7-38	The fractal after 3 iterations	97
7-39	The fractal iterator	98
7-40	The fractal after 4 iterations	98
8-1	Number of differences of best evolved automaton and Conways game o life	102
9-1	Definition of Cellular automata equivalent to PDOL systems	108
9-2	Initial generation of the cellular automaton	109
9-3	Cellular Automaton – DOL system comparison: the first 12 generations of the a utomaton simulate the first two derivations of t he L system. T he automaton subsequent states are shown in the left side.	110
10-1	Initial generation of the cellular automaton.	118
10-2	Cellular Automaton – DOL system comparison :.....	119
10-3		129
10-4		130
10-5		130
11-1	The initial configuration of the cellular automaton	131
11-2	At the first step in the derivation, signal < is transmitted to the left	132
11-3	Ending the first derivation in the cellular automaton equivalent to the DIL system	133
11-4	The cellular automaton mechanisms to generate the second word derived by the DIL-system	134
11-5	The cellular automaton joins the contexts of the deleted cells and keeps it unchanged until the end of the derivation, when a especial signal ($\diamond' \leftarrow$) is sent to the left, putting the correct context in its place, identical to the new	135

	derived word.....	
11-6	Derivation of AB from BBABB in the cellular automaton described in example 1.....	143
11-7	Derivation of λ from BBB in the cellular automaton described in example 2.....	146

Introduction





Motivation and plan of the thesis

Motivation

Theoretical Computer Science is a basic discipline of Computer Science, because most of the advances in this field are supported on solid results from that discipline. It started with concurrent developments in formal logics, digital electronics and linguistics, which gave rise to the Theory of Formal Languages and Automata, a main subject of study for every computer scientist.

In the last years, perhaps due both to the rise in the computing power of computers and to the nearness of the physical limits in the miniaturization of electronic components, there has been a resurgence of interest about formal computational models, which could be an alternative to the classic architecture proposed by John von Neumann.

Many of these models are inspired by the way in which nature solves efficiently very complex problems. Natural systems are thus formalized and their properties studied. Most of them are computationally complete, therefore they are considered new programming paradigms. We can mention, for example, cellular automata, and the use of parallel-derivation grammars, which were designed during the sixties, respectively by John von Neumann and Aristid Lindenmayer, as a discrete alternative to traditional simulation techniques, based on continuous functions and differential equations. Other examples are quantum programming, or computations inspired by DNA, cellules, and their membranes and different constitutive parts.

This situation shows, therefore, a wide range of abstract architectures, which are as powerful as conventional computers and, sometimes, even more efficient. Most of them improve the execution time (performance) needed to solve NP-complete problems, providing non-exponential solutions. In this thesis we shall use cellular automata and Lindenmayer systems.

With all these new tools, a new basic problem appears: how to program these architectures, or, in other words, how to design particular instances of these architectures that can solve some specific problem. To imagine the difficulty of the answer, it is enough to remember that this question, focusing on traditional computers, has required the development of its own branch of engineering: *Software Engineering*.

The hardest difficulty to program these architectures is the fact that they are, in general, specialized to solve some very particular types of problems or, at least, problems that are encoded in a specific way. For instance, in the case of DNA-computing, it is possible to find a string of symbols complementary to a given input string, with a performance independent of the length of the input string. Cellular automata have been applied to simulate several physical processes, which have in

common the possibility of dividing the space in a regular way. Finally, Lindenmayer systems (or parallel-derivation grammars) can simulate processes that simultaneously change in a different way at different locations.

Nevertheless, in most of the cases, there are no tools to program these architectures, as there are with general-purpose computers. We might say that we are as far from our goal as the programmers of the ENIAC computers were from CASE tools. We do not have compilers, not even assemblers, and sometimes it is hard to imagine which is equivalent to machine code.

On the other hand, genetic algorithms are inspired by the mechanism of reproduction and natural selection (reproduction of the fittest individual of the population), with differentiation (recombination of the parents' genetic information and the possibility of mutations). It has been proven that these algorithms provide an efficient way for stochastic search of near-optimal (good enough) solutions for many kinds of problems. The only condition to use genetic algorithms is to represent the possible solutions in a proper way. This requisite is not really a restriction, because any computer-based solution of a problem requires the problem to be codified.

Automatic programming (writing programs that write programs) is one of the objectives of *Computer Science*, of special interest for this thesis. *Automatic programming* can be considered as the search for a program that performs the required task, among the set of all the possible programs. Therefore, it is possible to use genetic algorithms to solve this problem. This approach is called genetic programming. Candidate programs are individuals which have to compete for reproduction and to remain in the next generation of candidate solutions. Genetic programming can sometimes produce automatically programs with better performance than hand-made programs.

Grammatical evolution is one of the most recent variations of genetic programming, that generalizes this process in a way independent of the programming language. Throughout this thesis, we shall use both classical genetic algorithms and grammatical evolution.

Let us take a superficial look at the main concepts that will be considered during the development of this thesis. In the next part, the state of the art about all those concepts will be described in depth, as needed by the remainder of the thesis.

Fractals

There is not a general agreement in the definition of *fractal*. This term is used to refer to phenomena that share some mathematical characteristics:

- They behave as objects with a different dimension as the one that would apparently correspond to them. For instance, Peano defined a curve able to cover completely a surface; so, this curve (one-dimensional) behaves as a plane (two-dimensional). Classic definition of dimension is related to the degrees of

freedom of the studied object. Dots, curves, surfaces and volumes have, respectively, dimensions 0, 1, 2 and 3. This anomaly motivates the definition of *fractal dimension*, which allows assigning a fractional number to the dimension of an object. Consequently, a curve that behaves as a surface has a dimension close to 2; an object defined as a surface, but with a dimension of 2.8, will behave and share more properties with volumes than with surfaces. Several algorithms to calculate the fractal dimension have been proposed (e.g. Hausdorff-Besicovitch), though all of them provide the same value except for slight differences.

- In a fractal object, it is usually possible to find copies of itself, regardless of the scale used.

Formally, there are three main ways to represent fractal sets or phenomena:

- The limit between the convergence and divergence domains for some recursive complex-variable functions (as in Julia and Mandelbrot sets).
- The figure obtained as the limit (after an infinite number of iterations) of a base graphic (the initiator) using always the same transformation (the iterator).
- Random or Brownian movements.

Fractal geometry, therefore, completes some deficiencies of classic geometry. But its interest is not only theoretical; fractals have shown their expressive power for modeling many complex phenomena in the real world: meteorology, cardiac rhythm, economic fluctuations, etc.

Cellular Automata

Some biological systems create new individuals as copies of themselves (this property is called self-reproduction). Cellular automata, which try to model this characteristic, are mathematical abstractions of physical systems in which time, space and the variables that describe the states of the system are discrete. A cellular automaton has three main components: a finite automaton, a regular and not necessarily finite grid (which can be considered as a generalization of a matrix), and a rule for determining the neighborhood (the set of cells considered neighbors of a given cell in the grid). Each cell in the grid contains a copy of the finite automaton.

The global behavior of a cellular automaton can be described locally, because each finite automaton in the grid has as inputs the states of its neighbors. Even though finite automata are very simple devices, they can behave globally as very complex dynamic systems. Therefore, cellular automata have theoretical and practical interest: they have been used in simulation (traffic, geographic growth of cities, etc.), and some of them have the same theoretical computational capacity as universal computers, to the point that they have been considered apt for designing parallel computers.

Lindenmayer systems

Lindenmayer systems (or L-systems, in brief) were defined to describe multicellular organisms whose shape and size changes with time, and whose different parts change in a different way. These systems are known as *developmental systems*. L-systems are parallel-derivation grammars, i.e. their production rules are applied simultaneously. They have been used successfully to simulate various biological processes, such as plant growth, leaf development, seashell pigmentation, etc. They are also useful to represent other phenomena with complex behavior, such as some fractal objects, especially those of the initiator-iterator type, although an extension called *parametric L-systems* has been defined that is capable of representing the families of Mandelbrot and Julia fractals, and the like.

Genetic algorithms

Genetic algorithms perform stochastic searches inspired by the mechanisms of natural selection, identified by Charles Darwin, which, basically, are the survival of the fittest, the introduction of changes in the individuals by means of mutations, and sexual reproduction of the best adapted, to transmit part of their genetic legacy to their progeny. A genetic algorithm uses the following components:

- A representation of the possible solutions to the problem to be optimized by means of genetic algorithms. Each candidate solution is usually encoded as a string of symbols called *chromosome* or *genotype*. So the search space is a set of chromosomes or genotypes.
- A method for generating an initial population.
- A function that evaluates the nearness of a candidate solution to the target of the search. This function is usually called the *fitness function*.
- A set of genetic operators that combine the chromosomes of a generation to produce the next one. The two basic operators are recombination and mutation. The first one combines two chromosomes, splitting each one at one or several random positions, and mixing the parts together, producing two new chromosomes. Mutation, on the other hand, changes randomly the value of some positions in the chromosome.

At each generation, the fittest chromosomes are chosen. Genetic operators are applied to them to produce a new generation. This process is iterated until the fitness converges to a certain value. Some parameters of the algorithms are the size of the initial population, the way in which the random selections are made, the mutation rate (number of the individuals that mutate), the number of mutations, the size of the reproducing population, the total number of generations, etc.

Grammatical evolution

Grammatical evolution is a technique based on genetic algorithms, originally applied to automatic programming, that uses the grammar of the programming language to propose a new general mechanism to translate from genotypes to phenotypes in a deterministic way that minimizes syntactic mistakes. If the grammar of a different programming language is used, no further changes are necessary to automatically generate programs in the new language.

Grammatical evolution has shown to be as adequate as traditional genetic programming techniques and, for some benchmarks, even to improve the performance.

Objectives of this thesis

The objective of this thesis is double:

- To study the design of Lindenmayer systems and cellular automata to solve a given problem. We shall consider the application of genetic algorithms and grammatical evolution to handle this question.
- To complete the study of the equivalences between L-systems and cellular automata that the tutors of this thesis have previously worked on.

Application of genetic algorithms to design cellular automata that solve complete problems

The history of the evolution of cellular automata using genetic algorithms begins with Normal Packard and his team. They wanted to obtain binary one-dimensional cellular automata, to solve a classic classification problem, where the automaton should converge to a word with only *ones* if the initial word has more *ones* than *zeros*, and vice versa.

In this thesis, we use a genetic algorithm (among other techniques) to solve the problem of designing a cellular automata that solves a particular problem: Conway's game of life.

Application of grammatical evolution to design Lindenmayer systems that solve concrete problems.

As previously mentioned, grammatical evolution proposes a variant to genetic programming, which can be used when we have a grammar that the candidate solutions must conform to. It seems natural to use grammatical evolution, rather than classical genetic techniques, when the candidate solutions are Lindenmayer systems (a kind of grammars).

The supervisors of this thesis have previously developed an algorithm to compute the fractal dimension of some initiator-iterator fractals, represented by means of Lindenmayer systems. The algorithm obtains the dimension directly from the grammar.

In this thesis, the indicated algorithm will be used in the fitness function, to find, by means of grammatical evolution, L-systems that represent initiator-iterator fractals with a certain dimension. This problem can have practical interest, given that some industries are using elements with fractal geometry, because they improve the results obtained with traditional approaches. We can mention, in particular, the development of fractal antennas.

Study of the equivalences between L-systems and cellular automata

Apart from the differences between these two models, which will be shown in the first part of the thesis, L-systems and cellular automata share many properties, and it seems interesting to compare them. Their structural similarities have been underscored by applying both models to genetic programming in different works. The supervisors of this thesis have tackled previously the design of L-systems equivalent to given cellular automata. One of the objectives of the thesis is to finish this study in the reverse direction, by designing cellular automata equivalent to given L-systems.

In this thesis, cellular automata are considered equivalent to L-system if they generate the same language. The study begins with the simplest L-systems, PDOL. As explained in further chapters, PDOL stands for *propagative* (i.e. the symbols cannot be eliminated in the process of derivation), *deterministic* (i.e. each symbol can only be transformed in one way), and *context-free* (i.e., the transformation process is independent of the context) L-Systems. Next, rules that make it possible to delete symbols are added to build cellular automata equivalent to DOL systems (which are not necessarily propagative). Finally, context-dependent production rules are introduced, so as to design CA's equivalent to DIL systems (deterministic Lindenmayer systems with interactions).

Plan of the thesis

This thesis consists of five parts.

The first part (the one you are reading) describes the motivation and shows the plan of the thesis.

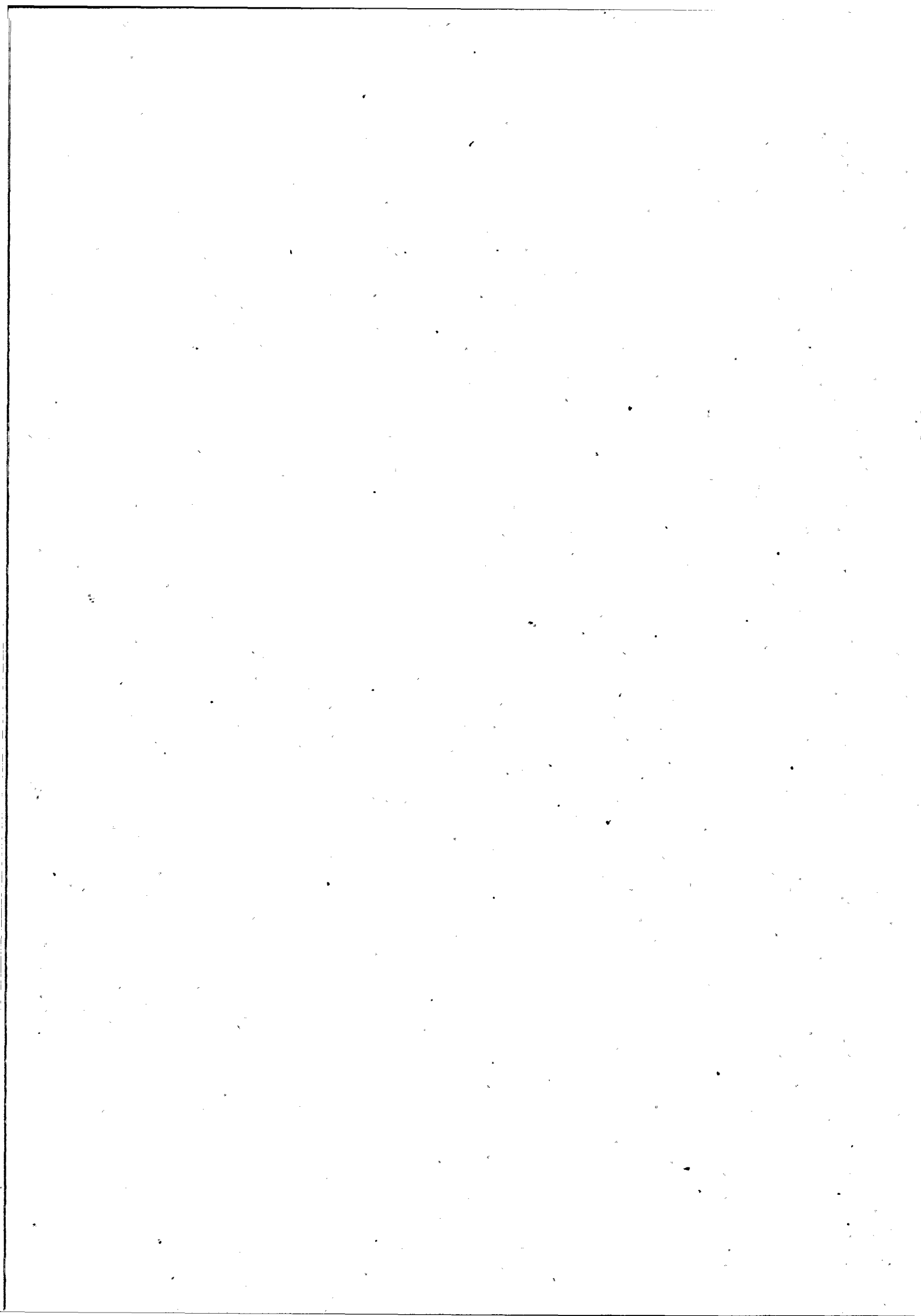
The second part introduces the basic concepts used in the thesis, and is divided into six chapters ("*Evolutionary Algorithms*", "*Fractals*", "*Lindenmayer Systems*", "*Fractals and Lindenmayer Systems*", "*Cellular Automata*" and "*Cellular Automata and Lindenmayer Systems*").

The third part describes the original results of the thesis, and divides into five chapters ("*Design of Fractal Curves with a given dimension by means of Grammatical Evolution*", "*Evolving the Game of Life*", "*Cellular Automata Equivalent to PDOL Systems*", "*Cellular Automata Equivalent to DOL Systems*" and "*Cellular Automata Equivalent to DIL Systems*").

The fourth part explains the conclusions and open lines of work.

The last part contains references.

Basic Concepts



Chapter 1

Evolutionary Algorithms

1.1. Overview

Evolutionary Algorithms (EA) are stochastic searching algorithms whose search methods are based on natural evolution, Darwinian theories of fighting for survival, genetic inheritance, natural selection and reproduction of best individuals [Gol89]. They provide robust search in complex space, and are computationally simple but powerful for finding optimal solutions in general search spaces. EA consider simultaneously several potential solutions that are treated as individuals to form a population. The individuals interact with each other and create new individuals to form a new generation.

The history of EA goes back to the middle of the 1960's, when three main branches of EA were defined: *Genetic Algorithms* (GA) by Holland [Hol67][Hol74], *Evolutionary Programming* (EP) by Fogel [Fog66], and *Evolutionary Strategies* (ES), by Beine, Rechenberg and Schwefel [Rec65][Rec73].

GA's have been applied successfully to solve a large number of problems in diverse disciplines. Several examples in optimization, automatic programming, economy, biology and ecology are described in [Mit96].

The following paragraphs briefly introduce GP, ES and EP. The reader can find a more detailed description in the *references* chapter.

GA and a recent variant of GP named Grammatical Evolution (GE) are used in depth in this thesis.

1.2. Genetic Algorithms

Figure 1-1 shows a possible scheme of a GA with the following main three steps:

- Generating an initial population.
- Re-ordering the population according to a fitness function.
- Generating a new population, by replacing the offspring of the better individuals for the worse.

Each step will be described in subsequent sections.

1. Generate an initial population
2. Compute the fitness of every individual
3. Sort the population from higher to lower fitness.
4. If the highest fitness individual's fitness is higher than the target fitness, stop and return this individual.
5. From the sorted population created in step 3, remove the individuals with least fitness and take the individuals with most fitness. Pair them. Each pair generates another pair, a copy of their parents, modified according to genetic operations. The new individuals are added to the remaining population and their fitness is computed.
6. Go to step 3.

Figure 1-1: Pseudo-code of a standard genetic algorithm

1.2.1. Individuals and Initial Population

Genetic algorithms work simultaneously on a population of possible solutions, usually represented by means of vectors or strings of symbols. These strings are treated as the population's genotypes. Genetic operators only handle the genotypes, because they are easier to manipulate than the candidate solutions themselves.

For example, if the algorithm is looking for an automatically written C program that performs a given task, genetic operators are easier to apply on each program's genotype, rather than on the actual C functions.

The initial population is usually generated at random, but it is worth noticing that the actual population used has an important influence on the convergence of the GA.

1.2.2. Fitness Function (Evaluation)

This function gives the highest values to the fittest individuals; therefore it evaluates how much each individual is close to the optimal solution. It has been proved [Gol89] that GA maximizes fitness functions.

1.2.3. Generating a New population: genetic operators

After ordering the population according to the fitness values of its individuals, a new population is generated. Genetic operators (crossover, mutation, elision, duplication...) are applied to the offspring of the best individuals, which then replace the worst individuals. There are other several possible approaches to select the better individuals [Gol89], but they are not used in this thesis.

Crossover

There are several different approaches to implement crossover. In this thesis, it will be done as described below:

Each pair of parents generates two new individuals by swapping segments of themselves. A single position inside both parents is randomly selected; each child has a different section of each parent's genome: the first half of the first parent and the second half of the second parent for one of them, and the first half of the second parent and the second half of the first parent, in that order. This approach is called one-point crossover.

For example, let us take two parents in the t -th generation (X_1^t and X_2^t)

$$X_1^t = \langle a_1, \dots, a_n \rangle$$

and

$$X_2^t = \langle b_1, \dots, b_n \rangle.$$

Let us assume crossover point is i , where $1 \leq i < n$. The offspring would then be

$$X_i^{t+1} = \langle a_1, \dots, a_i, b_{i+1}, \dots, b_n \rangle$$

$$X_j^{t+1} = \langle b_1, \dots, b_i, a_{i+1}, \dots, a_n \rangle$$

for some i and j in the new population.

Mutation

Mutation changes the value of some randomly chosen genes. It is known, from biology and ecology, that mutation introduces variability and diversity into the populations.

Mutation can be applied to every bit in the genome with a (usually small) probability, or it may be applied to a single bit with a much higher probability. This is the approach we are using in this thesis.

For example, let X_1^t be a genotype with 9 binary genes in some generation t .

$$X_1^t = \langle 1, 0, 1, 1, 0, 1, 0, 1, 1 \rangle$$

Let 6 be a randomly selected position (between 1 and 9). After mutation, X_1^t could become

$$\langle 1, 0, 1, 1, 0, 0, 0, 1, 1 \rangle$$

Other genetic operators

Two other genetic operators are also frequently used in GA's:



- Duplication: a segment, randomly chosen from one individual, is added at a given position of (possibly) another individual.
- Elision: one or more genes are deleted at random from the genome of an individual.

These two operators make it possible to mix, in the same population, genotypes of different lengths.

1.3. Genetic programming (GP)

Koza [Koz89] introduced GP as a model for automatically generating LISP programs to solve a given task.

Unlike genetic algorithms, GP does not represent the genotype as strings of symbols. Programs are represented in GP by means of trees, to minimize the possibility of generating syntactic mistakes. New versions of the genetic operators are defined accordingly. Koza has successfully applied his approach to a wide range of problems.

1.4. Evolutionary Strategies (ES) and Evolutionary Programming (EP)

ES are used to evolve populations of fixed length real vectors. ES offer a general-purpose search technique for any domain that can be modeled in this way. New definitions of the genetic operators are needed. For example, mutations are conceived as perturbations that satisfy several statistical conditions, crossover can be implemented as the average of the set of parents, etc.

EP is very similar to ES. One of the main differences is that EP does not use the crossover operator.

1.5. Variants of the classical GA/GP scheme

Standard GA's can be enriched by a number of mechanisms identified in biology and ecology. For instance, coevolution is a phenomenon present in natural evolution. Different species coexist and interact in the same habitat (such as herbivores and carnivores). If a species improves develops an improved feature to survive, (for example, if the prey population develops new techniques to escape from their predators by running faster, better camouflage, etc.), the other species must improve itself (in our example, the carnivores will have no alternative to develop better attacking strategies such as stronger claws, better eye sight, etc. [Par97]).

Coevolving GA's have been successfully applied to various problems, such as: constraint satisfaction problems [Par94b], evolving neural nets for classification

[Par94a], process control [Par98], path planning [Par97a], and the evolution of cellular automata (CA) for density classification[Par97b].

GA's are also subject to a continual revision by the researchers. In the last years [Gold02] a big amount of work is being made about the concept of competent GA's. This family of GA's offers a general purpose black box GA tool, ready to use without any modification, whatever the domain in which it will be applied. Deceptive problems are a family of tasks particularly difficult to solve by GA's. Competent GA's must be able to solve this kind of problems. Gene linkage (the relationship between gene loci and their functionality) and parameter tuning are some of the problems that competent GA's must solve. Goldberg's Messy GA's [Gold02] is one of the most promising approaches to competent GA's.

1.6. Grammatical evolution (GE)

Grammatical evolution is a grammar based variable-length binary string genome system, originally applied in the area of automatic programming (as a variant of GP) to generate programs or expressions in arbitrary programming languages to solve particular problems[Neil03][Neil01][Neil99a][Neil99b][Neil99c][Ray98a][Ray98b][Ray98c][Ray98d].

GE reinforces the biological inspiration by adopting a genotype-phenotype distinction and introducing a genotype-phenotype mapping that is directed by the grammar of a given programming language. The grammar is a plug-in component of the system that determines the syntax and the language of the output code. So, it is possible to evolve programs in an arbitrary language simply by plugging-in the corresponding grammar. As a result of this approach, the genotype representation and the genotype-phenotype mapping become standardized, and the evolutionary engine remains independent of the target programming language.

1.6.1. Genotype-Phenotype mapping

GE genotype-phenotype mapping is inspired in the biological process that produces phenotypic effects in the final individual (traits such as hair color, for example) starting from its genetic material, encoded in the organism's DNA. These molecules are first translated into mRNA (messenger RNA), which directs the generation of proteins corresponding to the sequences of the bases, which code for amino-acids. Proteins are fundamental components in this process, because they have functional meaning for the organism at the phenotype level.

In a GE system, the genotype representation (strings of bits read as strings of integers) has the role of the genetic material (DNA/RNA), the grammar of the target programming language acts as the rules to obtain the amino-acids from the base sequences in the genetic material, the programs obtained by means of GE are equivalent to proteins, and the result of their execution is similar to their phenotypic effect in the biological model.

In typical GE applications, the genotype is used to map the axiom of the grammar onto a word of its language by iterating the following steps:

1. Read the next 8 bits and compute the corresponding "codon" integer value.
2. Select the next non-terminal symbol (the leftmost non-terminal symbol) in the current sentential form (derived from the axiom).
3. Select a production rule to be applied, by means of the following computation: the rule number is the remainder of the division of the codon integer value by the number of different production rules for the current non-terminal. 0 is always the first rule number for every non-terminal.
4. Apply the selected rule to the corresponding non-terminal symbol, to get a new derived sentential form.

The process iterates until one of the following conditions holds:

1. A complete program is generated, that is, the derivation process produces a sentence in the language of the grammar (a word made exclusively of terminal symbols).
2. The end of the genome is reached.

GE may apply a wrapping mechanism to reuse the genetic material if the latter condition holds, in which case the algorithm will stop when a predefined threshold on the number of wrappings is reached. The wrapping mechanism is inspired by the gene-overlapping phenomenon present in many organisms.

Example

Let us use the following context free grammar:

$$\{N_R, T_R, S_R, P_R\}.$$

Where

- N_R is the set of non terminal symbols:

$$N_R = \{ \langle \text{expr} \rangle, \langle \text{op} \rangle, \langle \text{pre-op} \rangle, \langle \text{func} \rangle, \langle \text{header} \rangle, \langle \text{body} \rangle, \langle \text{declarations} \rangle, \langle \text{code} \rangle, \langle \text{return} \rangle \}.$$

- T_R is the set of terminal symbols:

$$T_R = \{ \text{sin}, +, -, *, x, 1.0, (,), \text{float}, \text{symb}, \{, \}, ;, a, = \}$$

- $S_R = \langle \text{expr} \rangle$ is the axiom.
- P_R is the set of production rules.

$$P_R = \{ \langle \text{expr} \rangle ::= \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \mid (\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle) \mid \langle \text{pre-op} \rangle \langle \text{expr} \rangle \mid \langle \text{pre-op} \rangle (\langle \text{expr} \rangle) \mid \langle \text{var} \rangle,$$

```

<op> ::= + | - | * | / | ,
<pre-op> ::= sin,
<var> ::= 1.0 | X |,
<func> ::= <header>,
<header> ::= float symb (float x){<body>},
<body> ::= <declaration><code><return>,
<declarations> ::= float a; ,
<code> ::= a=<expr>; ,
<return> ::= return(a);
}

```

Notice that:

- The language generated by this grammar is a subset of C functions with a float result and a float parameter. All of them have the same structure:

```

float symb (float x)
{
    a= <expr>;
    return(a);
}

```

- That is, genotypes only differ in the expression assigned to the variable a.

Therefore, the grammar can be reduced to the following subset of the production rules, where we have numbered independently the rules that share the same left-side symbol:

```

{ <expr> ::= <expr><op><expr>           (0)
  | (<expr><op><expr>)                 (1)
  | <pre-op>(<expr>)                 (2)
  | <var>                             (3),
  <var> ::= x                         (0)
  | 1.0                               (1),
  <pre-op> ::= sin                     (0),
  <op> ::= +                           (0)
  | -                                 (1)
  | /                                 (2)
  | *                                 (3)
}

```

Let us map the following genotype:

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

The genotype-phenotype mapping starts with the first codon and follows these steps:

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

As explained before, we start on the axiom. The initial sentential form is thus

$\langle \text{expr} \rangle$

We must find the leftmost non-terminal symbol, in this case the only one, $\langle \text{expr} \rangle$.

- $\langle \text{expr} \rangle$ is the left hand side of 4 rules, so the rule to be applied is number $220 \bmod 4 = 0$ (that is, $\langle \text{expr} \rangle$ becomes $\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$), and the next codon will be considered.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

The second sentential form is:

$\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- The highlighted $\langle \text{expr} \rangle$ is the next non-terminal symbol to be considered. So the rule number $240 \bmod 4 = 0$ is applied again and the next codon is read.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

The third sentential form is:

$\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- As in the two previous steps, rule (0) $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$ is used again to replace the highlighted $\langle \text{expr} \rangle$

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

The fourth sentential form is:

$\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- At this point, rule $\langle \text{expr} \rangle ::= \langle \text{var} \rangle$ is used, because $203 \bmod 4 = 3$.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

The fifth sentential form is:

$\langle \text{var} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- $\langle \text{var} \rangle$ is the leftmost non-terminal symbol. $\langle \text{var} \rangle$ is at the left hand side of two rules, and $101 \bmod 2 = 1$. Rule number 1 ($\langle \text{var} \rangle ::= 1.0$) will be applied.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

The sixth sentential form is:

1.0 $\langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- $\langle \text{op} \rangle$ is now the leftmost non-terminal symbol. $\langle \text{op} \rangle$ is the left hand side of four rules, and $53 \bmod 4 = 1$. Rule number 1 ($\langle \text{op} \rangle ::= -$) is applied.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

The seventh sentential form is:

1.0- $\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- $202 \bmod 4 = 2$, thus $\langle \text{expr} \rangle$ is replaced by $\langle \text{pre-op} \rangle (\langle \text{expr} \rangle)$.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0- $\langle \text{pre-op} \rangle (\langle \text{expr} \rangle) \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- $\langle \text{pre-op} \rangle$ is associated to only one right hand side (sin), so this derivation does not consume any codon.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin($\langle \text{expr} \rangle$) $\langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- $\langle \text{expr} \rangle$ is now replaced by the right hand side of rule with number 3 ($\langle \text{var} \rangle$), because $203 \bmod 4 = 3$.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin($\langle \text{var} \rangle$) $\langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

- $\langle \text{var} \rangle$ will be replaced by its rule (0), " $\langle \text{var} \rangle ::= x$ ", because $102 \bmod 2 = 0$.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin(x)<op><expr><op><expr>

- <op> will be replaced by its fourth right hand side (*) because $55 \bmod 4 = 3$.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin(x)*<expr><op><expr>

- This time, rule number 0 (<expr>::=<expr><op><expr>) is applied to <expr>, because $220 \bmod 4 = 0$

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin(x)*<expr><op><expr><op><expr>

- <expr> is now replaced by its rule number 2 (<expr>::=<pre-op>(<expr>)), because $202 \bmod 4 = 2$

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin(x)*<pre-op>(<expr>)<op><expr><op><expr>

- As previously explained, <pre-op> will always be replaced by *sin* without consuming any codon.

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin(x)*sin(<expr>)<op><expr><op><expr>

- Now 243 is consumed to replace <expr> by <var>, because $243 \bmod 4 = 3$

220	240	220	203	101	53	202	203	102	55	220	202	243	130	37	202	203	140	39	202	203	102
-----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----	-----	----	-----	-----	-----	----	-----	-----	-----

1.0-sin(x)*sin(<var>)<op><expr><op><expr>

- $130 \bmod 2 = 0$, so <var> is replaced by x, which corresponds to applying rule number 0.

The interested patient reader can easily follow the algorithm in the same way and get the sequence of expressions that follows:

1-sin(x)*sin(x)<op><expr><op><expr>

$37 \bmod 4 = 1$

$1-\sin(x)*\sin(x)-\langle\text{expr}\rangle\langle\text{op}\rangle\langle\text{expr}\rangle$	$202 \bmod 4 = 2$
$1-\sin(x)*\sin(x)-\langle\text{pre-op}\rangle(\langle\text{expr}\rangle)\langle\text{op}\rangle\langle\text{expr}\rangle$	
$1-\sin(x)*\sin(x)-\sin(\langle\text{expr}\rangle)\langle\text{op}\rangle\langle\text{expr}\rangle$	$203 \bmod 4 = 3$
$1-\sin(x)*\sin(x)-\sin(\langle\text{var}\rangle)\langle\text{op}\rangle\langle\text{expr}\rangle$	$140 \bmod 2 = 0$
$1-\sin(x)*\sin(x)-\sin(x)\langle\text{op}\rangle\langle\text{expr}\rangle$	$39 \bmod 4 = 3$
$1-\sin(x)*\sin(x)-\sin(x)*\langle\text{expr}\rangle$	$202 \bmod 4 = 2$
$1-\sin(x)*\sin(x)-\sin(x)*\langle\text{pre-op}\rangle(\langle\text{expr}\rangle)$	
$1-\sin(x)*\sin(x)-\sin(x)*\sin(\langle\text{expr}\rangle)$	$203 \bmod 4 = 3$
$1-\sin(x)*\sin(x)-\sin(x)*\sin(\langle\text{var}\rangle)$	$102 \bmod 2 = 0$
$1-\sin(x)*\sin(x)-\sin(x)*\sin(x)$	

Once a correct phenotype is produced (a correct sentence in the language of the grammar), the mapping finishes.

Applications

The authors of GE have applied their approach to automatically program solutions for a wide range of problem and domains:

- Symbolic regression and integration
- The Santa Fe ant trail
- Caching algorithms

They have also tested the performance of GE as compared with other classical GP approaches [Neil03].

Chapter 2

Fractals

2.1. Overview

Euclides (430-360 BCE) is considered the father of Classical Geometry, which studies ordinary and (in some way) "régular" shapes (points, lines, surfaces and volumes, such as ellipses, circles, polygons, polyhedrons, etc.). One of the main concepts in Geometry is dimension, which is associated to the number of degrees of freedom for each object, that is, the number of parameters needed to define each point in the object. So, points have 0 dimension, lines 1, surfaces 2, volumes 3, and so on.

These dimensions were the only way to describe shapes for many centuries; objects too complex (most of the real objects: clouds, mountains, coastlines, etc., plus a few well known monstrous mathematical objects as, for instance, a curve defined by Peano able to fill a surface) remain out of the scope of Classical Geometry.

The history of fractals began with mathematicians' attempts to study such strange objects. Problems apparently trivial did not have a clear answer in Classical Geometry. For example: the lengths of coastlines or of the borders between countries depend on the method used to compute them. The underlying difficulty is the fact that these curves (if coastlines and borders can be considered curves) have so much complexity, that it is always possible to find more detail, and hence a greater length, while zooming in. Classical dimensions have no meaning in these cases, so Mandelbrot [Man75] suggested using real positive values, rather than natural numbers, to express the dimension of such objects. Mandelbrot also considered a wide diversity of strange phenomena and coined the term *fractal* to describe them, because they share a few (not necessarily all) curious properties, such as **self-similarity** (the same shapes are found at different levels with different scales all over the set), **underivability** at every point, **infinite length** covered in a finite space, **difficulties to calculate their dimension**, etc. In the next sections, different types of fractals will be introduced, although only one of them (initiator-iterator fractals) is considered in depth in the present thesis.

This chapter ends with an overview of several applications of fractals, and a brief definition of multi-fractals.

2.2. Categorizing Fractals

Most fractals have been represented by means of the following methods:

2.2.1. Initiator-Iterator Fractals

This kind of fractals is defined by means of two shapes, the initiator and the iterator. The fractal is defined as the limit curve after an infinite number of transformations from the initiator. Transformations are obtained by applying the iterator to each segment of the initiator in the curve at the previous stage. Many famous monstrous curves can be constructed by means of this method, such as Cantor set, von Koch snowflake, Peano's curve, and Sierpinski's gasket.

Cantor set

The initiator is a single segment. The iterator will erase a segment of length $1/3$ from the central part of each segment in the previous step. After applying the process an infinite number of times, the number of segments increases to infinite and the length of the segments decreases to 0. Figure 2-1 illustrates the first 4 steps in this process.

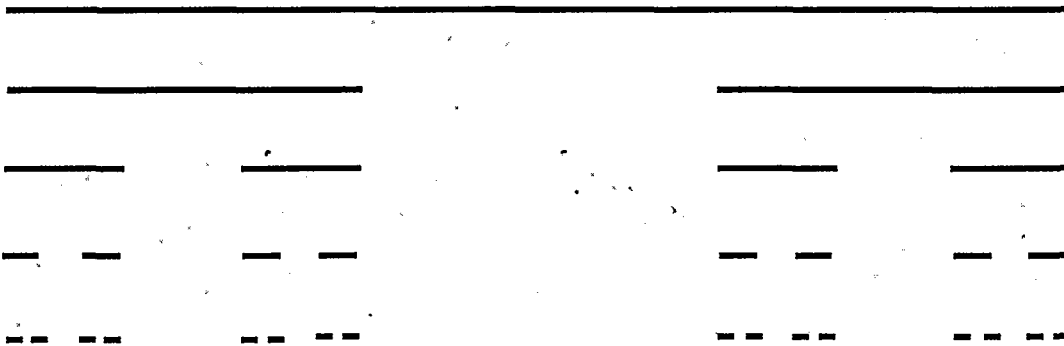


Figure 2-1: The first four steps in Cantor set construction

The reader may infer from the definition that the set contains an infinite number of segments, but the length of the complete set is 0, because each segment has length 0 and an infinite sum of 0's equals 0. It is obvious that Cantor set is a very strange one.

Von Koch snowflake

In this case, the iterator replaces the middle part of the initiator shape by the remaining two sides of an equilateral triangle.

Figure 2-2 illustrates the first five steps to build Von Koch snowflake. Two odd properties of this curve are that it has no tangent at any point (when the length of each segment goes to 0, every point is a vertex of some equilateral triangle), and that it represents a walk of infinite length comprised in a finite space. To explain this last property, let us study the length of segments at step N , and the total length. They are respectively equal to $(1/3)^n$ and the total length is equal to $(4/3)^n$. Table 2-1 ([Fla98]) illustrates the relation between the lengths of the segments and the total curve length.

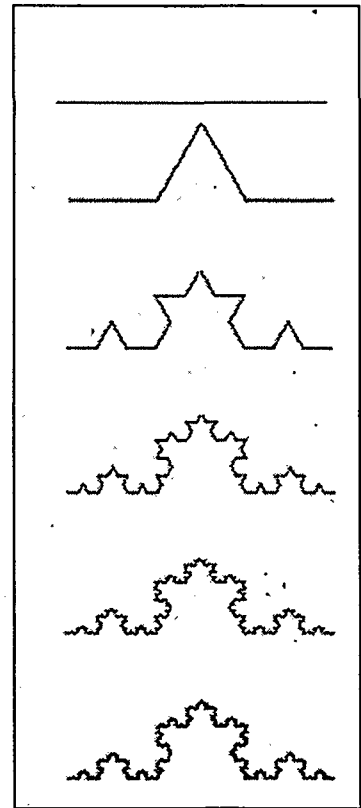


Figure 2-2: Von Koch's curve

Step	Number of segments	Length of a segment	Total length
0	1	1	1
1	4	0.33333	1.333333
2	16	0.11111	1.7777778
3	64	0.037037037	2.37037
4	256	0.0123457	3.16049
5	1024	0.00411523	4.21399
6	4096	0.00137174	5.61866
7	16384	0.000457247	7.49154
8	65536	0.000152416	9.98872
9	262144	5.08053×10^{-5}	13.3183
.	.	.	.
.	.	.	.
.	.	.	.
100	1.60694×10^{60}	1.94033×10^{-48}	3.11798×10^{12}

Table 2-1 The length of Von Koch curve.

Peano's curve

In 1890, Giuseppe Peano defined a curve that can be represented by means of an iterator that divides the segment into thirds and replaces the central part of the segment with 7 segments of length $1/3$, forming two squares around the removed middle third part. Peano's curve is continuous, that is, it is drawn without lifting the pen of the paper. Figure 5.3 shows its iterator.

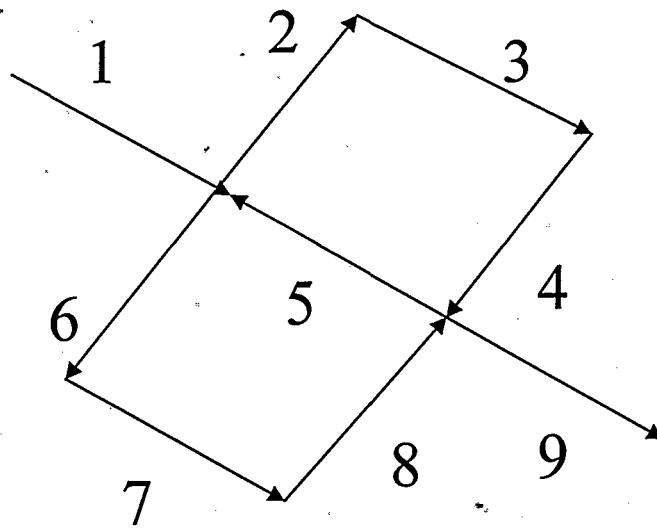


Figure 2-3: The iterator of Peano's curve

Peano's curve completely fills up the unit square that contains its initiator, and has also, like von Koch's snowflake an infinite total length

2.2.2. Random Fractals

Self-similarity may be regular or random. In the preceding examples, the generated fractals are very regular. This kind of regularity is not frequent in natural shapes that rather exhibit, to some extent, a random self-similarity.

This is the case of a plethora of phenomena from bacteria colonies to clusters of galaxies. An example of a random fractal, called a Brownian-motion, fractal was used to describe the motion of a particle in a gas or a liquid. It is defined as a random walk in the Cartesian plane in the following way: starting at some initial point, just move to any other point in the Cartesian plane with equal probability. An example can be seen in figure 2-4. It shows clearly its scale-independent self-similarity.

Figure 2-5 shows other similar phenomenon: bacteria aggregation.

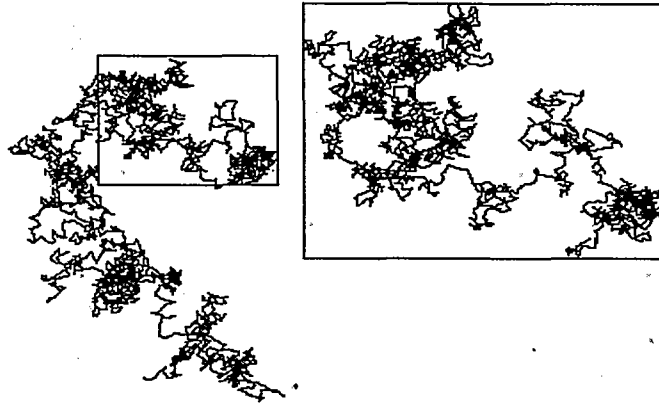


Figure 2-4: Example of a Brownian-motion fractal, from [Int].

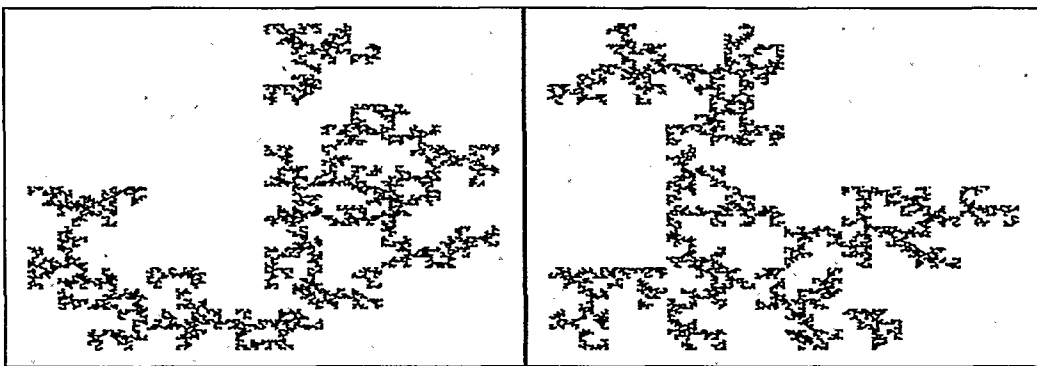


Figure 2-5: Illustrates an example of a bacteria aggregation fractal [Int].

2.2.3. Other ways of defining fractals: affine transformation fractals and iterated function systems

Affine transformation fractals can be constructed by iterating a fixed original pattern with different scaling factors. The copies of the pattern must remain in the same position with respect to the other generated copies, at each iteration.

Several algorithms can be used to generate fractals in this way. One of them is MRCM (Multiple Reduction Copy Machine), where a regular self-similar recursive function is used. The combination of geometric transformations (translation, rotation, scale factors) applied in MRCM is the same through all the process. The result of the algorithm is obtained when the number of iterations goes to infinite.



Iterated Function Systems (IFS) is a method similar to MRCM. Instead of iterating the complete initial shape, only a subset of points chosen at random is handled.

2.2.4. Iterative Dynamical System Fractals.

Some Fractals appear as the boundary between convergence and divergence of certain recursive mathematical functions in the complex domain. The most famous fractals of this kind are Mandelbrot and Julia sets.

Mandelbrot set is the boundary between the convergence and divergence domains of the recursive complex function:

$$z_{n+1}=f(z_n)=z_n^2+c,$$

where $c=c_x+c_y i$ is taken from a ball in the complex plane and $z_0=0$.

Julia set is the boundary between the convergence and divergence domains of the recursive complex function:

$$z_{n+1}=f(z_n)=z_n^2+c,$$

where $c=c_x+c_y i$ is a fixed complex point and z_0 is taken from a ball in the complex plane. [Man77].

2.3. How can we estimate the dimension of fractals

The concept of dimension is very old and seems easy and evident. We live in a space with three dimensions: length, width, and depth. Some of the objects in our environment are approximately bi-dimensional: a sheet of paper, a picture. Others have a single prevalent dimension: a distant road, a pencil line drawn on paper. What we call *dimension* may sometimes be defined as the number of directions in which movement is allowed.

Things appear very clear and elegant: dimensions are consecutive integers: 0 (a point), 1 (a line), 2 (a surface), 3 (a volume), with no doubtful cases. Some do exist, however, as Mandelbrot proved in his famous book on fractals [Man77]. Depending on the size of the observer, a ball of thread can be considered as

- A point (zero dimensions) if the observer is very large (a mountain, a planet) or very far away.
- A sphere (three dimensions) if the observer is comparable to the size of the ball (a human being) and is located near the ball.
- A twisted line (one dimension) if the observer is smaller than the ball (an ant) and very near it.

- A twisted cylinder (three dimensions) if the observer is much smaller than the ball (a bacterium).
- A set of isolated points (zero dimensions) if the observer is even smaller and can see the atoms.
- A set of spheres (three dimensions), if the observer's size is comparable to that of the atoms.
- And so forth.

Therefore, it seems necessary to revisit the definition of dimension. In the following sections several approaches to estimate fractal dimension are briefly described.

2.3.1. Hausdorff dimension

In 1919, Hausdorff proposed a new definition of dimension, applicable to such doubtful cases, to distinguish them from normal surfaces and lines. With his definition, strange curves in the plane may have a fractional dimension between 1 and 2. For instance, Peano's curve, which covers a complete square, has a Hausdorff dimension of 2, while Von Koch snowflake has a Hausdorff dimension of about 1.261.

The Hausdorff dimension of a curve is considered very difficult to compute and is not practical for calculation, because of its abstract characteristics.

Benoit Mandelbrot, who coined the term *fractal* in 1975, gave the first definition of a fractal based of the Hausdorff dimension, as mentioned before:

Definition: *A fractal is a set for which the Hausdorff dimension strictly exceeds the topological dimension.*

2.3.2. Richardson-Mandelbrot dimension estimation

Richardson found a linear relation between the logarithm of the measured length and the logarithm of the unit used for measuring it, and considered the slope of this line as a good indicator of the curve. Mandelbrot associated that slope to the concept of fractal dimension.

This method computes the fractal dimension of a curve as a quotient of two measurements taken while "walking" the fractal line in a number of discrete steps. We take as a unit the distance between the beginning and the end of the fractal line to be walked. The first measurement is P_1 ; the length of the step used, or pitch length, which must be constant during the whole walk. The second is the number of steps needed to reach the end of the walk by following the fractal curve, $N(P_1)$.

We call D_{p_1} the number for which the following relation holds:

$$N(P_1) \approx P_1^{-D_{P_1}}$$

If we take logarithms on both sides of this equation, we obtain

$$\log [N(P_1)] = -D_{P_1} \log(P_1)$$

The fractal dimension is the limit of D_{P_1} when P_1 goes to zero:

$$D_{P_1} = \lim_{P_1 \rightarrow 0} \left(\frac{-\log[N(P_1)]}{\log(P_1)} \right)$$

2.3.3. Box Counting

This method computes the fractal dimension of a curve as a quotient of two measurements taken while covering the fractal curve with a set of boxes. $N(d)$ denotes the number of boxes with size of length d which are necessary to cover the curve, considered as a set of points in the two-dimensional plane. Box dimension is defined as the exponent D_b in the equation

$$N(d) \approx d^{-D_b}$$

If we take logarithms on both sides of the equation, we obtain

$$\log[N(d)] = -D_b \log(d)$$

The fractal dimension is defined as the limit of D_b when d goes to zero

$$D_b = \lim_{d \rightarrow 0} \left(\frac{-\log[N(d)]}{\log(d)} \right)$$

There are many variations to this algorithm. Some of them assign a weight to each box, depending on the number of points it contains. Instead of computing the number of boxes, another variation estimates the information entropy for the set of boxes, where the number of points is considered to be the information.

Alternative ways of calculating Box dimension, change the shape and the nature of the set of boxes. Square-shaped boxes are usually used to define a grid on the image; other approaches place the boxes at any position and orientation. Families of concentric circular boxes with increasing radius are also used.

With this definition, Mandelbrot associated a fractal dimension to strange objects such as Cantor set, Koch snowflake and Peano's curve: Cantor set has a fractal dimension equal to $\log(2)/\log(3) \approx 0.63093$, which seems intuitively correct, because it is a set of isolated points that we cannot consider either as a curve or a point. Von Koch snowflake has a Richardson-Mandelbrot dimension of $\log(4)/\log(3) = 1.26186$, which

means that its behavior is more complex than that of a curve, but doesn't reach to fill a two-dimensional surface. It is also clear that the total length of von Koch's curve is infinite. Peano's curve has a Richardson-Mandelbrot dimension equal to 2, which seems intuitively correct, because, when the number of iterations goes to infinite, it completely fills the bi-dimensional region that includes its initiator.

2.4. Multi-fractals

In natural phenomena with a fractal structure, it usually happens that there is not a uniform fractal dimension applicable all over the set. This means that different parts of the fractal will have different dimensions (different degrees of complexity). Therefore, while the global fractal dimension represents the global complexity of the whole object, different parts of the object may have different dimensions.

Multi-fractal structures are found in various contexts, where they are usually represented by multiplicative cascades of random processes.

2.5. Application of Fractals

This section briefly introduces some of the areas where fractals have been used, such as Physics, Chemistry, Astronomy, Geology, image compression, Psychology, Economics, medical imaging, et cetera.

Many natural phenomena are better described using fractional dimensions. Fractals are thus used as descriptive models for the growth of plants, particle aggregation, river cartography, realistic images, and similar phenomena. Fractal dimension characterizes most of the properties of these models.

In image processing, fractals are considered useful approaches to analyze and quantify the complexity of images. The fractal dimension of an image is a parameter that makes it possible to study the roughness or the smoothness of digitized images. For example, for 2-D images, the fractal dimension should lie between 2 and 3. A fractal dimension closer to 2 represents smooth images. A 3-D representation of a 2-D digitized image is possible by associating to the third dimension the intensity of each pixel in the 2-D image [Dat02].

Fractals have also been used for medical imaging, to analyze X-ray medical images, or to check the quality of ultrasonic C-scan images of glass-epoxy and carbon-epoxy composite laminates containing flaws. Fractal dimension has been used to characterize mammographic patterns [Li97], trabecular bones [Maj99], etcetera.

The physical characteristics of some bodies are related to the fractal dimension of their surfaces [Mel01]: the growth pattern of bacteria has a fractal dimension of 1.7. Another example is geological patterns: the fractal dimension of clouds is 1.30-1.33; 1.7 for snowflakes; 1.05-1.25 for coastlines in South Africa or Britain; 1.28-1.90 for woody plants and trees; et cetera. [Tay02].

In medicine, fractal dimensions have been found for various bio-molecules, such as DNA and proteins. For instance, the fractal dimension of Lysozyme (egg-white) is 1.614, for hemoglobin it is 1.583, for myoglobin 1.728 [Ian96]. The fractal dimension of the perimeter of surface cell sections has been used to distinguish healthy cells and cancerous cells [Bau99]. In analytical chemistry, the fractal dimension is used as a tool to characterize chemical patterns and problems of sample homogeneity [Dan02]. A given fractal dimension makes it possible to simulate a variety of systems: fluid extraction or contaminant mitigation techniques [Mel01], the hybrid orbital model of proteins [Tor01], or the growth of conflict rate in aircraft flays [Mon01].

Antennae are electromagnetic devices designed to radiate or capture signals. Some of their characteristics are gain, bandwidth, return loss and resonant frequencies. In the last years, fractal geometry has provided a new approach to traditional antenna design methods [Vin01]. Several classical fractals of the initiator-iterator kind (von Koch Snowflake, Sierpinski's gasket, for example) have been proposed as antenna prototypes. Certain properties of fractal antennae are related to their fractal dimension: an increase in the fractal dimension may be translated into higher gain, low return loss and a shifting down of the resonant frequencies.

Multi-fractals have been applied to different areas, such as signal processing [Rel02], earthquake distribution analysis [Har01] and network data traffic modeling [Rie99] [Rib01] [Abr02] [Sar01].

Chapter 3

Lindenmayer Systems (L-Systems)

3.1. Overview

Lindenmayer systems (L-systems) were originally created by Aristid Lindenmayer [Lin68] to study formal languages. They were then used as formal discrete models of plant development and other biological development systems of different multi-cellular organisms. Lindenmayer defined a new type of grammar (a parallel derivation grammar), which differs from the normal Chomsky grammars (sequential derivation grammars) because the rules are applied simultaneously, rather than one at a time.

An L-system consists of an *alphabet* (a set of symbols), an initial string called the *axiom* and rewriting rules called *production rules*. The production rules are applied recursively; in the first iteration they are applied to the axiom.

L-systems have many interesting properties such as their simplicity, variety, modularity, and universality [Goe91]. They have been successfully applied to the simulation of biologic processes such as plant growth, leaf development, pigmentation of snail shells, and several others [Pru90][Pru94][Pru98]. Recently, many investigations of artificial life are trying to use L-system to simulate and generate realistic images that model plants, simulate branching and flowering patterns, or study the influence of the environment (light, nutrients and mechanical obstacles) on a developing plant, as we can see in figures 3-1 to 3-3. Discrete mathematical models and symbols based on formal languages and abstract machines have proved to be a useful tool for such applications.



Figure 3-1: Development process of an artificial tree using a special type of L-systems called stochastic L-systems (from [Pru94]).

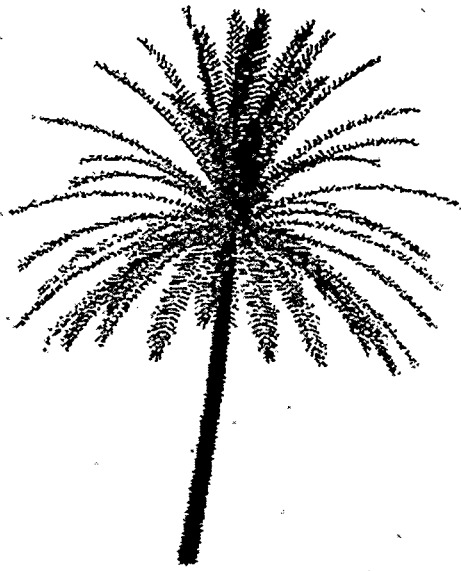


Figure 3-2: A model of a date palm tree simulated by an L-system variant (from [Pru95]).



Figure 3-3: Found in [Pru95], illustrates the development of a plant

Lindenmayer systems are also appropriate to represent fractal objects, including not only *iterated geometrical transformations* (as Cantor dust, the Sierpinsky gasket or the Von Koch snow flake), but also fractal sets found in the complex plane, as Mandelbrot or Julia sets [Ort02]. Lindenmayer systems can also represent the evolution of discrete mathematical models such as Von Neumann auto-reproductive cellular automata, and McCulloch and Pitts neuronal models.

3.2. Classes of L-Systems

Lindenmayer derivation grammars, also called L systems, can be classified in different ways. In the next sections, some of these classes will be defined, specially those that will be used later in this thesis.

3.2.1. 0L Systems

0L systems are context-free L-systems; that is, each symbol is replaced by the same right hand side of the rule wherever it is found. The transformation of a symbol does not depend on the symbols close to it. As there is no interaction between a symbol and its neighbors in the derived strings, 0L systems are also called L-systems without interactions.

D0L Systems

D0L systems are 0L systems where two different rules may not have the same left-hand side symbol. These systems are also called deterministic L-system without interactions.

A D0L system is a three-fold with an alphabet (a finite non-empty set of symbols), a set of production rules that determines the only way each symbol of the alphabet can be changed in a derivation, and a starting word or axiom. A derivation of a word in a D0L system is the new word obtained when each symbol in the word is replaced by applying the allowed transformations.

Formal definition

A D0L system is the three-fold

$$S = (\Sigma, P, \omega)$$

where:

$\Sigma \neq \Phi$ is a non-empty set of symbols, the alphabet.

$P \subseteq \Sigma \times \Sigma^+ \mid \forall a \in \Sigma \Rightarrow \exists! \alpha \in \Sigma^+, a ::= \alpha \in P$, is the set of production rules.

$\omega = \omega_0 \dots \omega_{|\omega|-1} \in \Sigma^+$, is the axiom and $|\omega|$ is the length of the word ω (the number of its symbols).

The following expression

$$x \Rightarrow_S y$$

means that word y is derived from word x by means of the production rules of system S . When it is clear which system is used, the following expression indicates the same fact:

$$x \Rightarrow y$$

The set of words that can be derived from the axiom is called the language of a DOL system.

Example 1

Let S be the following DOL system

$$S = \{\Sigma = \{A, B\}, P = \{A ::= B, B ::= AB\}, A\}$$

We can get the following derivations from the axiom:

Stage 0: A
 Stage 1: B
 Stage 2: AB
 Stage 3: BAB
 Stage 4: ABBAB
 Stage 5: BABABBAB
 Stage 6: ABBABBABABBAB
 Stage 7: BABABBABABBABBABABBAB

If we count the length of each string, we obtain the Fibonacci sequence of numbers:

1 1 2 3 5 8 13 21 34 55 89

Consider the DOL system $G = \{\Sigma, h, \omega\}$. We define h^n as the number of times the production rules have been applied to the axiom ω , as follows:

$$h(h(h\dots h(\omega)\dots))$$

The growing function of G (denoted $f_G(n)$) is defined as follows:

$$f_G: \mathbb{N} \rightarrow \mathbb{N}, f_G(n) \rightarrow |h^n(\omega)|$$

where the growing sequence of the G system is:

$$\{|h^n(\omega)|\}, n \in \mathbb{N}$$

Example 2

Let $G_{\text{Fibonacci}} = \{\{a, b\}, \{a ::= b, b ::= ab\}, a\}$. Table 3-1 shows the first five derivation steps, and the first five elements in the growing sequence.

n	$h^n(\omega)$	$ h^n(\omega) $
0	a	1
1	b	1
2	ab	2
3	bab	3
4	abbab	5
5	bababbab	8

Table 3-1: Derivation steps that generate the Fibonacci sequence

P0L and PD0L Systems

If no rule in a 0L-system has the empty word (λ) in its right hand side, the system is called a propagative L-system or P0L.

An L-system can be at the same time deterministic and propagative. These systems are called PD0L.

3.2.2. $\langle k, l \rangle$ IL Systems

In the same way that Chomsky context-free grammars can be extended to context-sensitive grammars, 0L-systems can be extended to L-systems with interaction, or IL Systems, i.e., the left-hand side of the rules can contain strings instead of single symbols. These strings indicate the symbol that will be transformed and the context in which the rule can be applied. The size of the context must be the same in every rule. This may be problematic if a symbol is too close to the beginning or the end of the string. In these cases, to solve this problem, a new symbol will be used to fill the context. Such a symbol can only appear at the beginning or the end of the string.

These systems are called Lindenmayer system with $\langle k, l \rangle$ interactions, where k is the size of the left-hand side context, and l is the size of the right-hand side context. k and l cannot be negative values.

Formally

A $\langle k, l \rangle$ IL system is a four-fold (Σ, P, g, ω) where:

Σ is the alphabet.

ω the axiom.

P the set of production rules.

$g \notin \Sigma$ is the symbol added to the context outside the word boundaries,

such that

1. If $\omega_1 a \omega_3 ::= \omega_4 \in P$ then if

- $w_1 = \bar{w}_1 g \bar{w}_1$ for some $\bar{w}_1, \bar{w} \in (\Sigma \cup \{g\})^*$, then $\bar{w}_1 \in \{g\}^*$
- $w_3 = \bar{w}_3 g \bar{w}_3$ for some $\bar{w}_3, \bar{w}_3 \in (\Sigma \cup \{g\})^*$, then $\bar{w}_3 \in \{g\}^*$

2. For every $w_1 a w_3 ::= w_4 \in (\Sigma \cup \{g\})^k \times (\Sigma \cup \{g\})^l$ such that w_1, w_3 satisfy the two conditions in point 1, there exists $w_4 \in P$

3.2.3. Systems with tables

To simulate some kind of processes, it may be necessary to consider more than one set of production rules, to be applied in different circumstances. These systems are *L-systems with tables* or *TL systems*.

Formally

A TOL system is a triple fold (Σ, \wp, ω) .

Where

Σ and ω are defined as previously.

\wp The set of the tables of the system, a finite non-empty set.

Each element P in \wp , called a table, is a finite non empty subset of $\Sigma \times \Sigma^*$ such that

$$\forall a \in \Sigma (\exists \alpha \in \Sigma^* | a ::= \alpha \in P).$$

3.2.4. Systems with extensions

To simulate some biological systems, it is convenient to specify a subset of the alphabet in the L-system, so that the language generated by the L-system only contains the words that are made of symbols in that subset. In this case, we say that the system has extensions. An EOL system is a system without interactions and with extensions. An EIL system is a system with interactions and with extensions. An ETOL system is a system without interactions, with tables and with extensions.

3.2.5. Other combinations

The previous definitions describe the most common families of L-systems. It is possible to have other combinations of these properties, for example, DIL systems, PDIL systems, DTL systems, etc.

3.2.6. Other L-System extensions

- **Bi-dimensional IL- systems**

In bi-dimensional IL systems, the words are matrices of characters, rather than linear strings. The context is determined by a function c that generates the horizontal and vertical displacements of the context symbols with respect to the current symbol.

Formally

A bi-dimensional $\langle k, 0 \rangle$ IL system is defined as the five-fold $\langle \Sigma, P, g, \omega, c \rangle$

where

Σ, P, g, ω were defined previously.

$c: [1, K] \cap \mathbb{N} \rightarrow \{-1, 0, 1\}$.

- **Probabilistic L- Systems**

A probabilistic L system differs from other L-systems because each production rule has an associated probability ($R, p(R)$), where R is the derivation rule and $p(R)$ its probability of being applied. The sum of the probabilities associated to all the rules applicable to a symbol must be 1.

- **Parametric L-Systems**

Real parameters or arguments can be associated to the symbols of the alphabet. For example, while in standard L-systems, the production rules have the form

$$A ::= BCD$$

in parametric L-Systems it is possible to associate an argument to each symbol, plus the conditions that argument must satisfy. For instance, if we have the following rule:

$$A(t): t > 5 ::= B(t+1)CD(t^{0.5}, t-2) \text{ it will be applied only when } t > 5.$$

3.3. L-System Design

The first step while designing an L-system for simulation (e.g. to construct a biological model) consists of understanding the specifications of the model and the way in which the Lindenmayer system should work. Once this is understood, the task for solving the concrete problem must be designed. There are two main methods to design such L-systems: *ad hoc* manual solutions and genetic algorithms.

3.3.1. Ad-hoc solution

In many previous works, L-systems were designed by hand to solve specific tasks, and then they were implemented to be run on a computer [Pru94][Pru95][Pru98][Ort02]. We can describe the ad-hoc solution as the following process:

- Reading thoroughly and understanding the way in which the defined Lindenmayer system works.
- Listing the required input-output.
- Writing the process in the form of an algorithm.
- Implementing the system with the appropriate axiom and production rules.
- Validating the system with different axioms and productions to identify its limitations.

The manual synthesis of L-systems modeling the morphogenesis of a particular biological species is a difficult task. For that reason, the use of evolutionary algorithms facilitates the L-system design. An example of manual L-system design to simulate an ecosystem, with appropriate definition of L-systems and their production rules, is described at [Ort00], which simulates a biological system where many different species exist and interact among them.

3.3.2. Genetic L-System programming

EAs have been used to evolve solutions to problems in many different areas, including L-system design problems. Previous work by other authors has applied genetic algorithms to L-systems. Many researches have designed L-systems that can simulate different kinds of biological models by means of EAs.

First, a random population of L-systems is generated. Next, an EA is applied for a specific number of generations to achieve the pre-defined goal or the proposed model. Ochoa [Och98] evolves D0L systems with a single rule that generate shapes similar to plants. Jacob [Jac94][Jac96a][Jac96b] used genetic programming techniques to find the proper axiom and the production rules that describe growth processes of plant structure, to model the growth of a real plant (see figure 3-4). Other authors [Hor03][Tra96] evolve parametric L systems (an extension of Lindenmayer grammars). In this thesis a different approach will be used, grammatical evolution, which provides a better parallel to biological evolution.

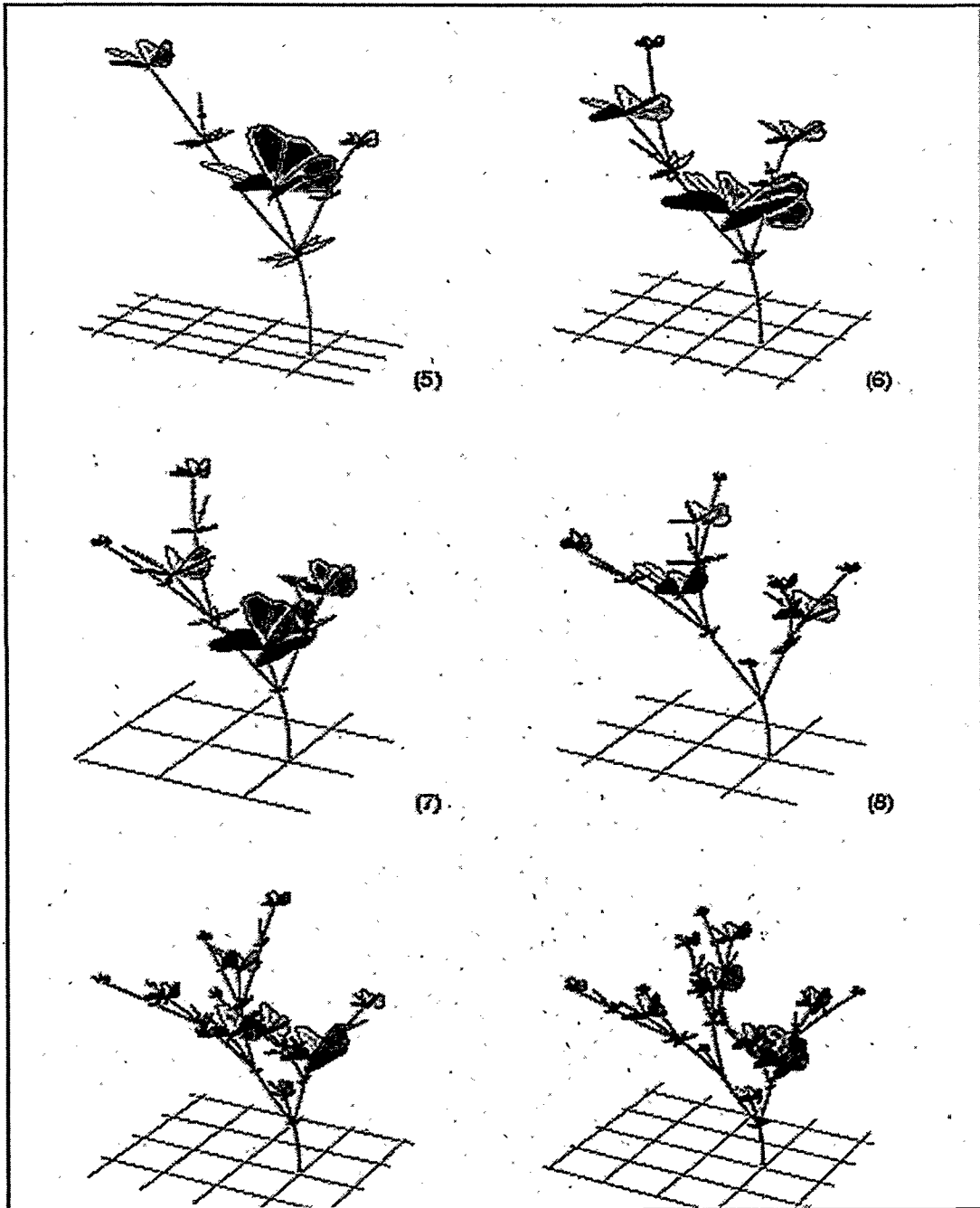


Figure 3-4: Plant growth simulation using Genetic Programming and L-systems, from [Jac94]



Chapter 4

Fractals and L-Systems

4.1. Overview

In previous chapters, different types of fractals were introduced. One of them is the initiator-iterated fractal. As we know, an L-system starts with the axiom and applies a set of production rules to generate derivations. Clearly we need a graphical interpretation of the words derived by the L-system to obtain a visible fractal object from them.

There exist two different kinds of graphical interpretations of L-systems; the most common one is called turtle graphics. Another method is known as vector graphical interpretation. In the following sections the two methods will be discussed. Then a method for calculating the fractal dimension of the generated fractal will be discussed.

4.2. Turtle Graphics Method

Turtle graphics were introduced by Papert at 1980 [Pap80]. He devised it to make computers and algorithms easier to understand for kids. In 1984, Smith [Smi84] used this method for the first time to give a graphical interpretation to L-system strings. His work encouraged other researchers to develop more applications related to various disciplines.

A turtle graphic is considered as the trail left by an invisible turtle that follows very precise but brief motions defined by its position and the direction of its movement. The turtle changes its position by moving forward a specific distance m , and its direction by rotating an angle α . The turtle graphic interpretation can be applied at different levels of complexity.

The turtle graphic interpretation (T, α) used in this thesis is one of the simplest versions, where the alphabet of the related L-system can be expressed as the union of four disjoint subsets:

$$\Sigma = N \cup D \cup M \cup \{), (, +, - \}$$

where

$A \in N$ Leaves the state of the turtle unchanged. N is called the set of non-graphic symbols.

$F \in D$ Moves the turtle one step forward, in the direction of its current angle, leaving a visible trail. D is called the set of draw symbols.

$f \in M$ Moves the turtle one step forward, in the direction of its current angle, with no visible trail. M is called the set of move symbols. Using move symbols, Cantor set fractals are easy to generate.

- (Pushes the turtle state in the stack.
-) Pops and restores the turtle state from the stack.
- + Rotates the direction the turtle is pointing by a positive angle α .
- Rotates the direction the turtle is pointing by a negative angle α .
- α The unit rotation angle, restricted to $2k\pi/n$ with k, n integers.

A given fractal can be represented (with an appropriate scale factor) by means of the four components of an L-system, a turtle interpretation, an angle step and a distance step.

4.3. Vector Graphics Method

In this family of graphical interpretations, each symbol in the alphabet of the L-system is associated to a vector in a rectangular Cartesian system. A word derived by the L-system can be graphically represented as a concatenation of the vectors associated to the symbols that make the word. As the turtle can leave (or not leave) a visible trail, the vector graphics interpretation will assign a visibility coefficient to each symbol in the L system alphabet. This makes it possible to represent non-connected curves. For the visibility coefficient, we will assume that 0 means invisible and 1 means visible.

Branching fractals can be obtained with a vector graphical interpretation, since it is possible to come back to the start point of the branching, by returning by the same way in the opposite direction. This can be achieved if, for every vector associated to a symbol, an opposite vector is associated to some other symbol.

Formally

$$V; \Sigma \rightarrow \{0,1\} \times \mathbb{R}^2$$

Where $\{0,1\}$ indicates the visibility or the invisibility of the vector, and \mathbb{R}^2 indicates the Cartesian X-Y System.

4.4 Equivalence between the Turtle and Vector graphical representations of L-systems

To represent complex fractals, non-standard extensions of L-systems were proposed by different authors. In a previous work by Alfonseca and Ortega [Alf97], DOL systems were used to provide a wide range of fractals using the two graphical interpretation methods mentioned above. In this approach, the DOL system is isolated from its graphical interpretation, i.e. the same L-system may generate completely different fractal curves by changing the graphical interpretation.

We call the pair $GL = (L, G)$ a Lindenmayer graphic, where L is a Lindenmayer system, and G is a graphical interpretation (using either turtle graphics or vector graphics). When the graphic interpretation is of turtle type, we call GL a Lindenmayer turtle graphic.

[Alf97] proved that both methods are equivalent, in the sense that a fractal that can be represented by an L system with one type of graphical interpretation, can also be represented by a (different) L system and the other type of graphical interpretation in a wide family of cases. This equivalence between the two methods has important advantages, as the turtle graphic interpretation is more flexible for actions such as area filling and coloring, while vector graphics are usually faster. Thus, it may be useful to be able to build a Lindenmayer turtle graphics from an equivalent vector graphics or vice versa.

We call *TGDOL* the set of all DOL systems that represent fractals by means of the turtle graphics interpretation (T, α) .

We call *VGDOL* the set of all DOL systems that represent fractals by means of a vector graphics interpretation.

For the two methods to be equivalent, a few restrictions must be taken into account:

First, the strings (rules) of the DOL system under the turtle graphics interpretation must be angle-invariant, which means that the direction of the turtle at the beginning and the end of the string should be the same. This restriction is not really important, as it can always be fulfilled by adding a certain number of + or - symbols to the end of the string.

We call *AITGDOL* to the set of all angle-invariant *TGDOL*-schemes, where all the right-hand sides of the rules in *TGDOL* are angle invariant strings.

In the same way, vector interpretation graphics must have some conditions and modifications to achieve the equivalence.

A *VGDOL* with a Vector Interpretation VI is called a rationally related DOL system (*RRVGDOL*) if both the set of modules and the set of angles of all the vectors in VI are rationally related. This means that, for any finite rationally related scheme with a vector graphics interpretation, there exist two real numbers r and α such that all the modules of the vectors in VI are positive integer multiples of r , and all the angles of the

vectors in VI are positive integer multiples of α . A *VGDOL* system must be rationally related to be able to be transformed into an equivalent *TGDOL*.

Alfonseca and Ortega [Alf97] introduced two equivalence theorems between the two families of graphical interpretations *RRVGDOL* and *AITGDOL* as follows:

Theorem 1

For every AITGDOL system which represents a fractal with the usual turtle graphics interpretation and $\alpha=(2 \times k \times \pi)/n$ there exists a fractal-equivalent RRVGDOL system.

For every *AITGDOL* scheme which represents a set of fractals with the usual turtle graphics interpretation and there exists a fractal-equivalent *RRVGDOL* scheme.

Theorem 2

For every RRVGDOL system which represents a fractal with a vector graphics interpretation, there exists a fractal-equivalent AITGDOL system.

For every RRVGDOL scheme which represents a set of fractals with a vector graphics interpretation, there exists a fractal-equivalent AITGDOL scheme.

For the proof and some examples, refer to [Alf97]. In this thesis, only turtle graphics are used to generate fractals.

4.5. Computing the dimension of initial-iterated fractals

In chapter 2, different methods for calculating the dimension of fractals were mentioned. In this section, the fractal dimension computed by Alfonseca and Ortega [Alf01a][Alf00a] is introduced. Similarly to typical ways for calculating the fractal dimension, it is obtained as the ratio between how much the curve grows in length and how much it advances, but it differs because the calculation of the dimension is done by operating directly on the L-system that represents the fractal curve, without performing any graphical representation. Obviously, computing the fractal dimension through operations on strings is an easier method than the computation of a limit.

Each word in the derivation represents a step of the recursive generation of the fractal curve. The production rules embody the allowed transformation between configurations. Therefore, the growth of the words is related to the corresponding growth of the curve. The graphic interpretation of the L-system makes it possible to assign bi-dimensional co-ordinates to the letters in each word. Once these co-ordinates have been computed, it is straightforward to obtain the distance between the different points. These distances may be used as a measure of how much the curve grows in length. This method for computing the dimension of the fractal curve will be used in chapter 7.

The algorithm in [Alf01a][Alf00a] applies to a set of fractal curves which can be represented by an L system containing a single draw symbol and no move or non-graphic symbols. The production set, therefore, consists of a single rule, apart from the trivial rules for symbols +, -, (and). In fact, the algorithm may be applied to other more complicated L-systems, but we won't need them in this thesis.

Informally, the algorithm takes advantage of the fact that the right side of the only applicable rule provides a symbolic description of the fractal generator, which can thus be completely described by a single string. The algorithm computes two numbers: the length N of the visible walk that follows the fractal generator (equal in principle to the number of draw symbols in the generator string, but see below), and the distance d in a straight line from the start to the end point of the walk, measured in turtle step units (this number can also be deduced from the string). The fractal dimension would then be:

$$D = \frac{\log(N)}{\log(d)}$$

The example given below illustrates the use of the algorithm.

The PD0L scheme

$$\begin{aligned} F &::= F+F--F+F \\ + &::= + \\ - &::= - \end{aligned}$$

with axiom $F--F--F$, and a turtle graphic interpretation, where $\{F\}$ is a draw symbol, and the step angle is 60 degrees, represents von Koch snowflake curve.

The string to be considered is the right hand side of the rule:

$$F+F--F+F$$

This string describes the fractal generator. The number of steps along the walk (N) is the number of draw symbols in the string, 4 in this case. The distance d between the extreme points of the generator, computable from the string by applying to it the turtle interpretation, is 3. Therefore, the dimension is:

$$D = \frac{\log(4)}{\log(3)} = 1,2618595071429\dots$$

in accord with the results obtained by other methods, specified by [Man77].

The algorithm presents the following problems:

- The distance d in the denominator may be zero. Computed by the previous formula, D becomes zero. These cases may be excluded, as they do not give rise to fractal curves, but to the same figure indefinitely repeated.
- The distance d in the denominator may be one. Computed by the formula, D becomes infinite. These cases are also excluded, because in every step of derivation the curve expands and is not limited to a finite space. Therefore it is not a fractal in the strict sense.

-
- The length N of the visible walk may not be equal to the number of draw symbols in the generator string. This may happen in two ways:
 - The turtle graphic associated to the string passes more than once along a set of points with a non-zero measure. The algorithm takes this case into account, by computing a non-integer N .

The turtle graphic associated to a derivation of the string passes more than once along a set of points with a non-zero measure. The algorithm also computes this case by taking a certain number of derivations until the quotient converges. For performance reasons, this approach won't be used in this thesis.

For the algorithm itself and further examples, refer to [Alf01a][Alf00a].

Chapter 5

Cellular automata

5.1. Overview

Cellular automata (CA) were originally introduced by Von Neumann in 1966 [Neu66] as a formal model of self-reproducing biological systems.

Cellular automata are a mathematical abstraction method, where the space, time and states that describe the state of the system are all considered discrete. A cellular automaton [Bur70][Kar95][Wol86] has three main components: a *finite automaton*, a *regular lattice (grid)* not necessarily finite, each of whose cells contains a copy of the finite automaton, and a *neighborhood rule* that defines the set of neighbor cells to every position in the grid.

The global behavior of a cellular automaton may be described locally, because each finite automaton in the grid takes as input the states of its neighbors. However, cellular automata with simple local behavior may give rise to complex dynamic systems [Wei91].

The set of particular states of all the automata in the grid of a CA at a given time is called a configuration. The grids can be seen as matrices of states with a given dimension. The most used grids are one-dimensional and bi-dimensional.

Cellular automata have been successfully used in the *practical domain*, in many different ways as simulation tools for a wide variety of disciplines: physical modeling and simulation [Man90], biology [Erm93], fluid dynamics [Mar86], pattern recognition [Bor91], to study the logical organization behind self-reproduction [Lan84]; traffic simulation [Nag94] [Sch93], and urban development simulation [Mes99][War99]; and in the *theoretical domain*, where they have been used as parallel computer abstract architectures [Ima98] [Lin90] [Mor92] [Tof77]. In [Nor89][Cul90] cellular automata were connected with formal languages. They have also been used as a standard method to study other decentralized spatially extended systems. CA's have also been used as an alternative method to solve differential equations [Tof77], and to simulate several physical systems where differential equations are useless or difficult to apply [Tof77]

5.2. Description and types of Cellular automata

There are many different types of cellular automata that differ on their components. These components are the states of the cell, the geometrical form of the lattice, the neighborhood of a cell, and the local transition function. In the following, we

will give a standard formal definition of classic cellular automata, indicating later some of their more or less usual variants.

5.2.1. Lattice

In cellular automata, sets of finite automata are distributed over a regular topological structure, usually associated with a matrix, with no limitation on its dimension (possibly infinite, even).

Definition

Given a set E ,

given n sets of indices not necessarily finite (subsets of increasing contiguous integer numbers starting at 0) $\{I_i\}_{i=0}^{n-1}$, $I_i \subseteq \mathbb{Z} \forall i$.

an n -dimensional lattice over E is any function

$$R: I_0 \times I_1 \times \dots \times I_{n-1} \rightarrow E,$$

$$R_{(i_0, \dots, i_{n-1})} = R_{0, \dots, n-1} \text{ or } R_{[i_0, \dots, i_{n-1}]}$$

$$\forall (i_0, \dots, i_{n-1}) \in I_0 \times I_1 \times \dots \times I_{n-1}$$

Both notations will be used.

Examples

- R_∞ over \mathfrak{R} represents all the infinite vectors of real numbers.
- $R_{\infty, \infty}$ over $\{0, 1\}$ represents an infinite Boolean two-dimensional lattice (grid).
- $R_{3, 3, 3}$ over \mathbb{Z} represents a tridimensional lattice (grid) of integer numbers with a $3 \times 3 \times 3$ dimension.

Theoretically, cellular automata may be considered to include an infinite lattice, but in practice the lattice is always finite. Simulating infinite lattices can be done by means of the periodic boundary condition, with a circular neighborhood, that is, for both rows and columns, the first and the last ones are neighbors.

5.2.2. Neighborhood

The transition function defines the next state of the cell depending on its current state and the states of its neighbors (which act as the input to the finite automaton in the cell). There are different ways to define the neighborhood. The most common neighborhoods are the following:

Von Neumann's neighborhood

- In a bi-dimensional grid, Von Neumann's neighborhood, takes into account the cell and its four nearest neighbors according to the Euclidean distance (see figure 5-1).

$V_N = (5, ((0,0),(0,1),(1,0),(0,-1),(-1,0)))$ where 5 is the number of neighbors

$((0,0), (0,1), (1,0), (0, -1), (-1,0))$ is the offset vector

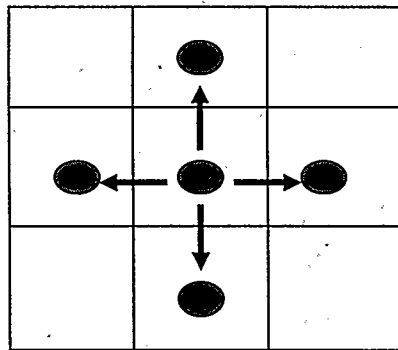


Figure 5-1: Von Neumann neighborhood in a bi-dimensional grid with a central position.

- In a tri-dimensional grid, Von Neumann's neighborhood considers the cell under study and its six nearest neighbors according to the Euclidean distance (see figure 5-2).

$V_N = (7, ((0,0,0),(0,1,0),(1,0,0),(0,-1,0),(-1,0,0),(0,0,1),(0,0,-1)))$

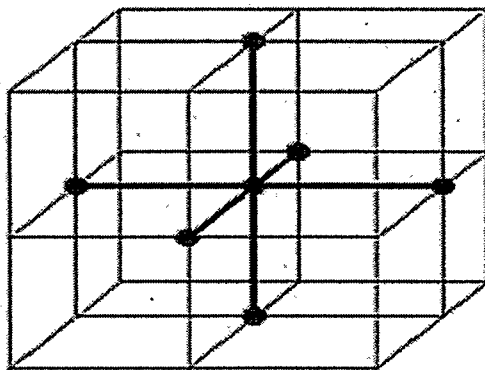


Figure 5-2: Von Neumann's neighborhood in a tri-dimensional grid with a central position.

- Von Neumann's neighborhood can be generalized for an n-dimensional lattice in the expected way.

Moore's Neighborhood

Moore's neighborhood considers the current cell under study and its eight nearest neighbors according to the Euclidean distance.

Formally, in a bi-dimensional lattice, Moore's neighborhood would be the following, also shown in figure 5-3:

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

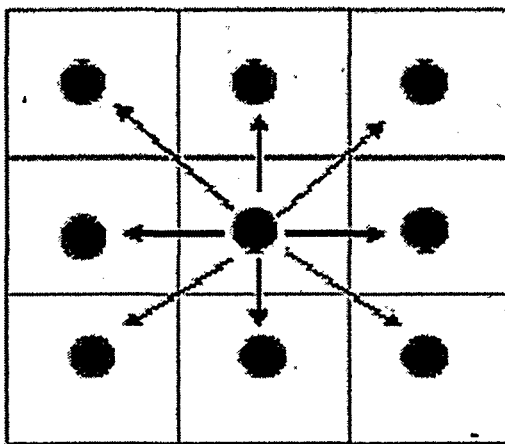


Figure 5-3: Moore's neighborhood

5.2.3. Local Transition Function

A local transition function defines the next state of each finite automaton, given its current input (the set of the states of its neighbors) and its own state.

$f: Q \times Q^k \rightarrow Q$ is the transition function that assigns a next state to each automaton in the grid, depending on its current state and the states of its k neighbors.

This function may be described by means of two different representations forms: *Wolfram representation* and *De Bruijn graphs*.

Wolfram Representation

Wolfram [Wol83] introduced a convenient numbering scheme for elementary CA's. First, all possible neighborhood configurations are written as binary numbers and listed in decreasing order.

In the simplest CA the set of possible states is $\{0,1\}$ and the lattice is one-dimensional. The neighborhood considered consists of the three nearest neighbors (right, left, and the current cell state). The number of possible neighborhood configurations is $2^3 = 8$, and the number of possible transition functions is $2^8 = 256$.

Neighborhood Configuration	Transition function 0	Transition function 1	Transition function 2	...	Transition function 54	...	Transition function 255
000	0	1	0	...	0	...	1
001	0	0	1	...	1	...	1
010	0	0	0	...	1	...	1
011	0	0	0	...	0	...	1
100	0	0	0	...	1	...	1
101	0	0	0	...	1	...	1
110	0	0	0	...	0	...	1
111	0	0	0	...	0	...	1

Table 5.1: Wolfram representation of elementary CA's.

De Bruijn Graphs

For one-dimensional lattices, De Bruijn graphs represent the CA in the following way, which will be clearer with an example: "Transition function 54" is represented as in figure 5-4. The names of the nodes represent the state of the current cell in the lattice and their neighbors in two different ways. There is an arrow from each node whose name represents the state of the left neighbor and the current cell, to the node whose name is made of the states of the current cell and its right neighbor, labeled with the value of the transition function for that neighborhood configuration. So, for instance, $f(110)=0$. Therefore, there is an arrow from node 11 to node 10, with label "0".

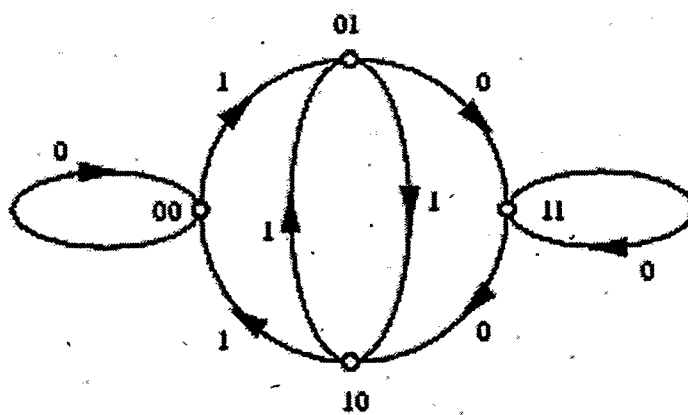


Figure 5-4: De Bruijn representation of rule 54 (from [Del98]).

5.2.4. Classic Cellular Automata

A Classic cellular automaton consists essentially of the following elements:

- A finite set of input symbols.

- A finite set of states.
- A lattice, of any dimension, where each cell contains a deterministic finite automaton.
- A deterministic transition function, defining the next state of the automaton given its current input and state. This function is sometimes described by means of a *transition table*.
- An initial state of the automaton, a distinguished member of the set of states.
- A finite set of objective states.
- The current state.

Definition

Given R , the n -dimensional grid of the automata.

Given Q , the set of all possible states of the cells in R .

We call *configuration* of R and Q in time t , and write it $C(R, Q, t)$ or simply $C(t)$ if there is no ambiguity respect to the considered grid and the set of states, an injunctive function C depending of time, which assigns at every instant in time one state to every automaton in the grid.

$$c(R, Q, t) : R \rightarrow Q$$

A d -dimensional deterministic cellular automaton is the six-fold

$$(G, G_0, Q, V, f, T)$$

where

G is a d -dimensional grid of automata.

Q is the finite and non-empty set of possible states of the automata in the grid.

G_0 is the initial configuration, a mapping $G_0 : G \rightarrow Q$ that assigns an initial state to each automaton in the grid.

$V = (k, N)$ is a one-dimensional neighborhood.

$f : Q \times Q^k \rightarrow Q$ is the transition function that computes the next state of each automaton in the grid, depending on its current state and the states of its k neighbors.

T is a finite and non-empty set of final states of the automaton in the grid.

A well-known deterministic cellular automata known as the game of life is discussed in the next section.

The game of life

Introduced by John Conway [Con], the *game of Life* is a very simple cellular automaton that gives rise to extremely complicated behavior, and has been proved to be computationally complete, being able (in principle) to perform any computation which may be done by digital computers, Turing Machines or neural networks.

The cellular automaton associated to the game of Life is defined thus:

- The grid is rectangular and potentially infinite.
- The set of neighbors to a point in the grid consists of the point itself plus the eight adjacent points in the eight main directions in the compass (Moore's neighborhood).
- Each finite automaton has two states: empty (also called dead; represented by a zero or a space character) and full (also called alive, represented by a one or a star symbol *). The set of states is thus represented by the two Boolean numbers {0,1}, or the two characters: ' *'.
- The transition function is defined by the following simple rules:
 - If the automaton associated to a cell is in the empty state, it goes into the full state if and only if the number of its neighbors in the full state is exactly three.
 - If the automaton associated to a cell is in the full state, it goes into the empty state if and only if the number of its neighbors in the full state is less than two or more than three.
 - In any other case, the automaton remains in the same state.

Each time step is called a "generation". The set of all the cells alive at a given time step is called the "population". In figure 5-5 we see 2 successive generations of the game of life.

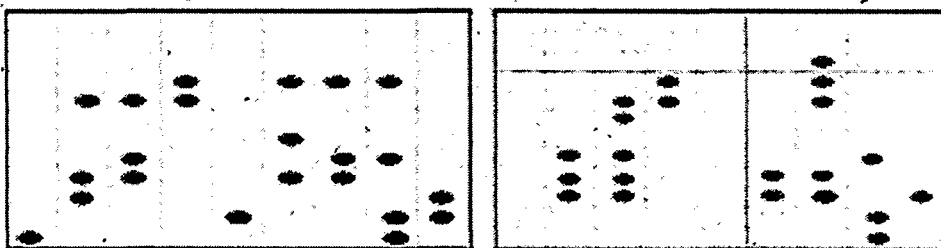


Figure 5-5: a) One initial configuration for Conway's Life. b) The second generation.

5.2.5. Cellular automata variants

There are many different types of cellular automata, depending on the differences of their components. These components are the states of the cell, the geometrical form of the lattice, the neighborhood of a cell, and the local transition function [Sar00]. In the next section several types of cellular automata will be explained

- *Cell states*

In the classical model, all the cells have the same set of attributes, but it is possible to extend the model to allow each cell to have a different set of states.

- *Topology of the lattice*

Cellular automata can differ in the dimension of their grid and the shape of their cells. The most common grids are one-dimensional and bi-dimensional. In the classical model all the cells have the same shape.

- *Neighborhood*

The transition function defines the next state of the cell depending on its current state and its neighborhood. There are different ways to define the neighborhood as mentioned previously.

- *Transition function*

In the classical model, the transition function is the same for all cells. Cellular automata that use different functions for different cells are called inhomogeneous cellular automata. Types of cellular automata that use different transition functions have been studied in connection to VLSI applications [Ser90] [Cha97] [Sar98].

Non-deterministic and stochastic cellular automata use respectively non-deterministic and probabilistic transition functions.

5.3. Cellular automata and the edge of chaos

Wolfram [Wol94] studied the one dimensional binary CA, and noticed that they reveal the full behavior spectrum of dynamical systems. He proposed a qualitative classification of all CA's in four groups:

Class I: is related to limit points (homogeneous state) in the phase space.

Class II: is related to limit cycle (periodic structure).

Class III: is related to chaotic behaviors, unpredictable space-time behaviors.

Class IV: is related to complex behaviors, sometimes long-lived.

Some Class IV CA's are assumed to support universal computation.

This classification was initially proposed for one-dimensional cellular automata, and was later extended by Packard [Pac88] to include bi-dimensional cellular automata.

Langton [Lan90] studied the relationship between the average dynamical behaviors of cellular automata. He hypothesized that there is a virtual value that forces the CA to change its behavior from one class to another. He called this value lambda (λ). The lambda of a given CA rule is the fraction of non-quiescent output states in the rule table, where the quiescent state is arbitrarily chosen as one of the possible k states. For binary-state CA's, the quiescent state is usually 0 and therefore lambda equals the fraction of output-1 bits in the rule table.

Langton assumed that CA's capable of universal computation would have a critical λ value corresponding to a phase transition between ordered and chaotic behavior [Mit94]. He used various statistics methods to classify CA average behavior at each λ value.

5.4. Cellular Automata Programming

Programming cellular automata to solve a specific predefined task is considered very difficult, because it is very hard to design or modify the local cellular automata dynamics, in order to perform the pre-specified global task. The best known problems tackled in CA programming are the task classification problem, generating random numbers, and synchronization. However, many researchers attempt to solve these problems with different techniques, to obtain a higher performance. This problem is still open.

Another problem under study concerning CA's is related to signal transformation: how the information in the cellular automata can be interpreted as signals that propagated across the lattice, as we will see in following sections.

5.4.1. Task classification problem

The task problem mostly tackled in one-dimensional grids consists in foreseeing if the CA, after a predefined number of iterations, will relax to a configuration of all 0's or all 1's, depending on the initial configuration. If the initial configuration has a majority of 1's, the CA should relax to all 1's or to all 0's. As previously mentioned, the CA is governed by local interaction rules, which means that there is no information

given to the automata about its global behavior when the neighborhood configuration is very small, compared to the size of the grid.

This problem is considered a complex computation problem that CA should solve, where the local transition function is encoded as a program that should execute the input (encoded as the initial configuration) and, after a number of generations, give rise to the output, in the form of another spatial configuration [Hor99].

5.4.2. GAs: How they work with CA

Genetic algorithms and evolutionary computation have been used with CA's since 1988, when Packard [Pac88] attempted to evolve the transition function rules of one-dimensional cellular automata to find a better performance of the Gacs-Kurdyumov-Levin rule [Kur78]. A GKL CA is a one-dimensional lattice with two states (0,1) and a (-1,0,1) neighborhood, whose the lattice size is 149. He analyzed the frequency of the CA transition function in the population as a function of the Langton parameter λ . In the final generation of his GA experiments, two different peaks were observed in the frequency distribution around critical values of λ . Considering that Langton associated the critical value of λ as a parameter of the phase transition from periodic to chaotic boundaries, and claimed that at this value the complex behavior occurs [Hor99], Packard hypothesized that:

1. CA's that are able to perform complex computation are found near the critical value of lambda.
2. When CA rules are evolved to perform a complex computation, evolution will tend to select rules with lambda values close to the critical values.

Mitchell and her colleagues [Mit93] conclude that there is no evidence for a generic relationship between λ and the computational ability of a CA. They repeated Packard's experiment but did not get the same results as those Packard obtained. The difference between their experiment results and the results obtained by Packard is probably due to additional mechanisms in the CA's used in the original experiment, that were not reported by Packard.

Later, other researches [Mit93][Mit94a][Mit94b] have tried to use genetic algorithms with different parameters, or other evolutionary algorithms, to obtain better performance than the handmade CA's designed by Gacs, Kurdyumov and Levin. However, it was assumed that it is impossible to obtain 100% performance.

Capcarrere [Cap01] assumed that two conditions are necessary to correctly classify a CA. First the density of the initial configuration must be preserved overtime and, secondly, the density of the rule table must be one-half. This means that the proportion of ones and zeros of the rules should be equal.

Other works [Sip96a][Sip96b] have used GA to evolve non-uniform CA's to solve different topics of CA programming, such as the density classification problem, random number generation, ordering, or rectangle filling, where some of this tasks were done on bi-dimensional CA.

Paredis [Par97] used Co-evolutionary Genetic Algorithms (CGA), where two non-interbreeding populations are co-evolved (CA function rules and initial configurations), which interact as predator and prey. Co-evolutionary Genetic Algorithms produce better performance than the handmade CA's designed by Gacs, Kurdyumov and Levin.

Synchronization

The idea of programming CA's to perform synchronization tasks has to do with finding cellular automata that can recover the initial configuration, after N iterations, to obtain periodic patterns around configurations of all 0's or all 1's. There are different types of synchronization; for example, instead of a vertical periodic pattern, we can be interested in a horizontal periodic pattern synchronization. For more details, see [Hor99].

5.4.3. Signals

Signal is a concept associated to data transmitted through the grids of CA. To model massively parallel computation in CA, it seems that signals could be of great interest, they are not only a natural tool to collect and dispatch the information through CA's grids but more deeply, this notion appears to be a strength way to encode and combine information.

In [Maz99][Terr91] signals in one-dimensional CA are exhaustively studied. The kind of signals possible in this type of CA could be described by means of geometric diagrams where successive generations of CA are shown as curves on the diagrams.

In [Terr91] the conditions that a diagram must hold to ensure that there exist a cellular automaton that can draw its signals are established; so, signals could be considered a tool for designing and programming one-dimensional CA.

Other works [Terr99] had been performed to study signals in CA with dimension greater than one.

Chapter 6

L-Systems and Cellular Automata.

6.1. Overview

Different facets of the relationship between L Systems and Cellular Automata have been explored. [Koz93] underlines the structural similarities between both formalisms by applying genetic programming to them, Stauffer and Sipper [Sta98a][Sta98b][Sip97] build cellular automata equivalent to the turtle graphic interpretation of some self-replicating L systems.

This chapter summarizes the work of the directors of this thesis [Alf00b] concerning the possibility of generating L-Systems equivalent to given cellular automata.

6.2. L-Systems and Cellular automata

There are several similarities between L-systems and cellular automata:

- Both of them have data that can be considered their initial states:
 - The axiom is the first string for an L-system.
 - The initial configuration of a CA is its beginning state.
- They also have some mechanism to drive how the system changes:
 - The set of production rules for an L-system
 - The transition function of its finite automata for a cellular automaton.
- Both architectures exhibit intrinsic parallelism: production rules and transition function are applied simultaneously to every symbol and single automaton.

None of the previous works mentioned have really faced the equivalence between CA and L-systems.

In [Alf00b] an algorithm to design L-systems equivalent to given cellular automata is proposed. They show three preliminary examples to fully understand the method.

6.3. One-dimensional binary cellular automaton with three inputs that generates the Sierpinski gasket

In the cellular automaton of this example, the new state of each automaton is a function of its own state and that of its immediate neighbors, left and right.

These Cellular automata can be defined as follows:

Three bits are used to represent the state of the three neighbors of each automaton.

- There will be 8 possible configurations $2^3=8$.
- It is clear that there will be 2^8 possible state change rules.
- The transition function of the automaton can be encoded in decimal notation with a number from 0 to 255, which represent the eight new state bits corresponding to the eight input configurations.

For example, the function that correspond to the decimal number 90 with binary notation 01011010 will have the following output

State of previous automaton	State of current Automaton	State of following Automaton	New state of this automaton
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

Table 6-1: Cellular automata transition function 90.

It is easy to construct a (1,1) DIL system whose words correspond to the consecutive generations of this automaton.

The alphabet is defined as $V_{90}=\{0,1\}$.

The set of production rules P can be directly obtained from the table

$P_{90}=\{111::=0, 110::=1, 101::=0, 100::=1, 011::=1, 010::=0, 001::=1, 000::=0\}$.

The axiom is the binary string that represents the initial configuration of the cellular automaton. There is one single automaton with initial state 1, and all the others were initialized with state 0. State 1 is represented by the * symbol. The axiom for this example is $\alpha_{90}=0...010...0$. The following figure shows the first 24 generations of the cellular automata.

The L-system for this example is $S_{90}=(V_{90}, P_{90})$

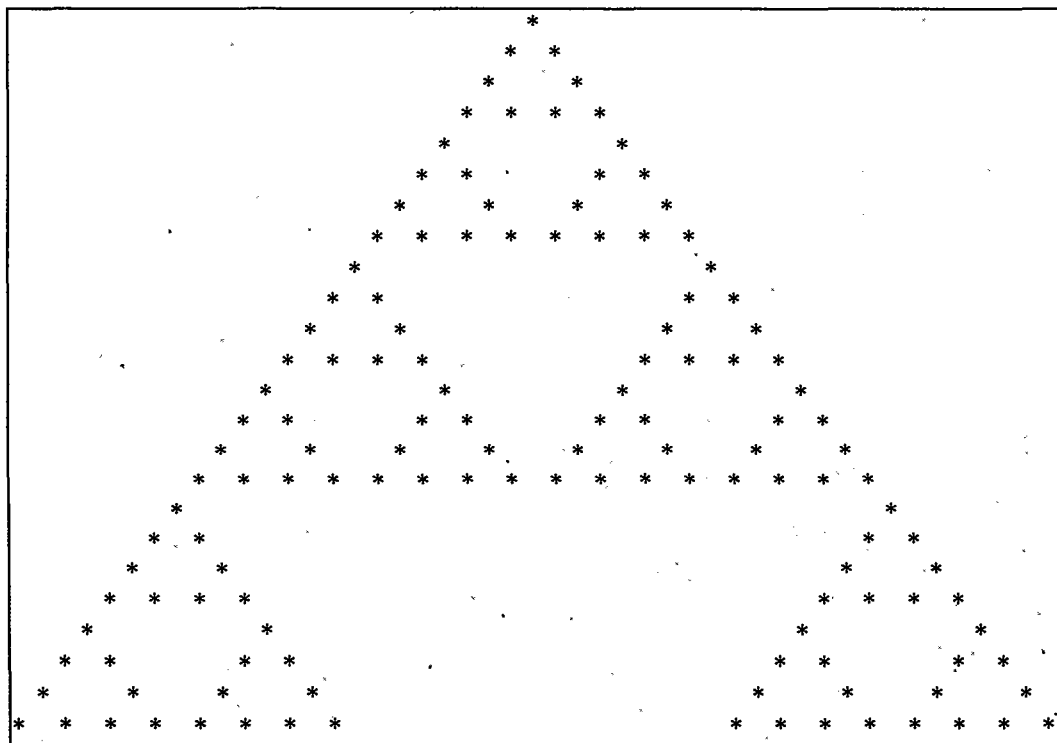


Figure 6-1: The first 24 generations of the CA₉₀ from 0...010...0

6.4. IL-System equivalent to bi-dimensional CA that simulate ecosystem model

In [Alf00b] a CA that simulates an ecosystem is described: it has a rectangular grid and their states represent a combination of individuals (predators and preys). There are two possible states for every predator (a and b) and one state for the preys (x). Each cell can have 4 individuals of every kind as maximum. The state of each cell changes in two alternative steps:

1. In the first step, the neighborhood of each cell is just itself. Predation and reproduction happens according to the following rules:
 - a) A predator in the a state dies if there is no prey in the same cell.
 - b) A predator in the a state goes into the b state if there are at least two prey individuals in the same cell and there is room for a predator in the state b in the cell. In this case one of the prey individuals dies (is eaten).
 - c) A predator in the b state goes into the a state if there is no prey in the same cell.
 - d) A predator in the b state becomes two predators in the a state (reproduces) if there are at least two prey individuals in the same cell and there is room for the two predators in the a state in the cell. In this case one prey dies (is eaten).
 - e) The prey reproduces if there are at least two and at most three individuals in the same cell.



2. In the second step, movement of predators and preys takes place. The rules for movement use the Von Neumann neighborhood. The goal is to simulate some kind of non-deterministic movement for each individual. Each individual changes its direction by choosing at random one of the four possible direction states (north, south, east and west)

It is possible to devise a bi-dimensional L-system whose derived words correspond to the generations of this automaton. For instance, $a^1 b^1 x^3$ (that means 1 predator of type a, 1 predator of type b and 3 preys) is transformed into $a^2 b^1 x^2$ applying the previous rules for predation and reproduction; so the following rule must belong to the set of rules of the bi dimensional L system:

$$S_{113} ::= S_{212}$$

The total number of symbols equals the number of variations with the number of repetitions of five elements (0,1,2,3,4) taken 3 at a time that is $5^3=125$.

The equivalent L-system is $(\{S_i\}_{i=1}^{125}, P, \alpha_1)$. The axiom (α_1) is the matrix of the initial states of all the automata in the grid, translated by means of the following function, which converts a state of the finite automaton into a symbol of the L system:

$$f(a^{#a} b^{#b} x^{#x}) = S_{\#a\#b\#x}$$

Concerning the movement of the individuals, each of them has associated one of the following special symbols $\{\leftarrow, \uparrow, \rightarrow, \downarrow\}$. It is not allowed that two individuals of the same type have the same direction. If each sub index is represented as binary numbers with four digits, each digit could be associated with a direction to indicate that there is an individual of its type that will move to the corresponding direction. For example the L-system state $S_{0111,1111,0010}$ could indicate the directions of three individuals with type a, for example, \uparrow , \rightarrow and \downarrow ; four individuals with type b, pointing to the four possible directions, and one prey, pointing to direction \rightarrow .

The rules of the final L system, which includes information about direction, will have the following form:

$$S_{0111,1111,0010} \text{ becomes } S_{0111,1111,0010,3,4,1}$$

And the alphabet will be:

$$\Sigma = \{ S_{n_a, n_b, n_x, e_a, e_b, e_x} \mid e_i \text{ is the number of ones in } n_i \forall i \in \{a, b, x\} \}$$

For more details, see [Alf00b].

6.5. IL -System equivalent to three-dimensional CA that generates and propagates pulses

The cellular automaton has the following three-dimensional grid:

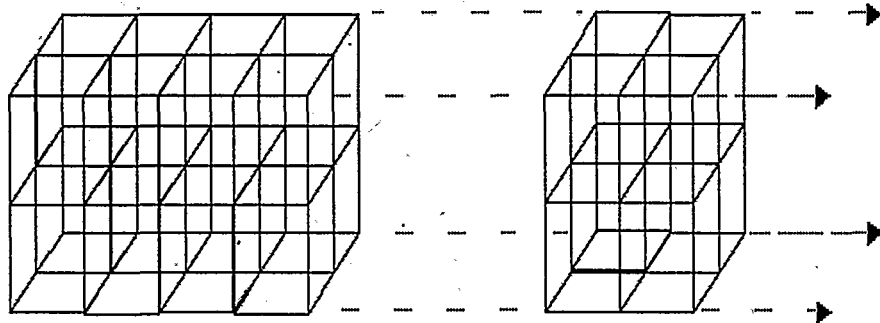


Figure 6-2: Three dimensional cellular automaton grid

The neighborhood of each cell includes the cell itself and the six nearest neighbors, the six cells that surround it at distance 1. The set of states is $\{0,1\}$. To calculate the next state of the automata the following rules are applied:

- Horizontal neighbor to the right are not take into account.
- If the four vertical neighbors are at state 0, the state of the considered automaton changes no matter the value of the neighbor to the left.
- If the four neighbors in the vertical plane have state 1, the next state of the current automaton depends on the state of its neighbor to the left and the state of the current automaton. The state changes when both values are not the same.
- Otherwise the automaton remains unchanged.

Again, it is possible to devise a three dimensional IL-system whose derived words correspond to consecutive generations of the automaton. The alphabet of this system is $\Sigma=\{0,1\}$. The set of production rules can be defined as the follows:

$$P=\{ \begin{array}{l} 0000xy0::=1 \quad \forall x,y \in \Sigma; \\ 0000xy1::=0 \quad \forall x,y \in \Sigma; \\ 1111x00::=0 \quad \forall x \in \Sigma; \\ 1111x01::=0 \quad \forall x \in \Sigma; \\ 1111x10::=1 \quad \forall x \in \Sigma; \\ 1111x11::=1 \quad \forall x \in \Sigma; \\ \text{otherwise } xyzuvw::=s \quad \forall x,y,z,u,v,w,z,s \in \Sigma \}. \end{array}$$

Where the symbols of each left hand side represent the state of, respectively, the up, down, front, back, right and left neighbors and the current automata.

The axiom α is the three dimensional binary array obtained from the initial state figure (1 dark, 0 otherwise). The equivalent L system is $\{\Sigma, P, \alpha\}$.

It could be concluded that [Alf00b]:

- The dimension of the studied grids is not a limitation for the approach.
- The bottleneck of the approach is the formalization of the transition function of the CA. The following technique is recommended:
 - First, identify the independent and radically different behavior of the automata.
 - Second, design a set of production rules that shows every behavior.
 - Next, put them as an L-system with tables, and decide where and when each table should be used.
- One or more sets of production rules could be joined together in an L-system with tables. Building each set of production rules is possible because every automaton considers a finite number of nearest points in the grid as their neighborhood and every automaton has a finite set of possible states.

In subsequent work [Alf00b], the directors of this thesis show a general equivalence theorem between probabilistic CA and L-systems. The previous examples are particular cases of this general result.

Theorem: Given a probabilistic n -dimensional cellular automata $A = \{G, G_0, N, M, Q\}$, There is an equivalent probabilistic n -dimensional IL system that is step-equivalent to the cellular automata.

Where the step-equivalence relationship is defined as follows: let A be a probabilistic cellular automaton; let S be a probabilistic IL-system; A and S are step equivalent if and only if for every possible configuration of the automaton A it is possible to find a string in the language generated by S such that the probability of being in the configuration and the probability of deriving the word from the axiom are the same.

More details could be found in [Alf03].

6.6 Equivalence between L-Systems and Cellular automata.

There exists some attempt to study the reverse equivalence, that is, the design of CA that can simulate L-systems. This chapter outlines them: first the concept of signal is introduced, and then some results in this direction are briefly described.

Signal is the name given to the presence of information that moves across the grid of a cellular automaton. Several authors have exhaustively studied signals in one-dimensional cellular automata and have proposed [Terr91] [Maz99] methods to design CA by means of the graphical definition of signals. The same authors [Terr99] have started the study of signals in bi-dimensional CA.

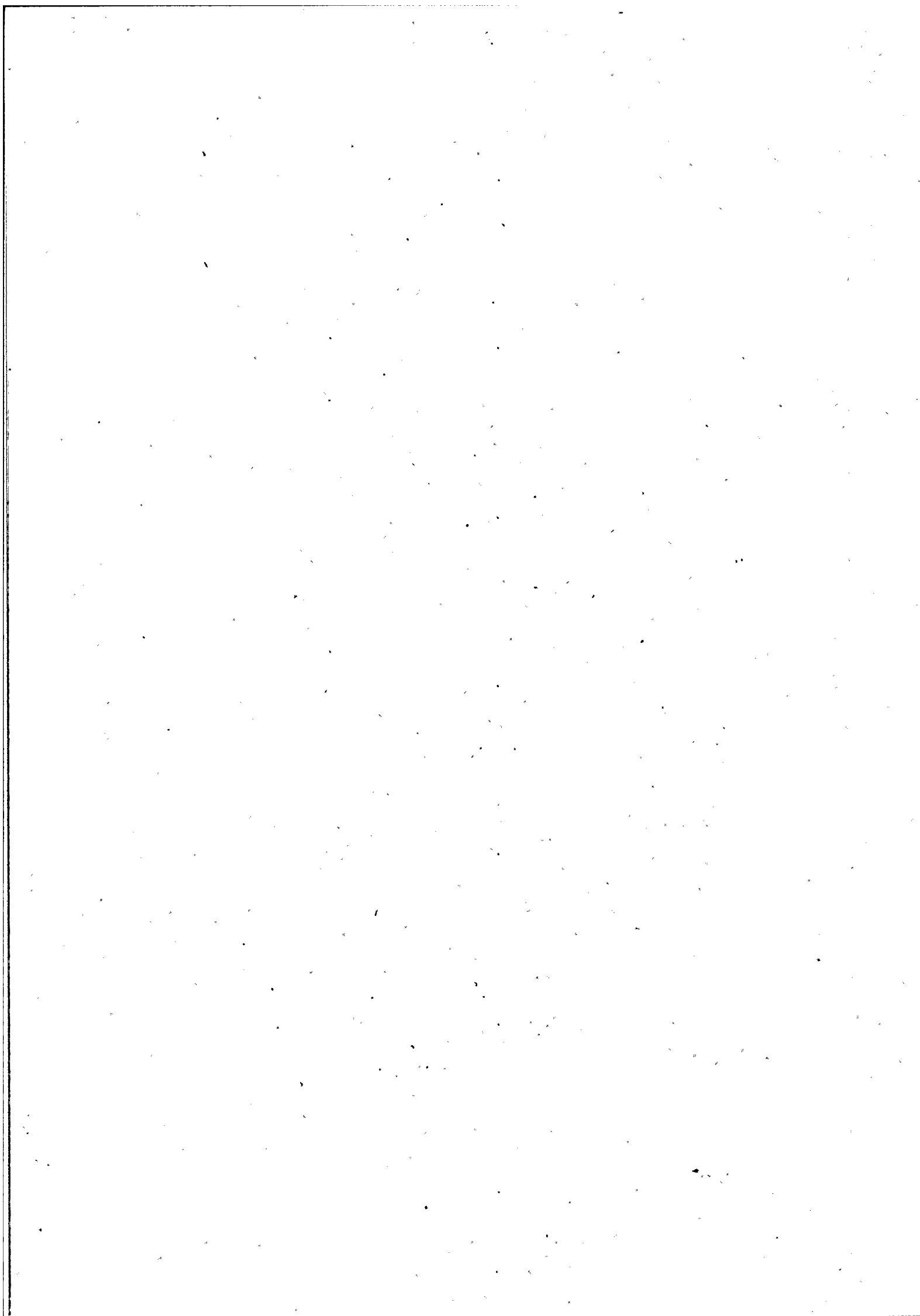
Signals have also been identified by Wolfram [Wol94] as a condition that a cellular automaton must satisfy to exhibit complex behavior. So it seems clear that signals could constitute a useful tool in the design and programming of CA.

Signals are one of the properties that make Conway's game of life computationally complete.

In [Terr91] [Terr99] some kind of equivalence between L-systems and CA is studied. The authors propose a method to design a one-dimensional interactive CA that generates a signal with a frequency that equals the grow function of any DOL system. Their method really designs graphical diagrams that contain some signals that can be generated by a one-dimensional cellular automaton.

As it will be detailed described in the corresponding chapters, the result of this thesis greatly differs from the approach mentioned above.

**Genetic Evolution &
Equivalence between
some Complex Systems**



Chapter 7

Grammatical Evolution to Design Fractal Curves with a Given Dimension

7.1. Overview

There are two techniques for generating fractal curves: deterministic techniques and non-deterministic.

Fractal curves of a given dimension can be obtained by means of deterministic techniques, by applying certain tools that use some measure of the regularity of continuous real functions with a single real variable. Hölder exponent is one of these tools [Gui98]. In [Dao98] three different methods are described to build a function that interpolates a set of points with a prescribed local regularity, measured by Hölder exponent. Hölder exponent is calculated by means of Schauder basis [Jaf95], using Weierstrass type functions and by a generalization of Iterated Function Systems (IFS).

In this chapter we propose a new method that applies a non-deterministic technique to work on a formal representation of the target system, rather than the real curve. This approach seems more flexible and general, because formal models as Lindenmayer grammars are powerful enough to simulate a wide range of different complex systems. We expect our technique to be also applicable to other domains that can be described in a similar way.

This chapter extends Grammatical Evolution (described in chapter 3) to Lindenmayer systems, to solve the problem of obtaining arbitrary fractal curves with a given dimension. Other evolutionary algorithms, rather than Grammatical Evolution, have been applied by other authors to L systems. Ochoa [Och98] evolves DOL systems with a single rule that generate shapes similar to plants. Other authors [Hor03][Jac94][Tra96] make parametric L systems evolve (an extension of Lindenmayer grammars described in chapter 3), where they are faced with the important problem that parametric systems are not closed under the action of genetic algorithms. Grammatical Evolution would solve this problem easily, because it ensures that the generated grammars are syntactically correct. We are not tackling this problem here and leave it as future work, for in the case described in this chapter we are using DOL systems (which are not parametric). In this particular case, simple genetic algorithms would have been sufficient. However, we have grounds to prefer the Grammatical Evolution approach, as explained in the conclusions, in the final part of this thesis.

7.2. The design of L-Systems that represent curves with a given fractal dimension

Designing fractal curves with a given dimension is relatively easy for certain values of the desired dimension, but very difficult for others. The following L-system rules represent (with a turtle graphic interpretation based on an angle step of 60 degrees) the iterators for three different fractal curves with the same dimension: $1.2618595\dots$ ($\log 4 / \log 3$). The first one, as shown above in chapter 4, corresponds to von Koch's snowflake curve. All four, and a few more, could have been obtained by hand, by a simple geometrical study of the curve iterator.

$F ::= F+F--F+F$

$F ::= F+F-$

$F ::= +F-FF-F+$

$F ::= F+F-F-F+$

On the other hand, designing a fractal curve with a dimension of 1.255 would be much more complicated. The first step would consist of obtaining two integer numbers, a and b , such that

$$D=1.255 = \log a / \log b$$

This step could be relaxed to asking for two integers a, b such that the given dimension would be approximated within some degree of accuracy (for instance, with an error less than 0.001).

The second step would be to design a geometrical iterator such that it would take a steps to advance a distance equal to b .

Although there is no solution for every positive real number D , there is always a solution for positive rational numbers

$$D=p/q \quad \text{with } p, q \text{ positive integers}$$

In this case, we would like to find two numbers a and b such that

$$\frac{\log(a)}{\log(b)} = \frac{p}{q} \Leftrightarrow a^q = b^p$$

For any integer $x > 1$, $a = x^p$ and $b = x^q$ is an analytical solution for the previous equation.

Example:

$D=1.6=p/q=16/10=8/5$. In this case, $a=x^8$, $b=x^5$

For $x=2$, $a=2^8$, $b=2^5$

$$\frac{\log(2^8)}{\log(2^5)} = \frac{8\log(2)}{5\log(2)} = 1,6$$

Using the previous equation to generate a graphical shape is complicated, because we would need to design a fractal that walks $2^8=256$ steps to advance only $2^5=32$ steps with a specific angle.

We solve this problem automatically by means of Grammatical Evolution. Our genetic algorithm acts on genotypes made of vectors of integers and makes use of a fixed grammar to translate the genotypes into an intermediate level, which can be interpreted as a rule for an L system which, together with a turtle graphic interpretation, generates the final phenotype: a fractal curve with the desired dimension, or an approximation of the same.

7.3. The developmental algorithm

The initial population consists of 64 vectors of 8 integers in the interval [0,10]. The genetic algorithm later generates vectors of different lengths. Each number represents a rule from an L-system. Other intervals (such as [0,255]) can be used, so as to include genetic code degeneracy, i.e. when different integers in the previous interval (different genes) represent the same L-system rule, as we will see later. This has been tested and it also works, although no significant improvement in performance has been detected.

In our first experiment, the genotype of one individual in the population (a vector of n integers) is translated by making use of the following DOL grammar:

- 0: $F ::= F$
- 1: $F ::= FF$
- 2: $F ::= F+$
- 3: $F ::= F-$
- 4: $F ::= +F$
- 5: $F ::= -F$
- 6: $F ::= F+F$
- 7: $F ::= F-F$

8: F ::= +

9: F ::= -

10: F ::= ϵ

where ϵ is the empty string.

The translation from the strings of *codons* (the vector of integers) is performed according to the following developmental algorithm:

1. The axiom (first word) of the D0L grammar is assumed to be F.
2. As many elements from the remainder of the genotype are taken (and removed) from the left of the genotype as the number of F's in the current word. If there remain too few elements in the genotype, the required number is completed circularly.
3. The current word derives a new word in the following way: each F in the word is replaced by the right hand side of the rule with the same number as the corresponding integer obtained in the preceding step. If genetic code degeneracy is used, the rule applied is that one whose number is the remainder of the corresponding integer modulo 11.
4. If the genotype became empty in step 2, the algorithm stops and the last derived word is the output.

If the derived word has no F's, the whole word is replaced by the axiom.

5. Go to step 2.

In any derivation, the following implicit rules are also applied:

+ ::= +

- ::= -

Let us look at an example. Let us suppose that we are translating the following 7-element vector:

10 6 7 6 0 2 7

We start from the axiom:

F

It contains one F. Therefore, at step 2 we extract one element from the left of the genotype (10). The remainder of the genotype becomes

6 7 6 0 2 7

In step 3, by applying rule 10, the axiom derives ϵ (the empty string). The derived word has no F, thus in step 5 we replace it by the axiom:

F

This is the second word in the derivation. We go back to step 2. The current word contains one F. Therefore, we take one element (6) from the remainder of the genotype, which becomes

7 6 0 2 7

In step 3 we now apply rule 6 to the only F, deriving:

F+F

This is the third word in the derivation. We go back to step 2. The current word contains two F's. Therefore, we take two elements (7,6) from the remainder of the genotype, which becomes

0 2 7

We now apply rule 7 to the first F and rule 6 to the second F in F+F, deriving:

F-F+F+F

This is the fourth word in the derivation. We go back to step 2. The current word contains four F's. Therefore, we should take four elements from the remainder of the genotype, but we only have three. We complete the required number circularly and take (0,2,7,0). The genotype vector is now empty. We now apply rule 0 to the first F, rule 2 to the second, rule 7 to the third and rule 0 to the fourth F in F-F+F+F, deriving:

F-F++F-F+F

This is the last word in the derivation, the result of the algorithm.

We can now simplify the output by erasing unnecessary +- pairs, if any (there are none in this case). We may also add or delete + or - signs at the beginning or the end of the word, so that the turtle ends its movement in the same direction it started (this is a requirement for some of the theorems we are applying). In this case, we get F-F++F-F+F-. The rules of the DOL system generated by the developmental algorithm are:

F ::= F-F++F-F+F-

+ ::= +

- ::= -

Using an interval larger than the number of rules (e.g. [0-255] with the eleven rules seen before), tests the use of degeneracy. In this case, rule 7 (for instance) corresponds to 23 different genes:

7, 18, 29, 40, 51, 62, 73, 84, 95, 106, 117, 128, 139, 150, 161, 172, 183, 194, 205, 216, 227, 238, 249

The chosen rule is the *gene number modulo the number of production rules in the L-system.*

The typical genetic operators (mutation and cross-over) will affect the new individuals in the population. Depending on the interval used, their behavior will be slightly different. For instance, with the [0-10] interval, when the mutation is applied to gene 7, the probability to choose the same gene for the next generation is 1/11. On the other hand, if we work in the [0-255] interval, rule number 7 can be applied with different genes, and the probability of obtaining an equivalent gene after a mutation decreases to 23/256.

7.4. The genetic algorithm

We can now apply the algorithm described in chapter 6 to compute, from F-F++F-F+F-, the dimension of the fractal curve obtained from the DOL system by means of a turtle graphic interpretation with a given angle step. This dimension can be compared to the target dimension, providing a fitness rule for the genetic algorithm.

The scheme for the genetic algorithm is as follows:

1. Generate a random population of 64 vectors of 8 integers in the [0,10] or the [0,255] interval.
2. Translate every individual genotype into a word in the alphabet {F+-}, using the developmental algorithm described above.
3. Compute the dimension of the fractal curves represented by the corresponding DOL system.
4. Compute the fitness of every genotype as $1/|\text{target} - \text{dimension}|$.
5. Order the 64 genotypes from higher to lower fitness.
6. If the fitness of the genotype with the highest fitness is higher than the target fitness, stop and return this genotype.
7. From the ordered list of 64 genotypes obtained in step 5, remove the 16 genotypes with least fitness (leaving 48) and take the 16 genotypes with most fitness. Pair these 16 genotypes randomly to make 8 pairs. Each pair generates another pair, a copy of their parents, modified according to four genetic operations. The new 16 genotypes are added to the remaining population of 48 to make again 64, and their fitness is computed as in steps 2 to 4.
8. Go to step 5.

The four genetic operations mentioned in the algorithm are:

- Recombination (applied to 100% generated genotypes). Given a pair of genotypes, $(x_1, x_2 \dots x_n)$ and $(y_1, y_2 \dots y_m)$, a random integer is generated in the interval $[0, \min(n,m)]$. Let it be i . The resulting recombined genotypes are: $(x_1, x_2 \dots x_{i-1}, y_i, y_{i+1} \dots y_m)$ and $(y_1, y_2 \dots y_{i-1}, x_i, x_{i+1} \dots x_n)$.

- Mutation (applied to n1 % generated genotypes if both parents are equal, to n2 % if they are different). It consists of replacing a single random element of the vector by a random integer in the same interval.
- Fusion (applied to n3 % generated genotypes). The genotype is replaced by a catenation of itself with a piece randomly broken from either itself or its brother's genotype. (In some tests, the whole genotype was used, rather than a piece of it).
- Elision (applied to 5% generated genotypes). One integer in the vector (in a random position) is eliminated.

The last two operations allow longer or shorter genotypes to be obtained from the original 8 element vectors. The optimal values of n1 (100), n2 (100) and n3 (25) have been obtained by means of a set of 22 tests that combine different angles and target dimensions. Table 7.1 shows that these parameters are important, for different combinations of values give rise to very different computing times.

n1	n2	n3	Average generations	Average CPU time
20	20	5	6668	1838
50	20	5	2979	1888
50	50	5	3794	3211
80	10	5	2625	1590
80	80	5	3917	1430
100	100	5	2216	1007
100	100	1	10172	4776
100	100	10	1027	615
100	100	25	146	176
100	100	50	163	497
100	100	90	49	497

Table 7-1: Results of experiments to get optimal values of genetic operation rates.

The algorithm has three input parameters: the target dimension, the target minimum fitness, and the angle step for the turtle graphics interpretation.

7.5. Parallels to biological evolution

This procedure is similar to biological evolution in many respects. There are three different levels (see figure 7.1):

1. The genotype (nucleic acids), here represented by vectors of integers. Each integer can be considered comparable either to a gen, or to a codon (a triplet of nucleotids, the basic unit for the translation of a gen into a protein).
2. The intermediate level (proteins), here represented by words on the {F,+,-} alphabet. The translation from the genotype to the intermediate level is performed using a fixed grammar (the equivalent of the fixed genetic code).

3. The final phenotype (organisms), here represented by the fractal curves, which are obtained from the L systems built from the intermediate level words by means of a turtle graphic interpretation.

The use of the interval $[0,255]$ for the first level introduces a form of code degeneracy, similar to that in the biological genetic code, where the same aminoacid may be represented by more than one codon. In our case (as explained before) the same rule in the grammar may be represented by 22 or 23 different integers. This is a degree of code degeneracy much larger than the biological one, where the maximum number of codons that represent the same aminoacid is four. Intermediate degrees could be experimented easily, but we chose not to do it, seeing that degeneracy doesn't seem to affect performance in this case.

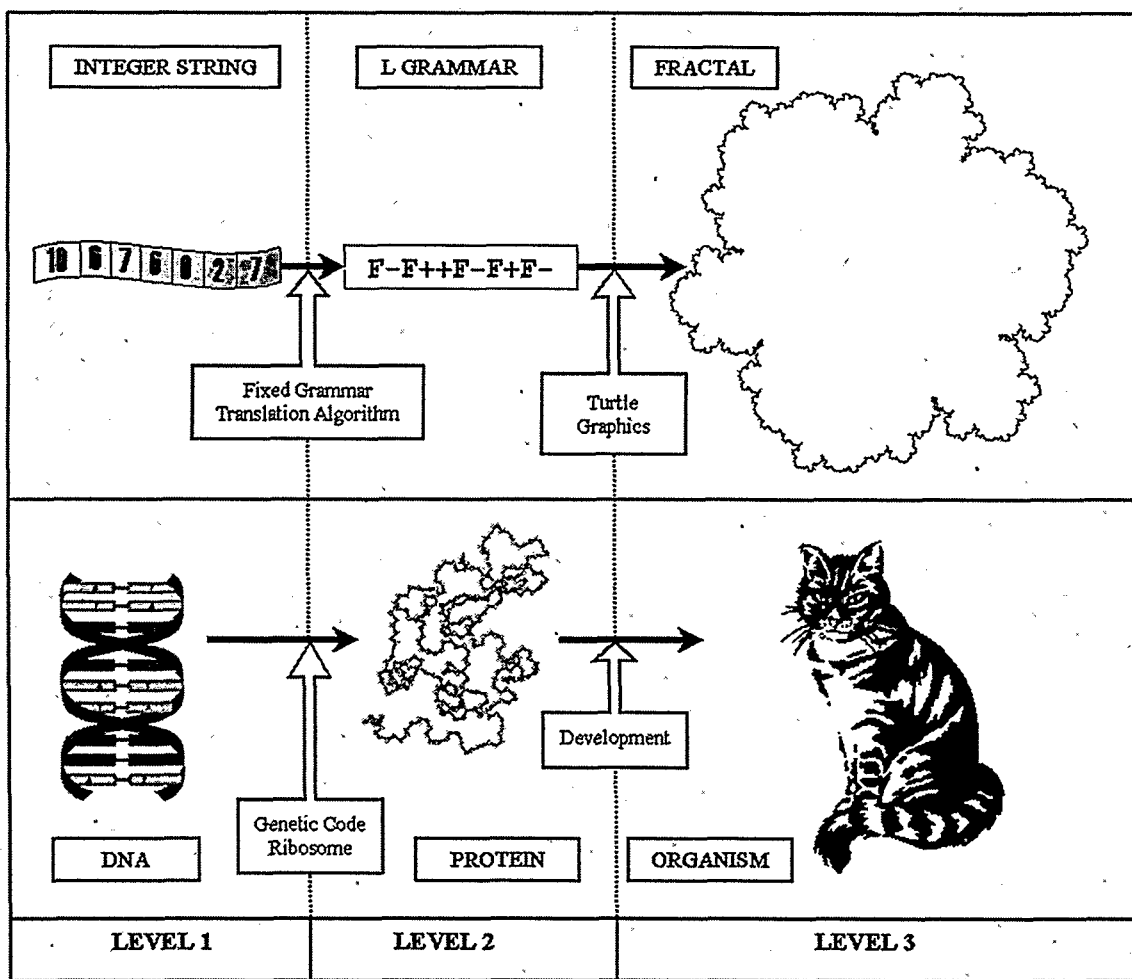


Figure 7-1: Parallels between our Grammatical Evolution approach and biological evolution.

7.6. Evolving the turtle angle

In a second experiment, we take the angle step for the turtle graphics interpretation out from the input parameters and evolve it at the same time. To do that, the genotype of each individual in the population contains one more element (it is a vector of $n+1$ integers). The first element (or its remainder modulo 11) is interpreted as an index to a vector that defines the angle to be used in the graphic interpretation of the phenotype. Eleven possible angles have been used: 120, 90, 72, 60, 45, 40, 36, 30, 24, 20 and 18 degrees (which have been chosen among the first submultiples of 360). The developmental algorithm is applied only to the last n elements of the genotype. The genetic algorithm applies to all the $n+1$ elements of the genotype. In this way, the angle itself evolves, and fractal curves with unexpected angles may be obtained.

7.7. Results

The algorithm described above reaches its targets with surprising speed. Sometimes (for the simplest dimensions, those that can be done by hand) the target is reached in the first generation: a set of 64 random eight-element genotypes has a big probability of including the codification of one of those phenotypes). For other, less standard dimensions, the number of generations to reach a given approximation to the target is usually larger, sometimes quite large. Table 7-2 shows a few of the results we have obtained.

Dimension	Angle	Nr.of tests	Number of generations to reach target
1.1	45	10	37 to 9068
1.1	60	4	119 to 72122
1.2	45	8	188 to 11173
1.2	60	10	21 to 750
1.3	45	9	50 to 18627
1.3	60	4	14643 to 66274
1.25	60	2	1198 to 3713
1.255	60	15	1 to 2422
1.2618595...	60	4	1 to 2
1.4	45	10	79 to 781
1.4	60	10	33 to 1912
1.5	45	11	52 to 11138
1.5	60	8	12 to 700
1.6	45	5	275 to 3944
1.6	60	1	116913
1.7	45	2	585 to 1456
1.7	60	8	18 to 1221
1.8	45	2	855 to 2378
1.8	60	13	69 to 3659
1.9	72	1	5467
1.95	90	1	956
2	45	5	1
2	90	5	1

Table 7-2: Number of generations to reach the target in a set of tests of our Grammatical Evolution approach.

Since the algorithms use random numbers, different random seeds give different results. We have thus obtained sets of fractal curves, sometimes quite different in appearance, that share the same fractal dimension. Table 7-3 shows some results for a target dimension of 1.255 and an angle of 60 degrees. In all of them, the minimum fitness was set to 1000 (which corresponds to an error in the target dimension below 0.001). The dimension of all the results came to be 1.2549. This fractal dimension has been computed without considering possible overlappings of the curves with themselves. A definition of dimension that would take this into account could also be considered [Alf00a], [Alf01a], at the cost of longer computation times, and perhaps

more generations. Figure 7-2 displays the fractal curves, approximated by the fourth derivation of the corresponding L systems.

Number of generations	Size of genotype	L System word developed	Axiom
4	16	-F+FF+FF-	F--F--F
44	7	F-F++F-F+F-	F++F++F
72	8	FF--F+FF+	F--F--F
255	8	FF-FF++F-	F++F++F

Table 7-3: Different fractal curves sharing the same dimension, evolved by our method.

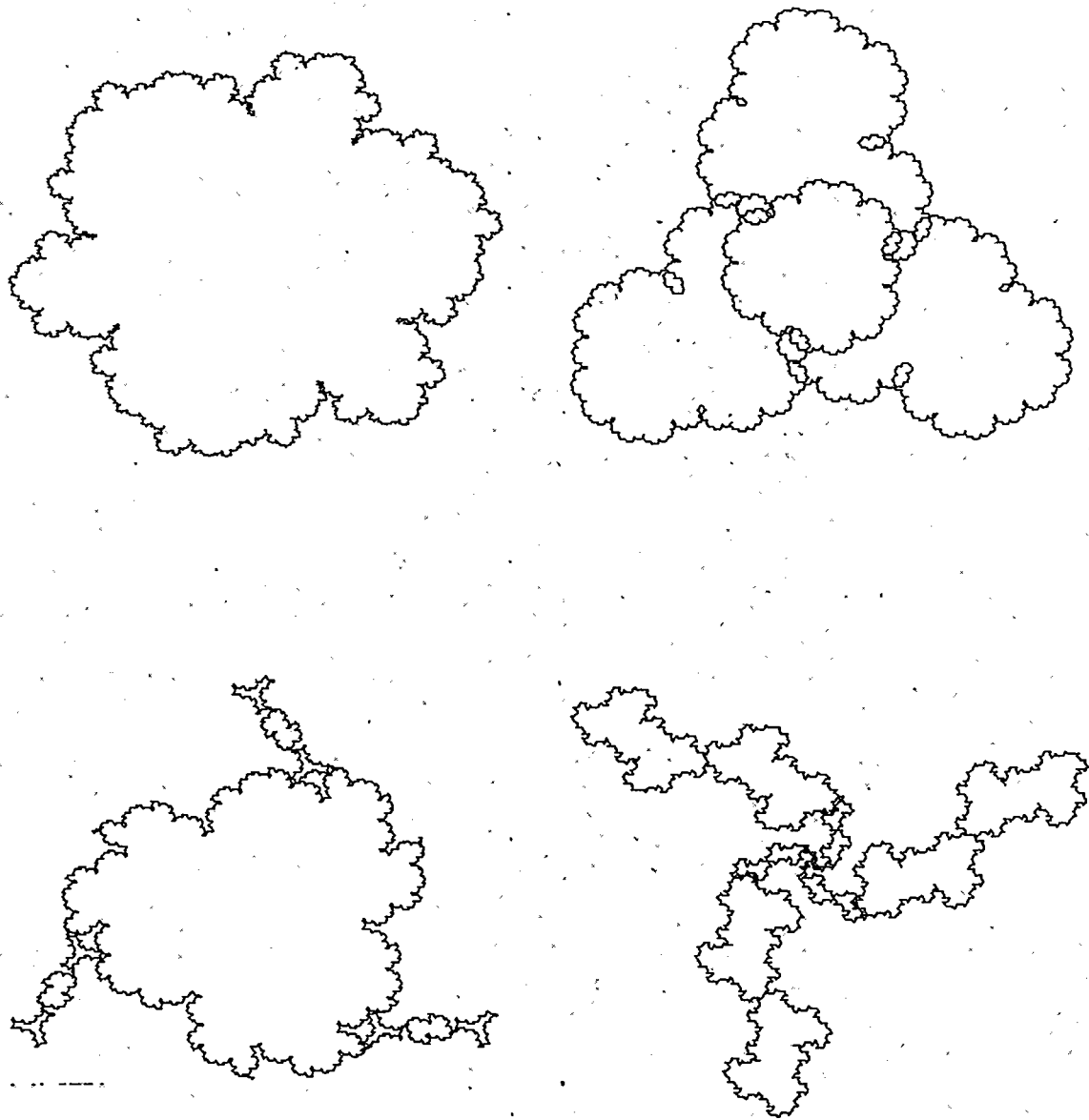


Figure 7.2: Four different fractal curves evolved for a target dimension of 1.255



Table 7-4 shows a few interesting fractals evolved by means of our algorithms. The first one has the same dimension as von Koch's snowflake, but with an angle of 36 degrees rather than 60. Figures 7-3 to 7-9 display the fractal curves, approximated by the third derivation of the corresponding L systems.

Dimension	Angle	Number of generations	Size of genotype	L System word developed	Axiom	Shown in figure
1.26178...	36	27	35	+F+F---F-F+F-F+F+	F-F-F-F-F-F-F-F-F-F	10.3
1.5018	45	2000	17	++F-F--F+F+F +FF---F-F++	F---F++F---F-F--F	10.4
1.8998	40	1460	41	++F-FF-F-F---F-F---F-- F-F++	F+F+F+F+F+F+F+F+F+F+	10.5
1.8008	45	2378	30	F-F-F-FF-F-F--F-F--F-F- --F -FF-F-F+F+FF-	F---F++F---F-F--F	10.6
1.7005	60	18	22	+++FF+F+FFF+F+F+FFF +FFF+F+F-F+F+---	F-F-F-F-F-F-F	10.7
1.6006	45	275	30	+F-F--F--F+FF--F+F-F- FF+FF-F--F	F++F++F++F	10.8

Table 7-4: A few fractal curves evolved by our method.

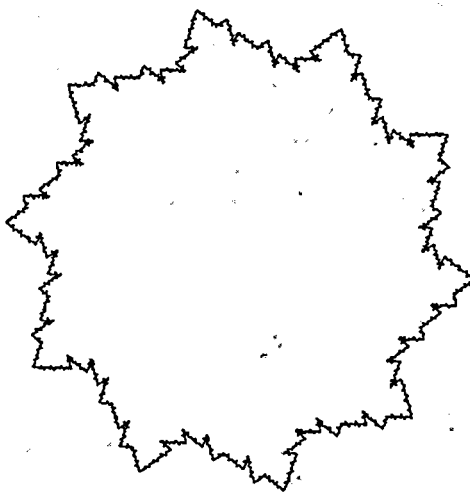


Figure 7-3: A fractal curve with the same dimension as von Koch's snowflake.

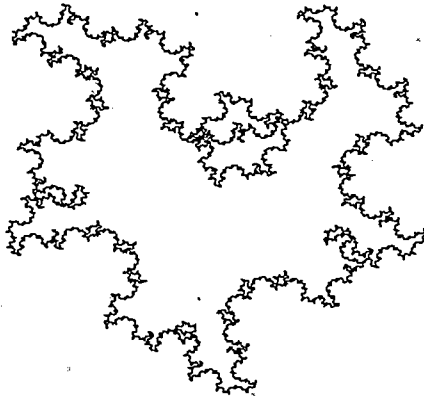


Figure 7-4: A fractal curve with approximate dimension 1.5.

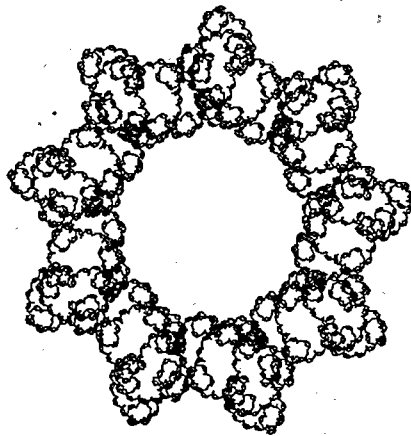


Figure 7-5: A fractal curve with approximate dimension 1.9.

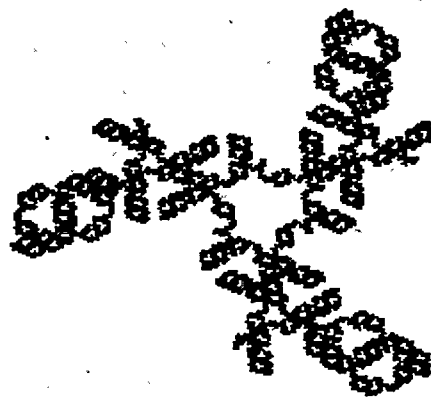


Figure 7-6: A fractal curve with approximate dimension 1.7

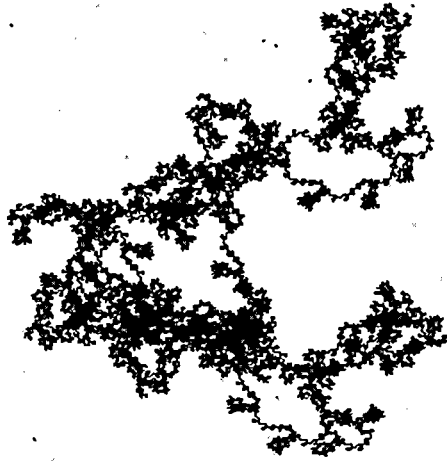


Figure 7-7: A fractal curve with approximate dimension 1.8.

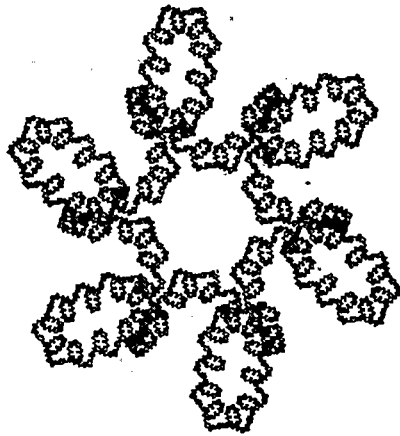


Figure 7-8: A fractal curve with approximate dimension 1.7.

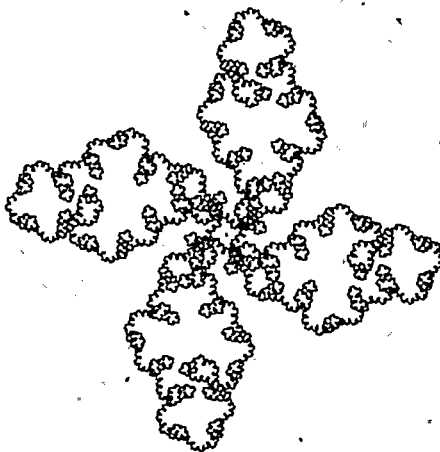


Figure 7-9: A fractal curve with approximate dimension 1.6.

Table 7-5 shows finally some of the results obtained with our second experiment, where the turtle angle itself was subject to evolution by means of the genetic algorithm.

Target dimension	Actual dimension	Number of generations	Size of genotype	Angle evolved
1.5	1.5	5	8	90
1.5	1.5	8	14	45
1.5	1.4999	64	19	90
1.5	1.4999	176	21	90
1.5	1.5	50	9	90
1.5	1.5009	940	124	36
1.5	1.5008	1337	86	36
1.5	1.5	68	9	90
1.9	1.8992	1069	55	72
1.9	1.8993	1306	42	24
1.9	1.8998	1460	41	40

Table 7-5: A set of tests where the turtle angle was evolved too.

7.8. Examples

In this section we will show two different examples. The first one demonstrates how the proposed algorithm evolves an L-system with a given angle and dimension. The second example repeats the example when the angle must be evolved too.

Example 1:

In table 7-6, we can see the evolution of the algorithm when its target is finding a curve with a dimension equal to 1.58496 (the same as Sierpinski's gasket), and the angle to be used is 60 degrees.

Generation no	Fractal dimension	L-system grammar
1	1.424828748	+F-+F+F+F
2	1.424828748	-F+F--F++F-
3	1.424828748	-F+F--F++F-
4	1.424828748	-F+F--F++F-
5	1.424828748	-F+F--F++F-
6	1.424828748	-F+F--F++F-
7	1.4924572	-+-+FF-FF-+++FF+FF-F
8	1.654174951	+F+F--F++F+F
9	1.584962501	+F-++F+F

Table 7-6: Execution of the genetic algorithm for fractal dimension 1.58496 and angle 60 degrees.

Figure 7-10 shows the axiom for the L-system.

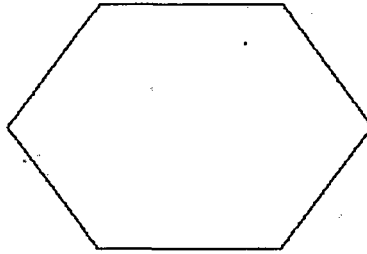


Figure 7-10: The initiator object

The iterator for Sierpinski fractal is shown in figure 7-11.

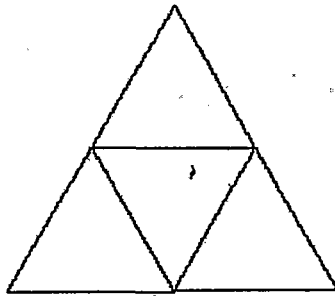


Figure 7-11: Sierpinski's iterator

Figure 7-12 shows the curve obtained from the preceding initiator and iterator, after 9 iterations of the process.

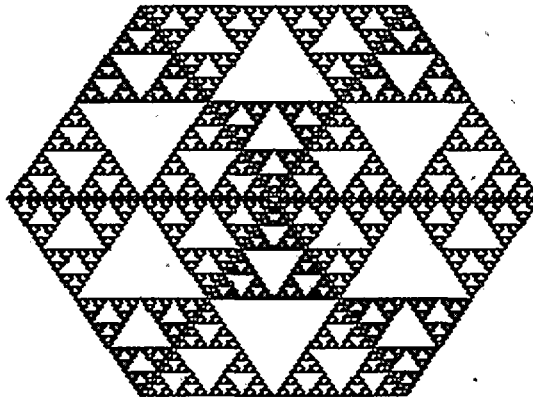


Figure 7-12: Sierpinski fractal after 9 iterations

The iterator for the first L-system found by the algorithm, which has the rule $F ::= +F - +F + F + F$, is graphically represented in the figure 7-13.



Figure 7-13: The first iterator approximation.

Figure 7-14 shows the corresponding curve after 6 iterations.

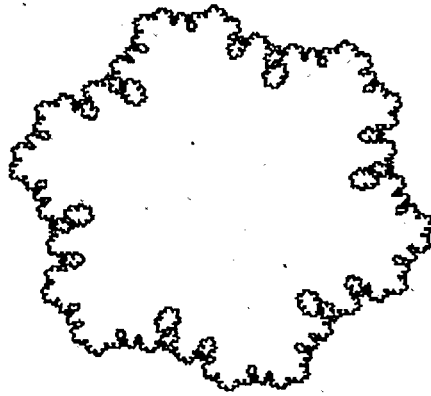


Figure 7-14: The first fractal approximation after 6 iterations

From generation 2 to generation 6, a second L-system is the best approximation ($F ::= -F+F--F++F-$). Figure 7-15 shows the iterator.

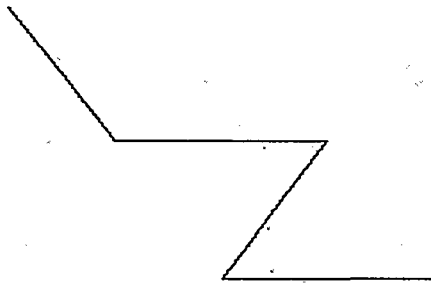


Figure 7-15: The second iterator

After 6 iterations, the curve obtained is shown in figure 7-16.

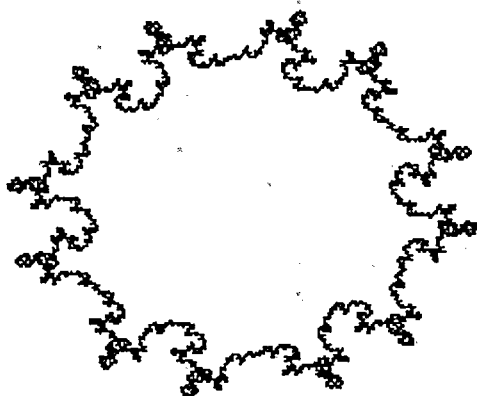


Figure 7-16: The second fractal after 6 iterations

Figure 7-17 shows the iterator of the best individual after 7 generations. It has the derivation rule $F ::= -+ +FF -FF -+++FF +FF -F$, that corresponds to the following iterator:

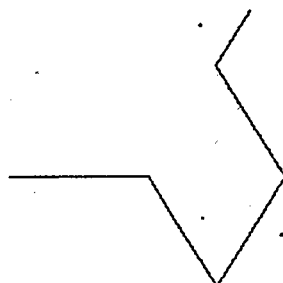


Figure 7-17: The third iterator

Figure 7-18 shows the third approximation after 4 iterations.

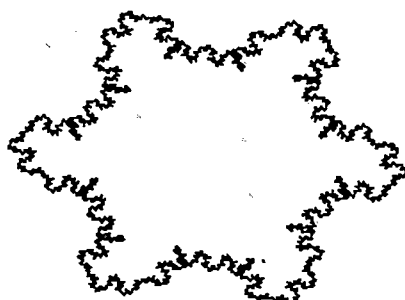


Figure 7-18: The third fractal after 4 iterations

In generation number 8, the best L-system has the derivation rule $F ::= +F +F --F ++F +F$, which represents the iterator in figure 7-19.

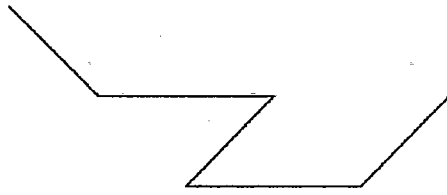


Figure 7-19: The fourth iterator

Figure 7-20 shows the fourth approximation after the fifth iteration.

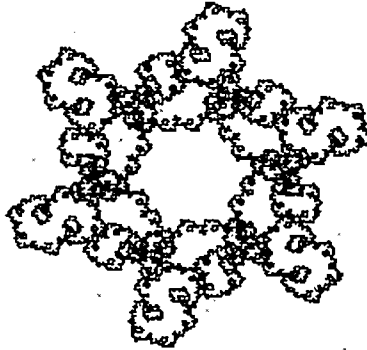


Figure 7-20: The fourth fractal after 5 iterations

In generation number 9, the best production rule is $F ::= +F-++F+F$, which is a solution to the problem (its fractal has dimension is 1.584962501). Figure 7-21 shows the iterator.

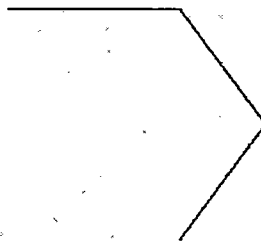


Figure 7-21: The final iterator

Which, after been applied 9 times to the initiator, produces the curve in figure 7-22.

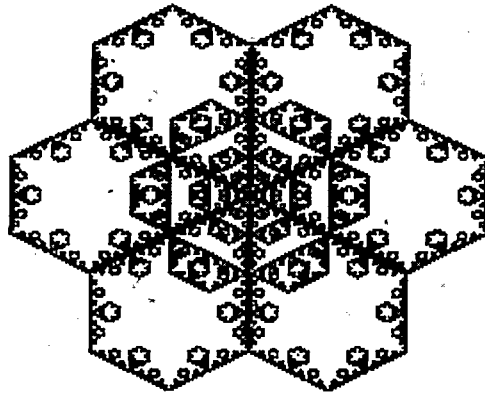


Figure 7-22: The final fractal after 9 iterations

Example 2

In this example, the algorithm also evolves the angle of the graphic interpretation. The target dimension is again 1.584962501. The result of the evolution process is shown by table 7-7.

Generation n ^a	dimension	Angle	L-system
1	1.424828748	120	F-FF+-F+
2...4	1.464973521	90	F+FFF+F--
5...11	1.654174951	120	-FF+F+-F-F+
12...57	1.556302501	90	F+FF+FF-F-
58...66	1.562756238	90	F+FF+F+-F+F+F+FF+F-F+FF+F--F+F -----
67...100	1.606876736	90	F+F+F+FF+F+F+F+-F+FF+F++F+F --F+FFFF--F-----
101...138	1.593692641	90	F++FF-+-+FFFF++-F--F+FF+FF+FF---
139...145	1.59327954	90	-F+F+FF+-+--FF++FF---FF+F+-F+F- F+
146	1.584962501	90	-F+F+FF+-+--FF++FF--+-FF+F--F+F- F+

Table 7-7: Genetic algorithm execution for fractal dimension 1.584962501

The following figures show the steps of the process, in the same way as in example 1.

- Figure 7-23: the best iterator after 1 generation, angle 120.

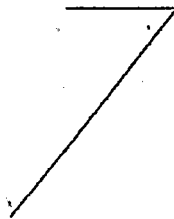


Figure 7-23: The first iterator approximation

- Figure 7-24: curve produced after 5 iterations.

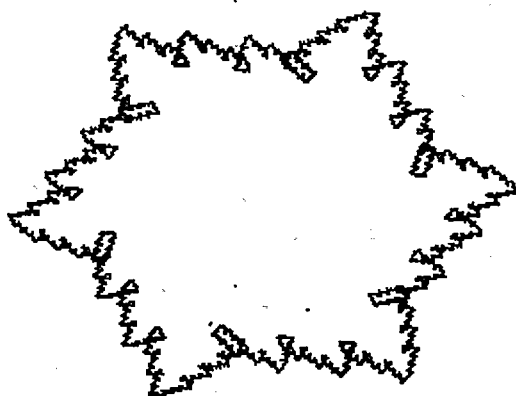


Figure 7-24: The first fractal after 5 iterations

- Figure 7-25: Best iterator from generations 2 to 4, angle 90.



Figure 7-25: The second fractal iterator

- Figure 7-26: The curve produced after 6 iterations.

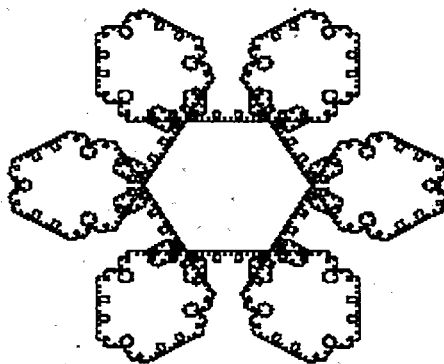


Figure 7-26: The second fractal after 6 iterations

- Figure 7-27: Best iterator from generations 5 to 11, angle 120 degrees.

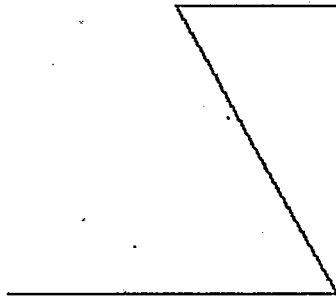


Figure 7-27: The third fractal iterator

- Figure 7-28: The curve produced after 6 iterations.

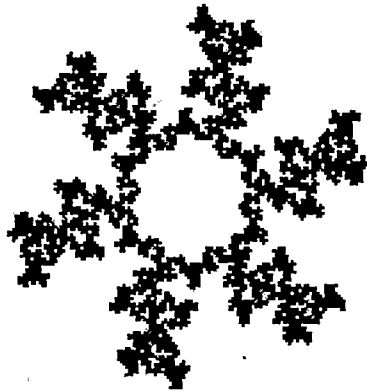


Figure 7-28: The third fractal after 6 iterations

- Figure 7-29: The best iterator from generations 12 to 57, angle 90.

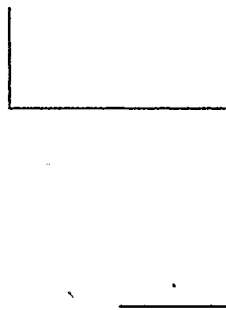


Figure 7-29: The fourth fractal iterator

Figure 7-30: The curve produced after 6 iterations.

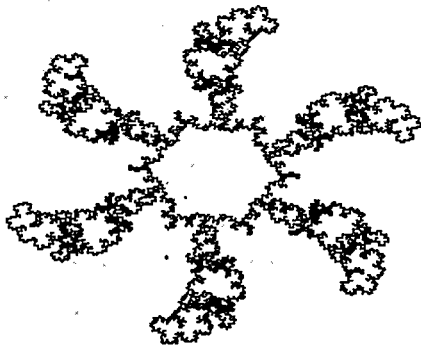


Figure 7-30: The fourth fractal after 6 iterations

- Figure 7-31: Best iterator from generations 58 to 66, angle 90.

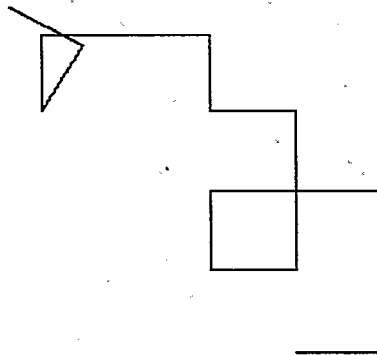


Figure 7-31: The fifth fractal iterator

- Figure 7-32: The curve produced after 4 iterations.

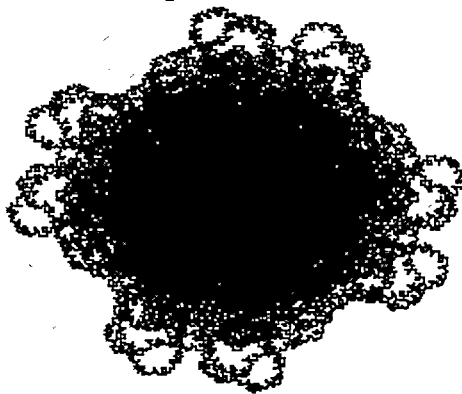


Figure 7-32: The fifth fractal after 4 iterations

- Figure 7-33: The best iterator from generations 67 to 100, angle 90.

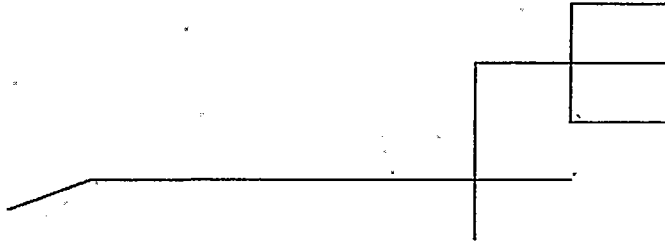


Figure 7-33: The sixth fractal iterator

- Figure 7-34: The curve produced after 3 iterations.

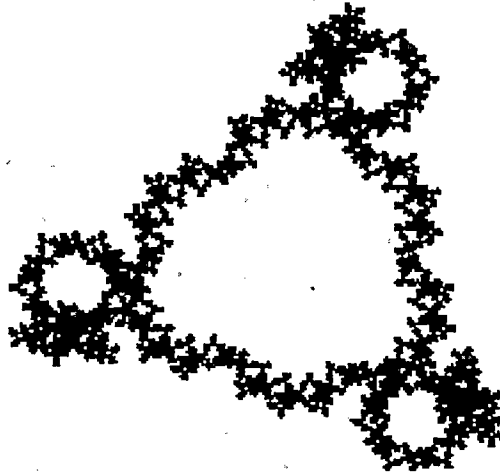


Figure 7-34: The sixth fractal after 3 iterations

Figure 7-35: The best iterator from generations 101 to 138, angle 90 degrees.

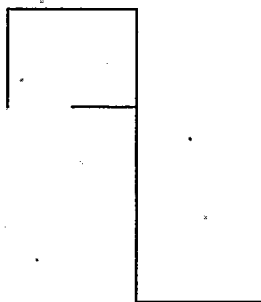


Figure 7-35: The seventh fractal iterator

- Figure 7-36: The curve produced after 4 iterations.

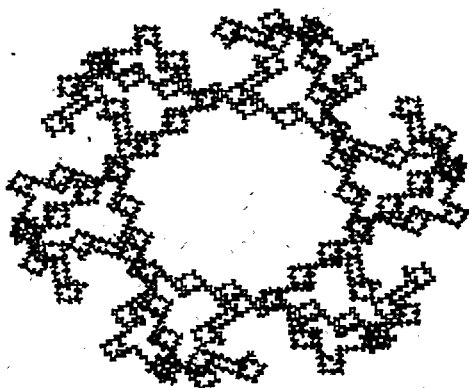


Figure 7-36: The seventh fractal after 4 iterations

- Figure 7-37: The best iterator from generations 139 to 145, angle 90.



Figure 7-37: The eighth fractal iterator

- Figure 7-38: The curve produced after 3 iterations.

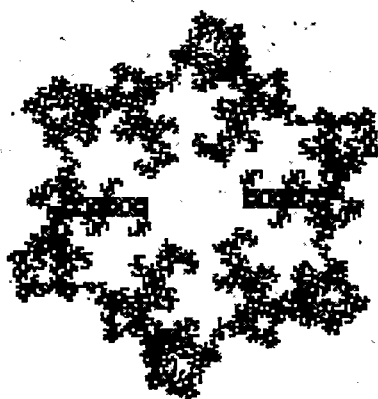


Figure 7-38: The eighth fractal after 3 iterations

- Figure 7-39: In generation 146 we obtain the curve with the target dimension (1.584962501) and angle equal to 90 degrees.

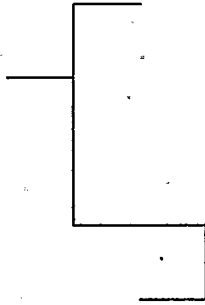


Figure 7-39: The final fractal iterator

- Figure 7-40: The final curve produced after 3 iterations.

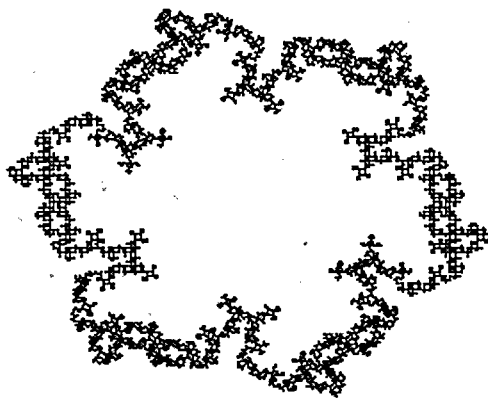


Figure 7-40: The final fractal after 4 iterations

Chapter 8

Evolving the game of life with a genetic algorithm

8.1. Overview

In chapter 5 we saw how cellular automata are evolved by means of genetic algorithms to perform a predefined specific task. In this chapter we propose a genetic algorithm that evolves two-dimensional cellular automata in a fast way, following a direct path from an arbitrary cellular automaton to Conway's game of Life.

A genetic algorithm may be used to evolve the transition rules of a cellular automaton. The algorithm we propose reaches the perfect target in about 25000 generations. Different grid sizes have been tested, but in most of the experiments the grid size of the automata has been restricted to a given finite size (an 8x8 matrix).

The genetic algorithm uses the two standard genetic operators: crossover and mutation. A large mutation rate has been found appropriate to speed-up the process, specially when both progenitors are identical. Random density in the initial states happens to be worse than pure randomness in this case. The algorithm performance is relatively independent of the grid size.

We begin, in Section 8.2, by defining the cellular automata that will be evolved, called *life-related cellular automata*. Section 8.3 describes the genetic algorithm, and Section 8.4 introduces the experiments performed by our genetic algorithm and the results obtained.

8.2. Concise representation of life-related cellular automata

The cellular automata which will be evolved, called *life-related cellular automata*, is defined as the set of cellular automata that comply with the following rules:

- The grid is rectangular.
- The set of neighbors to a point in the grid consists of the point itself plus the eight adjacent points in the eight main directions in the compass (Moore's neighborhood).
- Each finite automaton has two states, represented by the Boolean numbers {0,1}.
- The transition function of the finite automaton associated to every point is deterministic.

Life-related cellular automata differ in the transition functions of their finite automata. Since the range of the transition functions is made of the input to the automata (the eight states of their neighbors in a given order) and their own state, each possible value in the range can be represented by a nine-bit Boolean vector or, alternatively, by a number in the $[0,511]$ interval. Each member in the set of life-related cellular automata may thus be represented by its associated transition function, which can be expressed as a Boolean vector with 512 elements, giving the next state of the automaton for each of the possible 512 range values. This means that the number of possible different transition functions is 2^{512} , or approximately 10^{154} , an unimaginably large number.

8.3. A genetic algorithm that evolves into the game of Life

We have used a genetic algorithm to obtain Conway's game of Life by evolution from a random population of life-related cellular automata, taken arbitrarily from the full set of 10^{154} members. The grid has been restricted to an 8x8 square matrix. Evolution is fast, and reaches perfect target in a few tens of thousand steps or generations.

The algorithm can be described as follows:

- 1 Create 60 random life-related cellular automata, where every random life-related cellular automaton consists of 64 binary automata.
- 2 Choose random initial conditions for the 64 automata in the 8x8 grid. Two different initial condition families have been compared: uniform random and random density.
- 3 Compute the result of executing a step in Conway's game of Life with the chosen initial conditions, using any standard implementation. We used an APL2 program that executes the game of life on the finite rectangular grid. This results in an 8x8 Boolean matrix.
- 4 Compute the result of executing a step in each of the 60 life-related cellular automata with the chosen initial conditions. All the results are also 8x8 Boolean matrices.
- 5 Compare each of the results in step 4 with the result of step 3 and assign a fitness value to each of the 60 life-related cellular automata. The fitness value (an integer in the $[0,64]$ interval) is the number of coincidences between the elements of the two 8x8 Boolean matrices.
- 6 Order the 60 life-related cellular automata in the order of their fitness values.
- 7 The ten automata with top fitness values are paired two-by-two and reproduce, each pair generating two new automata, which replace the ten automata with bottom fitness values. The reproduction algorithm uses the two standard tools in genetic algorithms: mutation and cross-over at a random point (genetic recombination of the two 512 Boolean vectors that represent the transition function of the two parent automata). Different mutation rates have been tested.

8 Compute statistics from the 60 current automata.

9 Go to step 2.

Initially, the average fitness values of the 60 automata compute around 32, which is the number of coincidences expected from two 8x8 random Boolean matrices. As evolution proceeds, the average fitness values increase towards 64, although the actual values in a given step depend on the random initial conditions used in that step. As mutations and cross-over generate new automata, more and more of them get increasingly larger fitness values.

We implemented the whole process in the APL2 language, which is very concise and appropriate for this application: each step in the above description of the algorithm can be implemented by a single instruction. Additional auxiliary functions are used to implement the standard game of Life (10 lines of code [Alf99]), the random density generator of initial conditions (4 lines), the execution of a life-related cellular automaton (8 lines), and the reproduction of a pair of automata (less than 20 lines).

8.4. Experiments and results

We have tested both random density and pure random values selected from the $\{0,1\}$ set for the initial condition matrices, to detect possible differences between both families. In classical experiments with one-dimensional cellular automata, random density has been found to be significantly better.

We have also experimented with different mutation rates. In standard genetic algorithms, the mutation rate is usually quite low, around 0.5 percent per bit. However, higher mutation rates are used when both parents have the same genotype, because in that case cross-over does not have any effect.

In our case, we have used higher mutation rates applicable to a certain number of bits in each genotype. Different rates have been tested, and it is also better to use a higher rate when both parents are identical than otherwise.

Figure 8-1 displays the number of differences between the best cellular automaton and the target (the game of Life) as a function of the number of generations. It may be seen that convergence follows a Poisson curve, being quite fast at the beginning, and slowing towards the end.

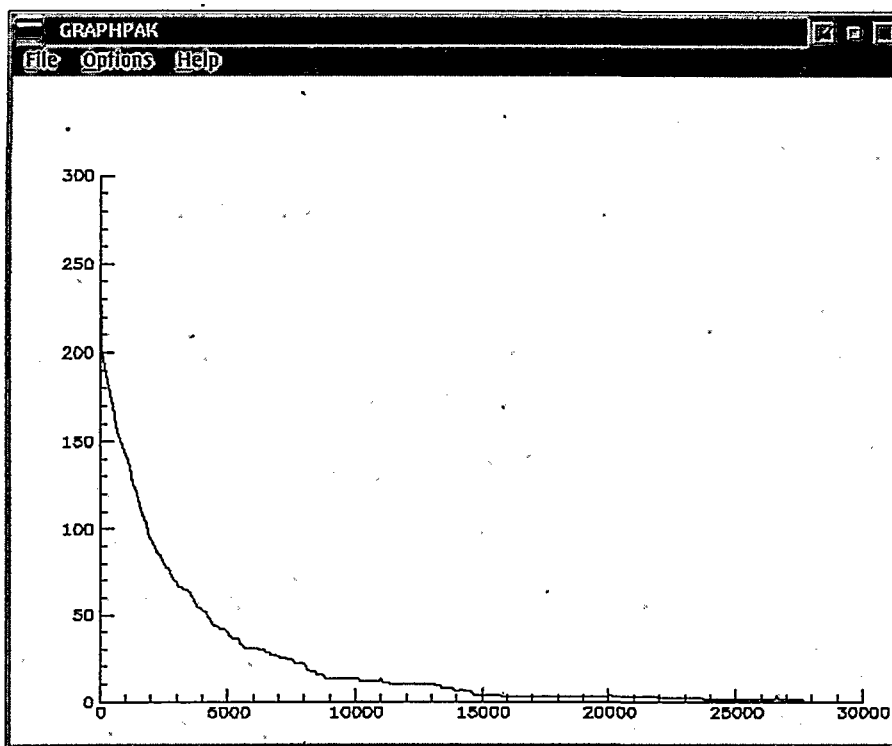


Figure 8-1: Number of differences of best evolved automaton and Conways game of life.

As can be seen, the difference between the evolved life-related cellular automata and the target Conway game of life is, at the beginning, higher than 200. When the number of generations approach 25000, the difference falls to zero, meaning that the best cellular automata has been found equal to the game of life.

Mutation rate		Initial conditions	
Equal parents	Different parents	Uniform random	Random density
10%2	1%1	55150	>120000
20%5	1%1	27677	>44000
20%5	5%2	25826	>33000
20%5	10%3	30242	108727
20%5	20%5	34648	72438

Table 8-1: Number of generations to reach the goal, for different settings in the mutation rates and the initial conditions.

Table 8-1 summarizes the results obtained for different configurations. An unexpected result is the fact that a uniform random distribution of ones and zeros in the initial condition matrices gives better results than a random density distribution, which is usually assumed to work better in these situations.

It is also interesting to notice that large mutation rates are advantageous in this experiment. The $x\%y$ code means that x percent of the children suffer y simultaneous random mutations in their genotypes. When both parents are different, the optimal mutation rate seems to be around 5 percent.

At the beginning of the process, many different automata coexist and the increase in the mutation rate for different parents produces the highest effect. Later, when many automata in the population have converged to the same genotype, the effect of the increase in the second column of the table is less notorious.

We have done some tests varying the size of the grid. When an $n*n$ matrix is used, the execution time of the algorithm per generation increases in proportion to the size of the grid, but the number of generations needed to reach a given goal (an automaton whose function differs 10 percent from the game of Life) decreases, which to some extent compensates the increase. Table 8-2 shows the results obtained.

Notice that the optimal size for faster convergence happens when a 10x10 grid is used.

Grid size	secs/generation	Nr. of generations	Total time (secs)
6x6	0.39	11800	4602
8x8	0.60	4300	2580
10x10	0.88	2700	2376
16x16	1.87	1500	2805

Table 8-2: Convergence speed as a function of grid size.

As we see in table 8-1, the genetic algorithm reaches successful convergence in about 25000 generations, starting from an initial set of 60 cellular automata, chosen randomly from a set of 2^{512} . The total execution time has been found to be relatively independent of the size of the grid, with the optimal size around 10x10.

The total number of automata generated in a run of the genetic algorithm with 25000 steps is about 250000 (ten new automata are generated in each step), much smaller than the size of the set of life-related cellular automata. This means that the algorithm takes a very direct search path towards the target.

Chapter 9

Cellular automata equivalent to PD0L systems

9.1. Overview

In this chapter we study the opposite direction to the one described in chapter 6. We present a method to build a one-dimensional cellular automaton associated to a given PD0L system. As we saw in chapter 3 (section 3.2.1), a PD0L system is a D0L system where no symbol may be transformed into the empty word. Our cellular automata produce the same words and in the same order as the given PD0L systems for a finite number of derivations.

Other authors have previously tackled this problem. As already said in Chapter 6, [Terr91] show that there exist interactive one-dimensional cellular automata able to generate signals with frequencies equal to the growth functions of PD0L systems. These cellular automata are designed by means of sets of signals able to solve the problem.

In a similar way, we design one-dimensional cellular automata equivalent to given PD0L systems but our approach differs from [Terr91] in several main features:

- Our cellular automata are not interactive; no input is needed apart from the initial configuration.
- We are not interested in the growth functions, but in the generation of the same languages, that is, our cellular automata generate the same words and in the same order as the PD0L systems.
- The description of the behavior of our cellular automata in terms of signals is also possible; we are mostly interested in the explicit definition of the whole cellular automata.

There is no constraint to the PD0L system considered, so the method is a general algorithm and can be used as a proof for an equivalence theorem.

9.2. One-dimensional cellular automata

In this section we define the one-dimensional cellular automata that will become the result of the equivalence algorithm.



9.2.1. Informal description

A one-dimensional cellular automaton is a chain of automata in a straight line. The neighborhood relationship in one-dimensional automata defines a set of predecessors and successors. One of the better-studied neighborhoods for one-dimensional automata consists of the automaton itself and its two nearest neighbors.

9.2.2. Formal definitions

Given a set E , a one-dimensional grid on E is a function

$$G: Z \rightarrow E$$

And $G[i]$ or G_i is the element of E at the i^{th} position in the grid.

R_∞ on E is the set of one-dimensional infinite grids on E .

A one-dimensional neighborhood V is a pair

$$(k, N)$$

where

- $k \in \mathbb{N}$ is the number of neighbors of every automaton in the grid.
- $N \in Z^k$ is a vector of k integer offsets. Given the index of a position in the grid, each offset points to a different neighbor of the automaton in this position.
- The predecessor / successor neighborhood is formalized as $V_{p/s} = (3, (-1, 0, 1))$.

A one-dimensional deterministic cellular automaton is the six-fold

$$(G, G_0, V, Q, f, T)$$

where

- G is a one-dimensional grid of automata.
- Q is the finite and non-empty set of possible states of the automaton in the grid.
- G_0 is the initial configuration. It is a mapping

$$G_0: G \rightarrow Q$$

that assigns an initial state to each automaton in the grid.

- $V = (k, N)$ is a one-dimensional neighborhood.

- $f: Q \times Q^k \rightarrow Q$ is the transition function that assigns the next state of each automaton in the grid depending on its actual state and the states of its k neighbors.
- T is the discrete time.

9.3. One-dimensional cellular automata n-equivalent to PDOL systems

9.3.1. Informal description

In this section the design of a cellular automaton equivalent to a given PDOL system is tackled. A cellular automaton is said to be n-equivalent to an L system if it is possible to find the n first words in the language generated by the L system in a finite number of successive configurations of the cellular automaton. That is, there must be a way to link:

- The axiom of the L system and the initial configuration of the cellular automaton.
- Each subsequent word in the language of the L system and a configuration of the cellular automaton, provided that the order of the derivation is preserved. That is, the configuration corresponding to the first derived word must be obtained before the configuration corresponding to the second derived word, and so forth.

Figure 9-1 shows the way in which the words derived by the L-system are linked to the configurations generated by the one-dimensional cellular automata, by means of a link mechanism ψ .

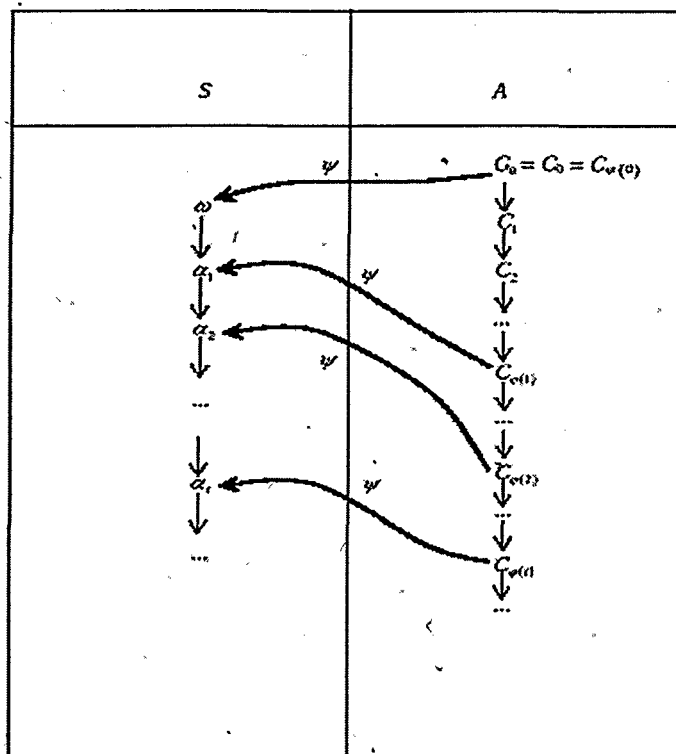


Figure 9-1: Definition of Cellular automata equivalent to PDOL systems

For clarity, the way in which the cellular automaton simulates the PDOL system is explained with an example:

Let us look at the following PDOL system S

$$S = \{ \Sigma = \{A, B\}, P = \{A ::= BB, B ::= AB\}, A \}$$

In order to build the equivalent one-dimensional cellular automaton the following decisions have been taken:

- There will be a cell in the linear grid of the cellular automaton to contain each symbol of the words derived by the L system.
- The states of the automaton must provide a mechanism to insert new symbols in a given position and to move the old ones to their final place, because the words increase their length as they are derived by the L system. So, the states of the automaton must take into account not only the symbol that will finally contain the cell, but also the substrings that will be displaced across the cell.
- As a consequence of the previous point, the beginning and the end of the word must be marked. The symbols $>$ and $<$ are, respectively, the left and right marker.
- If a generation only contains a set of contiguous cells embraced by the left and right markers and every cell that has not a marker has an empty displacing substring, then

the generation contains a word derived by the L system. The symbol \diamond (blank) represents an empty displacing substring.

Figure 9-2 show the initial generation of the cellular automaton. The symbol contained in each cell appears over the displacing substring. This configuration represents the axiom (A) because there is only one cell that has not a marker and it has an empty displacing substring.



Figure 9-2: Initial generation of the cellular automaton

The behavior of the cellular automaton can be summarized as follows:

- First, the right marker is transmitted to the left until it reaches the left marker. When this condition is reached, the cellular automaton is ready to begin the production process.
- Second, the production rules are applied in the opposite direction. At the same time, the symbols of the new word are displaced until they find their final position.
- At this moment, the new word has been derived and the automaton is ready for a new derivation.

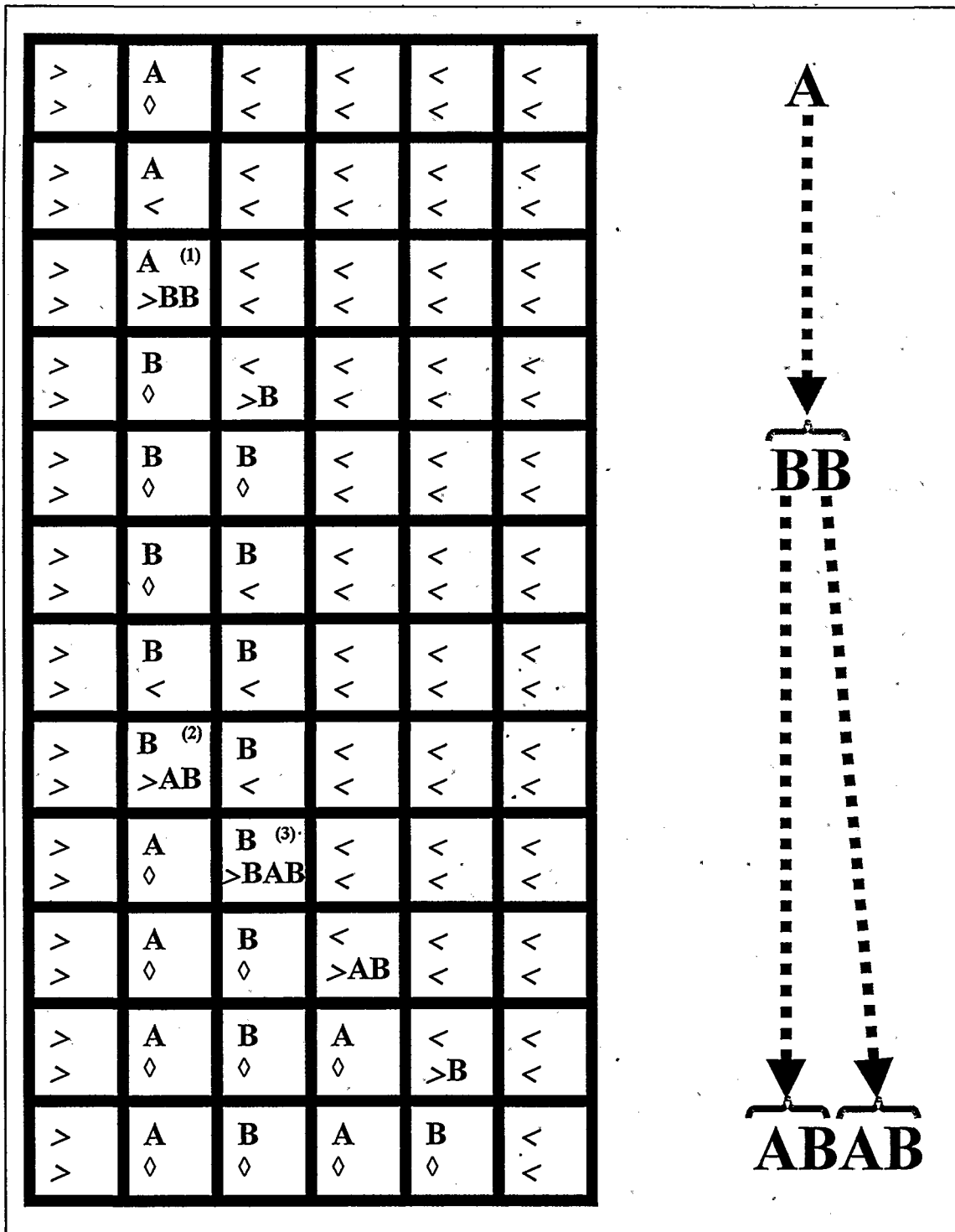


Figure 9-3: Cellular Automaton – D0L system comparison: the first 12 generations of the automaton simulate the first two derivations of the L system. The automaton subsequent states are shown in the left side

The following conditions describe the different phases of the automaton:

- The propagation of the right marker is shown in every cell that contains a symbol distinct from the markers (\langle, \rangle) and a displacing substring equal to " \langle ".
- After the application of a production rule, the new symbols must reach their final position. Cells that contain the symbol \langle (the right marker) and a displacing substring that begins with the left marker (\rangle) show this situation.
- The production rules are applied in cells that contain a non-marker symbol and a displacing substring that begins with the left marker (\rangle). Numbers (1) and (2) label, respectively, the cells in which the production rules $A ::= BB$ and $B ::= AB$ are applied. Number (3) is associated to the simultaneous application of the production rule $B ::= AB$ and the movement of some symbol towards its final position.
- To identify a cell that contains a symbol that is in its final position, an empty displacing substring (\diamond) is used.

Time is supposed to increase downwards. As it can be observed, when symbols are displaced they propagate the left marker (\rangle). In a displacing substring, the first symbol after the left marker will be finally placed in the cell containing the substring. So the next state of this cell represents a symbol that has found its final position.

The next state of each cell depends only on its left and right neighbors.

This method is generalized and formalized in the following section.

9.3.2. Formal description

9.3.2.a. Theorem

Given a PDOL system

$$S = (\Sigma, P, \omega)$$

where all the components have been previously defined, and being $L(S, n)$ the set of the first n words generated by S , starting at and including ω , there exists a one-dimensional cellular automaton whose sequence of states contains $L(S, n)$ in the same order. This automaton is n -equivalent to S .

9.3.2.b Proof

Our proof is constructive. In the following paragraphs, a one-dimensional cellular automaton (A) that generates the same language that a given PD0L system (S) is built.

To make the proof clearer, the following notation will be used:

- $\Sigma_{m_l} = \{>\}$. Σ_{m_l} is the alphabet of the left marker.
- $\Sigma_{m_r} = \{<\}$. Σ_{m_r} is the alphabet of the right marker.
- $\Sigma_{e_m} = \Sigma \cup \Sigma_{m_l} \cup \Sigma_{m_r}$. Σ_{e_m} takes its name from alphabet Σ extended with the markers.
- $\Sigma_e = \Sigma \cup \Sigma_{m_l} \cup \Sigma_{m_r} \cup \{\diamond\}$. Σ_e takes its name from alphabet Σ_{e_m} extended with the blank space.
- $\Sigma_{e,k} = \bigcup_{i=1}^k \Sigma_e^i$
- $\Sigma_k = \bigcup_{i=1}^k \Sigma^i$

The cellular automaton we want to build is one-dimensional. So, its grid (G) is a vector of finite and deterministic automata $G \in R_\omega$ on the set of deterministic automaton induced by the cellular automaton being defined.

The predecessor / successor neighborhood ($V_{p/s} = (3, (-1,0,1))$) is used.

As it was suggested by the informal description, each automaton must be able to contain the following information:

- A symbol from the alphabet of the PD0L system plus the end markers.
- A displacing string of the same kind of symbols.

Thus, each possible state of the cellular automaton will be a pair formally defined as follows

$$Q \subseteq \Sigma_{e_m} \times \Sigma_{e,k+i}$$

Where

- $k \leq \max_{A \in \Sigma} \{|\rho(A)|\} \times |h^k(\omega)|$, where max represents the maximum.

- Pairs ($\triangleright, \triangleright$) and ($\triangleleft, \triangleleft$) are used respectively as left and right markers that fill the portion of the grid that remains unused.
- Pairs (A, \triangleleft) ,, $A \in \Sigma$ are used to propagate the right marker.
- Pairs (A, \diamond) ,, $A \in \Sigma$ are used when the symbol A reaches its final position.
- Pairs ($B, \triangleright \alpha$) ,, $B \in \Sigma \cup \{\diamond\} \wedge \alpha \in \Sigma_k$ are used to move the displacing substrings.

The cellular automaton initially shows the axiom of the D0L system. 0 will be considered always the index value corresponding to the first symbol of the words. So G_0 is defined as follows:

$$G_0(i) = \begin{cases} (\omega_j, \diamond) & \text{if } i = j \\ (\triangleright, \triangleright) & \text{if } i < 0 \\ (\triangleleft, \triangleleft) & \text{if } i \geq |\omega| \end{cases}$$

The set of final states is empty:

$$T = \Phi$$

The transition function was informally described in the previous paragraphs and is formally defined by cases as follows:

$$\forall A, U \in \Sigma \wedge \forall L, R, S \in \Sigma_e \wedge \alpha, \beta \in \Sigma_k \wedge \delta \in \Sigma^+, \gamma \in \Sigma_k$$

$$1. f((L, \alpha), (\triangleright, \triangleright), (R, \beta)) = (\triangleright, \triangleright)$$

This case maintains the left filler unchanged.

$$2. f((L, \triangleright U), (\triangleleft, \triangleleft), (R, \beta)) = (\triangleleft, \triangleleft)$$

This case maintains the right filler unchanged.

$$3. f((L, \alpha), (S, \diamond), (R, \triangleleft)) = (S, \triangleleft)$$

This case propagate the right marker to the beginning of the grid.

$$4. f((L, \alpha), (S, \triangleleft), (R, \beta)) = (S, \triangleleft)$$

The two previous cases propagate the right marker.

$$5. f((L, \triangleright U \delta), (A, \triangleleft), (R, \beta)) = (A, \triangleright \delta \rho(A)) \text{ ,, } A ::= \rho(A) \in P$$

This case applies the derivation rule $A ::= \rho(A)$ to the symbol A, while simultaneously moving the displacing string from its left (δ).

$$6. f((L, >), (A, <), (R, \beta)) = (A, > \rho(A)) \text{ , , } A ::= \rho(A) \in P$$

This case applies the derivation rule $A ::= \rho(A)$ to the symbol A.

$$7. f((L, > U \delta), (<, <), (R, \beta)) = (<, > \delta)$$

This case moves the displacing string from the left (δ).

$$8. f((L, \alpha), (S, > U \gamma), (R, \beta)) = (U, \diamond) \text{ where } \gamma \text{ may be the empty string } (\lambda)$$

This case stops symbol U in its final position.

$$9. f((L, \alpha), (A, \diamond), (R, \beta)) = (A, \diamond) \text{ where } \beta \neq <$$

This case keeps symbol A in its final position.

The behavior of the cellular automaton can be divided in the following phases:

1. Propagation of the right marker:

- The initial configuration contains automata that have only three kinds of states: ($>, >$), (S, \diamond) and ($<, <$).
- The first and third states are maintained respectively by means of the first and second cases in function f . The only case of f that can be applied to the states of the kind (S, \diamond) is the third, that initiates the propagation of the right marker. Further steps of the propagation are done by the fourth case. This case records that the right marker has already passed this position.

2. The production rules are applied and the resulting symbols look for their final position:

- When both markers meet at the beginning of the word, the propagation of the right marker finishes. When this condition holds, there is a rule that can be applied. This is accomplished by the sixth case of f . So the application of the rules begins where the propagation of the right marker finishes, and proceeds along the word from left to right.
- Whenever a rule is applied, a substring is placed in a cell that will finally hold a single symbol: only the first symbol of the substring will be left in the cell. The remainder substring must move to the right until every symbol reaches its final position. Sometimes, these substrings can be displaced at the same time a production rule is applied in the next cell. The fifth case is used in this situation. The seventh case is used when only the displacement is possible. Whenever a symbol reaches its final position, it is stopped (by

means of the eighth case) and kept there (by means of the ninth case) until the next derivation.

3. The configuration associated to the next word is generated:

- After several steps, the cellular automaton generates the configuration associated with the next word derived by the PDOL system. As previously explained, this configuration has the following characteristics:
- It contains a set of contiguous cells embraced by the left and right markers.
- Each cell in this set has an empty displacing substring.

9.3.2.c Example

In this section, the method described in the previous proof is applied to the PDOL system described above:

$$S = \{ \Sigma = \{A, B\}, P = \{A ::= BB, B ::= AB\}, A \}$$

The cellular automaton (C) equivalent to the PDOL system S is defined as

follows:

$$C = (G_C = R_\infty, G_{0_C}, V_{p/s}, Q_C, f_C, T_C = \Phi)$$

where

- The initial configuration is

$$G_{0_C}(i) = \begin{cases} (A, \diamond) & \text{if } i = 0 \\ (>, >) & \text{if } i < 0 \\ (<, <) & \text{if } i \geq 1 \end{cases}$$

- The set of states is

$$Q_C = \{ (>, >), (<, <), (A, \diamond), (B, \diamond), (A, <), (B, <), (<, >\alpha), (A, >\alpha), (B, >\alpha) \}, \alpha \in \Sigma_k, \text{ where } k \leq |h^n(A)|$$

- The transition function is defined as follows:

$$\forall A, U \in \Sigma = \{A, B\} \wedge$$

$$\forall L, R, S \in \Sigma_e = \Sigma \cup \{>, <, \emptyset\} \wedge$$

$$\alpha, \beta \in \Sigma_k \wedge \delta \in \Sigma^+, \gamma \in \Sigma_k$$

$$1. f_c((L, \alpha), (>, >), (R, \beta)) = (>, >)$$

$$2. f_c((L, >U), (<, <), (R, \beta)) = (<, <)$$

$$3. f_c((L, \alpha), (S, \emptyset), (R, <)) = (S, <)$$

$$4. f_c((L, \alpha), (S, <), (R, \beta)) = (S, <)$$

$$5. f_c((L, >U\delta), (A, <), (R, \beta)) = (A, >\delta BB)$$

$$f_c((L, >U\delta), (B, <), (R, \beta)) = (A, >\delta AB)$$

$$6. f_c((L, >), (A, <), (R, \beta)) = (A, >BB)$$

$$f_c((L, >), (B, <), (R, \beta)) = (B, >AB)$$

$$7. f_c((L, >U\delta), (<, <), (R, \beta)) = (<, >\delta)$$

$$8. f_c((L, \alpha), (S, >U\gamma), (R, \beta)) = (U, \emptyset) \text{ where } \gamma \text{ could be the empty string } (\lambda)$$

$$9. f_c((L, \alpha), (\gamma, \emptyset), (R, \beta)) = (\gamma, \emptyset) \text{ where } \beta \neq <$$

Using this transition function, we can follow the algorithm that generates a one-dimensional cellular automata equivalent to the PD0L system. Figure 9-3 above shows how this function is applied.

In the next chapter, this algorithm is extended to D0L systems.

Chapter 10

Cellular Automata equivalent to DOL Systems

10.1. Overview

In this chapter we extend to DOL systems the equivalence algorithm of a one-dimensional cellular automaton equivalent to a given PDOL system. The difference is: that in a DOL system some symbols may be transformed into the empty string, by rules whose right hand part is the empty word. In this case, the cellular automata must provide a technique to solve the problem of propagating the symbols to the left. A cellular automaton is considered equivalent to an L-system if both generate the same words in the same order. Our cellular automata produce the same words and in the same order as the given DOL system for a finite number of derivations. As in the previous chapter, there is no constraint to the DOL system considered, so the method is a general algorithm and can be used as a proof for an equivalence theorem.

10.2. One dimensional Cellular automata equivalent to DOL Systems.

10.2.1 Informal description

Again, the way in which the cellular automaton simulates a DOL system is explained by means of an example:

$S = \{\Sigma = \{A, B, C\}, P = \{A ::= BC, B ::= AC, C ::= \lambda\}, ACCCB\}$

In order to build the equivalent one-dimensional cellular automaton, new features are added to the procedure used for PDOL systems. The total set of restrictions is:

There will be cells in the linear grid of the cellular automaton to contain the symbols of the words derived by the L system (one per cell).

The states of the automaton must provide some mechanism for the following situations:

Inserting new symbols in a given position, and then moving some to the right, because the words can increase their length as they are derived by the L system.

Deleting symbols that are derived to λ (the empty word) and therefore moving other symbols to the left, because the words can decrease their length if the number of symbols deleted is greater than the extra symbols generated by symbols that don't derive λ . We will use different signals to supply these mechanisms.

As a consequence of the previous points, the beginning and the end of the word must be marked. The symbols > and < are, respectively, the left and the right marker.

If a generation only contains a set of contiguous cells embraced by the left and the right markers, and every cell without a marker contains symbol \diamond , then the current generation of the cellular automaton contains a word derived by the L system.

As the length of the derived strings can increase, we must consider that the grid has no right end, so there are always as many right markers as needed.

Figure 10-1 shows the initial generation of the cellular automaton. The symbol contained in each cell appears over the displacing sub-string. This configuration represents the axiom (ACCCB). In every cell, the upper symbol represents the state of the cell, while the lower symbols show the string to be displaced at a given point and the signals used in the process.

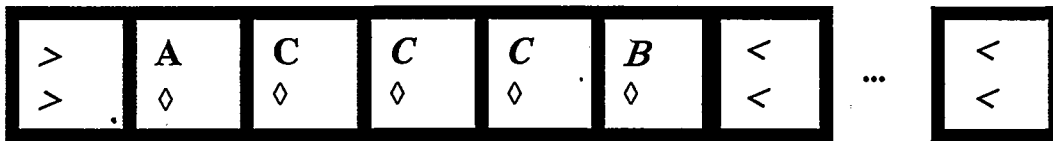


Figure 10-1: Initial generation of the cellular automaton.

The behavior of the cellular automaton is summarized in figure 10-2, which shows twenty-one steps in the evolution of the automaton that simulate the first derivation of the DOL system. Each row in the figure corresponds to a different step for the cellular automaton. Time increases downwards. The first row shows the initial generation of the cellular automaton, which represents the axiom (ACCCB) of the DOL system.

	1	2	3	4	5	6	7
1	$\rangle^{(5)}$ \rangle	$A^{(8)}$ \diamond	$C^{(9)}$ \diamond	$C^{(9)}$ \diamond	$C^{(9)}$ \diamond	$B^{(19)}$ \diamond	$\langle^{(53)}$ \langle
2	$\rangle^{(5)}$ \rangle	$A^{(8)}$ \diamond	$C^{(9)}$ \diamond	$C^{(9)}$ \diamond	$C^{(21)}$ \diamond	$B^{(23)}$ \langle	$\langle^{(53)}$ \langle
3	$\rangle^{(5)}$ \rangle	$A^{(8)}$ \diamond	$C^{(9)}$ \diamond	$C^{(21)}$ \diamond	$C^{(24)}$ \langle	$B^{(26)}$ \langle	$\langle^{(53)}$ \langle
4	$\rangle^{(5)}$ \rangle	$A^{(8)}$ \diamond	$C^{(21)}$ \diamond	$C^{(24)}$ \langle	$C^{(25)}$ \langle	$B^{(26)}$ \langle	$\langle^{(53)}$ \langle
5	$\rangle^{(5)}$ \rangle	$A^{(20)}$ \diamond	$C^{(24)}$ \langle	$C^{(25)}$ \langle	$C^{(25)}$ \langle	$B^{(26)}$ \langle	$\langle^{(53)}$ \langle
6	$\rangle^{(6)}$ \rangle	$A^{(27)}$ \langle	$C^{(25)}$ \langle	$C^{(25)}$ \langle	$C^{(25)}$ \langle	$B^{(26)}$ \langle	$\langle^{(53)}$ \langle
7	$\rangle^{(7)}$ \rangle	^(a) $A^{(34)}$ $\rangle BC$	$C^{(29)}$ \langle	$C^{(25)}$ \langle	$C^{(25)}$ \langle	$B^{(26)}$ \langle	$\langle^{(53)}$ \langle
8	$\rangle^{(5)}$ \rangle	$B^{(10)}$ \diamond	^{α} $C^{(36)}$ $\rangle C$	^{β} $C^{(30)}$ \langle	$C^{(25)}$ \langle	$B^{(26)}$ \langle	$\langle^{(53)}$ \langle
9	$\rangle^{(5)}$ \rangle	$B^{(8)}$ \diamond	$C^{(13)}$ \diamond	^{γ} $C^{(39)}$ \leftarrow	$C^{(32)}$ \langle	$B^{(26)}$ \langle	$\langle^{(53)}$ \langle
10	$\rangle^{(5)}$ \rangle	$B^{(8)}$ \diamond	$C^{(17)}$ \diamond	^{δ} $C^{(44)}$ \leftarrow	$C^{(40)}$ \leftarrow	$B^{(33)}$ \langle	$\langle^{(53)}$ \langle
11	$\rangle^{(5)}$ \rangle	$B^{(8)}$ \diamond	$C^{(17)}$ \diamond	$C^{(46)}$ \leftarrow	$B^{(43)}$ \leftarrow	$B^{(41)}$ \leftarrow	$\langle^{(53)}$ \langle



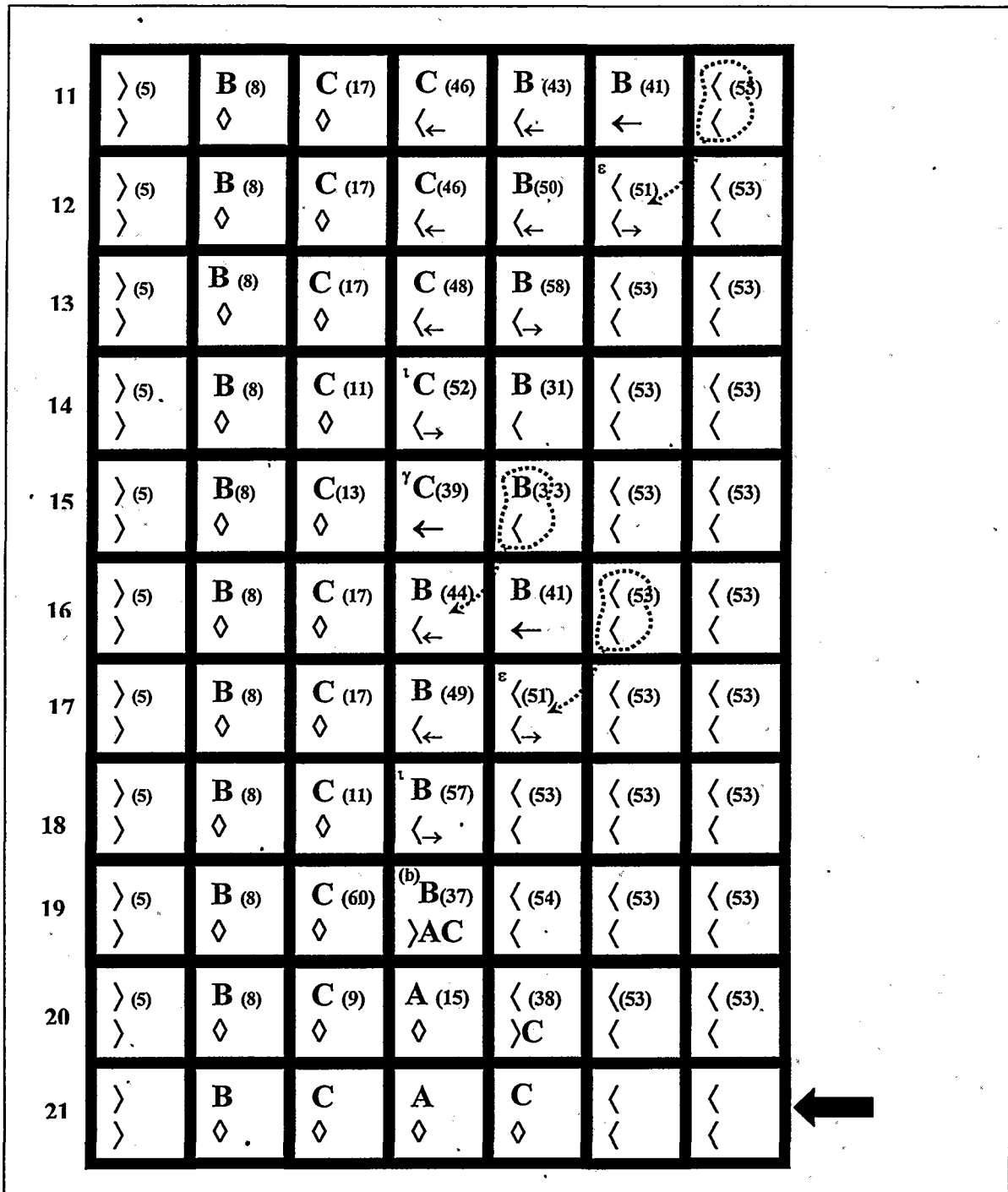


Figure 10-2: Cellular Automaton – DOL system comparison. (α) An erasing rule is applied. The sub-string not yet processed is not displaced to the left, because the sub-string being displaced to the right compensates that displacement. (β) Symbol C must disappear, because the sub-string being displaced to the right is empty and the applied rule ($C ::= \lambda$) does not append new symbols. (γ) At this moment, a signal (\leftarrow) is sent until it reaches the right markers. When the signal goes across a cell, its symbol is displaced to the left. The displacement to the left of the whole sub-string ends when the signal comes back after rebounding against the right end of the word. (δ) Cells that wait for the return of signal \leftarrow are marked with the symbol \leftarrow . (ε) The signal \leftarrow rebounds against the right end of the word, the cells that receive the returning signal are marked with the symbol \leftarrow . (ι) When the returning signal encounters a cell with an empty sub-string, the displacement to the left finishes and a new rule can be applied.

This behavior can be summarized as follows:

- The right marker is transmitted to the left until it reaches the left marker.
- The production rules of the DOL system are applied in the opposite direction. At the same time, the extra symbols and the sub-string not yet processed are displaced until they find their final position.
- As previously mentioned, the displacing sub-string may become empty at a given cell. In this case, a signal is sent to the right and every symbol traversed is displaced one position to the left. When the signal reaches the right marker, it rebounds until arriving at the original cell, and then derivations continue.
- After a word has been derived (rows 1 and 21 in figure 10-2) the automaton is ready for a new derivation.

Notice the following situations:

- The right marker propagates to the left when its left neighbor contains a symbol distinct from the markers (\triangleright , \triangleleft) and an empty displacing sub-string.
- After the application of a production rule, the new symbols must be moved to their final position. This is indicated by the fact that the displacing sub-string begins with the left marker (\triangleright).
- The production rules of the DOL system are applied in cells that contain a non-marker symbol and a displacing sub-string beginning with the left marker (\triangleright). (a) and (b) label the cells in which production rules $A::=BC$ and $B::=AC$ are applied.
- When a sub-string is displaced, the first symbol after the left marker becomes the state of the cell containing the sub-string, because it has found its final position. The left marker and the remaining symbols are propagated.
- If the rule applied has the empty word (λ) as its right hand side, it is possible that the sub-string that has not yet been processed (the cells between the current position and the right marker) must be displaced to the left. In this case the signal is transmitted to the right until it rebounds against the right marker. The cell sends the signal when the lower component of the state of its left neighbor belongs to the set $\{\triangleright s, s \in \Sigma\}$.
- Cells that have just transmitted the signal \leftarrow are marked with the displacing sub-string \triangleleft . They copy their next symbols from their right neighbor. A discontinuous arrow remarks this circumstance.
- When the signal \leftarrow reaches the right marker ($\triangleleft, \triangleleft$) it moves to the left until it finds a cell with an empty displacing sub-string (\diamond). Meanwhile, the symbol $\triangleleft, \triangleright$ is used to unmark the cells that were waiting for the return of the signal.
- An empty displacing sub-string (\diamond) indicates a cell that contains a symbol that is already in its final position.

When symbols are displaced, they propagate the left marker (\triangleright). In a displacing sub-string, the first symbol after the left marker will be finally placed in the cell containing the sub-string. So the next state of this cell represents a symbol that has found its final position. The next state of each cell depends only on its left and right neighbors.

This method is generalized and formalized in the following sections.

10.2.2. Formal description

10.2.2.1. Theorem

Given A D0L system $S=(\Sigma, P, \omega)$ where all the components have been previously defined, and being $L(S, n)$ the set of the first n words generated by S , starting at and including ω , there exists a one dimensional cellular automaton whose states contain $L(S, n)$ in the same order. This automaton is n -equivalent to S .

10.2.2.2. Proof

Our proof is constructive. In the following paragraphs, a one-dimensional cellular automaton (A) that generates the same language that a given D0L system (S) is built.

To make the proof clearer, the following notation will be used:

- $\Sigma_{m_l} = \{\triangleright\}$

is the alphabet of the left marker.

- $\Sigma_{m_r} = \{\triangleleft\}$

is the alphabet of the right marker.

- $\Sigma_m = \{\triangleleft, \triangleright\}$

is the alphabet of both markers.

- $\Sigma_{m_s} = \{\triangleleft\triangleleft, \triangleleft\triangleright, \triangleright\triangleleft\}$

are the signal symbols used for shortening the string due to the λ rules.

- $\Sigma_{em} = \Sigma \cup \Sigma_m \cup \Sigma_{m_s}$

is the alphabet Σ extended with the markers and the signals.

- $\Sigma_e = \Sigma \cup \Sigma_m \cup \Sigma_{m_s} \cup \{\emptyset\}$

- $\Sigma_{e,k} = \bigcup_{i=1}^k \Sigma_e^i$

- $\Sigma_k = \bigcup_{i=1}^k \Sigma^i$

The cellular automaton we want to build is one-dimensional. As indicated in the informal description, its grid (G) is, at least on the right, an infinite vector of deterministic finite automata.

The predecessor / successor neighborhood ($V_{p/s} = (3, (-1, 0, 1))$) is used.

As suggested by the informal description, each automaton must be able to contain the following information:

- A symbol from the alphabet of the D0L system plus the end markers.
- A displacing string of the same kind of symbols plus the signals needed to adjust the length of the next derived word.
- Thus, each possible state of the cellular automaton will be a pair formally defined as follows

$$Q \subseteq \Sigma_{e_m} \times (\Sigma_{m_l} \cdot (\Sigma_k \cup \{\lambda\})) \cup \Sigma_{m_s} \cup \Sigma_m \cup \{\diamond\}$$

where

- $k \leq \max_{A \in \Sigma} \{|\rho(A)|\} \times |\text{hn}(\omega)|$, where max represents the maximum.
- $(>, >)$ and $(<, <)$ are used respectively as left and right markers that fill the portion of the grid that remains unused.
- $(A, <)$, $A \in \Sigma$ are used to propagate the right marker.
- (A, \diamond) , $A \in \Sigma$ are used when the symbol A reaches its final position.
- $(B, >\alpha)$, $B \in \Sigma \cup \{<\}$, $\alpha \in \Sigma^k$ are used to move the displacing sub-strings.
- (C, D) , $C \in \Sigma \cup \{>\}$, $\alpha \in \Sigma^k \wedge D \in \Sigma_m$ are used to shorten the derived string.

The cellular automaton shows initially the axiom of the D0L system. 0 will be considered always the index value corresponding to the first symbol of the words. So G_0 is defined as follows:

$$G_0(i) = \begin{cases} (\omega_j, \lambda) & \text{if } i=j \\ (>, >) & \text{if } i < 0 \\ (<, <) & \text{if } i \geq |\omega| \end{cases}$$

The set of final states is empty: $T = \emptyset$.



The transition function for all the possible cases is formally defined by table 10-1:

	$c[i-1]0$	$c[i]0$	$c[i+1]0$	$f(c[i-1], c[i], c[i+1])$
1	(>, >)	(<, <)	(<, <)	(<, <)
2	(>, >)	(>, >)	(<, <)	(>, >)
3	(>, >)	(>, >)	(<, <→)	(>, >)
4	(>, >)	(>, >)	(s, ←)	(>, >) $\forall s \in \Sigma$
5	(>, >)	(>, >)	(s, ∅)	(>, >) $\forall s \in \Sigma$
6	(>, >)	(>, >)	(s, <)	(>, >) $\forall s \in \Sigma$
7	(>, >)	(>, >)	(s, > α)	(>, >) $\forall s \in \Sigma, \forall \alpha \in \Sigma^* \alpha \leq k$
8	(>, >)	(s, ∅)	(s', ∅)	(s, ∅) $\forall s, s' \in \Sigma$
9	(s', ∅)	(s, ∅)	(s'', ∅)	(s, ∅) $\forall s, s', s'' \in \Sigma$
10	(>, >)	(s, ∅)	(s', > α)	(s, ∅) $\forall s, s' \in \Sigma, \forall \alpha \in \Sigma^+ \alpha \leq k$
11	(s', ∅)	(s, ∅)	(s'', <→)	(s, ∅) $\forall s, s', s'' \in \Sigma$
12	(>, >)	(s, ∅)	(s', <→)	(s, ∅) $\forall s, s' \in \Sigma$
13	(s', ∅)	(s, ∅)	(s'', ←)	(s, ∅) $\forall s, s', s'' \in \Sigma$
14	(>, >)	(s, ∅)	(s', ←)	(s, ∅) $\forall s, s' \in \Sigma$
15	(s', ∅)	(s, ∅)	(<, > α)	(s, ∅) $\forall s, s' \in \Sigma, \forall \alpha \in \Sigma^+ \alpha \leq k$
16	(>, >)	(s, ∅)	(<, > α)	(s, ∅) $\forall s \in \Sigma, \forall \alpha \in \Sigma^+ \alpha \leq k$
17	(s', ∅)	(s, ∅)	(s'', <←)	(s, ∅) $\forall s, s' \in \Sigma$
18	(>, >)	(s, ∅)	(s', <←)	(s, ∅) $\forall s, s' \in \Sigma$
19	(s', ∅)	(s, ∅)	(<, <)	(s, <) $\forall s, s' \in \Sigma$
20	(>, >)	(s, ∅)	(s', <)	(s, <) $\forall s, s' \in \Sigma$
21	(s', ∅)	(s, ∅)	(s'', <)	(s, <) $\forall s, s' \in \Sigma$
22	(>, >)	(s, ∅)	(<, <)	(s, <) $\forall s \in \Sigma$
23	(s', ∅)	(s, <)	(<, <)	(s, <) $\forall s, s' \in \Sigma$
24	(s', ∅)	(s, <)	(s'', <)	(s, <) $\forall s, s', s'' \in \Sigma$
25	(s', <)	(s, <)	(s'', <)	(s, <) $\forall s, s', s'' \in \Sigma$
26	(s', <)	(s, <)	(<, <)	(s, <) $\forall s, s' \in \Sigma$
27	(>, >)	(s, <)	(s', <)	((s, > $\rho(s)$) $\forall s \in \Sigma \rho(s) \neq \lambda, \forall s' \in \Sigma$
28	(>, >)	(s, <)	(<, <)	(s, > $\rho(s)$) $\forall s \in \Sigma \rho(s) \neq \lambda$
29	(s', > s'' α)	(s, <)	(s''', <)	(s, > $\alpha \rho(s)$) $\forall s \in \Sigma \rho(s) \neq \lambda, \forall s', s'', s''' \in \Sigma, \forall \alpha \in \Sigma^+ \alpha \leq k$
30	(s', > s''')	(s, <)	(s''', <)	(s, ←) $\forall s \in \Sigma \rho(s) = \lambda, \forall s', s'', s''' \in \Sigma$
31	(s', <→)	(s, <)	(<, <)	(s, <) $\forall s, s' \in \Sigma$
32	(s', ←)	(s, <)	(s'', <)	(s, ←) $\forall s, s', s'' \in \Sigma$
33	(s', ←)	(s, <)	(<, <)	(s, ←) $\forall s, s' \in \Sigma$

34	(>, >)	(s, > s'α)	(s'', <)	(s', ∅) ∀s, s', s'' ∈ Σ, ∀α ∈ Σ* α ≤ k
35	(>, >)	(s, > s'α)	(s'', <)	(s', ∅) ∀s, s' ∈ Σ, ∀α ∈ Σ* α ≤ k
36	(s'', ∅)	(s, > s'α)	(s'', <)	(s', ∅) ∀s, s', s'', s''' ∈ Σ, ∀α ∈ Σ* α ≤ k
37	(s'', ∅)	(s, > s'α)	(s'', <)	(s', ∅) ∀s, s', s'' ∈ Σ, ∀α ∈ Σ* α ≤ k
38	(s, ∅)	(s, > s'α)	(s'', <)	(s', ∅) ∀s, s' ∈ Σ, ∀α ∈ Σ* α ≤ k
39	(s', ∅)	(s, ←)	(s'', <)	(s'', <←) ∀s, s', s'' ∈ Σ
40	(s', <←)	(s, ←)	(s'', <)	(s'', <←) ∀s, s', s'' ∈ Σ
41	(s', <←)	(s, ←)	(s'', <)	(s'', <←) ∀s, s' ∈ Σ
42	(>, >)	(s, ←)	(s'', <)	(s'', <←) ∀s ∈ Σ
43	(s', <←)	(s, <←)	(s'', <←)	(s, <←) ∀s, s', s'' ∈ Σ
44	(s', ∅)	(s, <←)	(s'', <←)	(s, <←) ∀s, s', s'' ∈ Σ
45	(s', <←)	(s, <←)	(s'', <←)	(s, <←) ∀s, s', s'' ∈ Σ
46	(s', ∅)	(s, <←)	(s'', <←)	(s, <←) ∀s, s', s'' ∈ Σ
47	(s', <←)	(s, <←)	(s'', <←)	(s, <←) ∀s, s', s'' ∈ Σ
48	(s', ∅)	(s, <←)	(s'', <←)	(s, <←) ∀s, s', s'' ∈ Σ
49	(s', ∅)	(s, <←)	(s'', <←)	(s, <←) ∀s, s' ∈ Σ
50	(s', <←)	(s, <←)	(s'', <←)	(s, <←) ∀s, s' ∈ Σ
51	(s, <←)	(s, <←)	(s'', <←)	(s, <←) ∀s ∈ Σ
52	(s', ∅)	(s, <←)	(s'', <←)	(s, <←) ∀s ∈ Σ ρ(s)=λ, ∀s', s'' ∈ Σ
53	(s', x)	(s, <←)	(s'', <←)	(s, <←) ∀s ∈ Σ, ∀x ≠ α, α ∈ Σ+
54	(s, > s'α)	(s, <←)	(s'', <←)	(s, <←) ∀s, s' ∈ Σ, ∀α ∈ Σ+
55	(>, >)	(s, <←)	(s'', <←)	(s, <←) ∀s ∈ Σ ρ(s)=λ
56	(>, >)	(s, <←)	(s'', <←)	(s, <←)
57	(s', ∅)	(s, <←)	(s'', <←)	(s, <←) ∀s ρ(s) ≠ λ, ∀s' ∈ Σ
58	(s', <←)	(s, <←)	(s'', <←)	(s, <←) ∀s, s' ∈ Σ
59	(s, > s'α)	(s, <←)	(s'', <←)	(s, <←) ∀s, s' ∈ Σ, ∀α ∈ Σ+
60	(s', ∅)	(s, ∅)	(s'', > α)	(s, ∅) ∀s, s', s'' ∈ Σ, ∀α ∈ Σ+

Table 10-1: Transition function of the cellular automaton equivalent to a DOL system.

The rows in the transition function are explained below:

- Cases 1 and 53 maintain the right marker unchanged.
- Cases 2 to 7 maintain the left marker unchanged.
- Cases 8 to 18 maintain symbols at their final positions.
- Cases 19 to 22 the derivation signal is receipt by symbols at their final position
- Cases 23 to 26 maintain unchanged symbols that are waiting for derivation
- Cases 27 to 29 apply a derivation rule
- Cases 30 to 33, 39 to 52 and 55 to 58 handle the deletion signals.
- Cases 34 to 38 place symbols at their final positions.
- Cases 54 and 59 move the displacing substring to the right.

This function defines the following mechanism to eliminate symbols: the signal \leftarrow is placed to the bottom of the cell, and propagated to the right. Then the symbol in the right neighbour is copied to the cell that contains the signal, and signal (\leftarrow) is assigned to the bottom of the cell. This process is repeated until the end of the grid is reached, which is marked by (\leftarrow , \leftarrow). At this stage, the other signal (\leftarrow), which indicates that the propagation is finished, is placed at the bottom of each left cell until the cell containing λ is reached again. At that point, the derivation process can be continued.

10.3. Examples

Example 1

The example shown in section 1.3.1 can be formalized thus:

$$S = \{\Sigma = \{A, B\}, P = \{A ::= BC, B ::= AC, C ::= \lambda\}, ACCCB\}$$

The cellular automaton (C) equivalent to this DOL system S is defined as follows:

$$C = \{G_c, G_{0c}, V_{p/s}, Q_c, f_c, T_c = \emptyset\}$$

where

- The initial configuration is

$$G_0(i) = \begin{cases} (>, >) & \text{if } i < 0 \\ (A, \emptyset), (C, \emptyset), (C, \emptyset), (C, \emptyset), (B, \emptyset) & \text{if } 0 \leq i \leq |\text{axioma}| \\ (<, <) & \text{if } i > |\text{axioma}| \end{cases}$$

- The set of states is $Q_c = \left\{ \begin{array}{l} (>, >), (<, <), (A, \diamond), (C, \diamond), (B, \diamond), (A, <), (B, <), (C, <), \\ (A, > \alpha), (B, > \alpha), (C, > \alpha), (C, \leftarrow), (C, \leftarrow), (B, \leftarrow), \\ (A, \leftarrow), (<, \leftarrow), (C, \leftarrow), (B, \leftarrow), (A, \leftarrow) \end{array} \right\}$
- The transition function is defined in table 10-1, and the first derivation in the S system corresponds to the cellular automaton evolution as shown in figure 10-2,

Not all the cases in the transition function are used in figure 10-2. The following examples show other possible cases the algorithm may have to face, and how they are solved.

Example 2

Let us consider a D0L systems defined thus:

$$\{\Sigma = \{A, B, C\}, P = \{A ::= \lambda, B ::= \lambda, C ::= \lambda\}, ACCCCCCCB\}$$

The cellular automaton (C) equivalent to this D0L system S is defined as follows:

$$C = \{G_c, G_{0c}, V_{p/s}, Q_c, f_c, T_c = \emptyset\}$$

where all the components were defined previously.

Figure 10-3 shows how the cellular automaton simulates its corresponding D0L system.

Example 3

$$\{\Sigma = \{A, B, C\}, P = \{A ::= BCA, B ::= B, C ::= B\}, A\}$$

The cellular automaton (C) equivalent to this D0L system S is defined as follows:

$$C = \{G_c, G_{0c}, V_{p/s}, Q_c, f_c, T_c = \emptyset\}$$

where all the components were defined previously.

Figure 10-4 shows how the cellular automaton simulates the first derivation of its corresponding D0L-system.

Example 4

$$\{\Sigma=\{C\}, P=\{C::=\lambda\}, C\}$$

The cellular automaton (C) equivalent to this DOL system S is defined as follows:

$$C=\{G_c, G_{0c}, V_{p/s}, Q_c, f_c, T_c=\emptyset\}$$

where all the components were defined previously.

Figure 10-5 shows how the cellular automaton simulates the derivation of λ by its corresponding DOL-system.

	1	2	3	4	5	6	7	8
1	> >	A ◇	C ◇	C ◇	C ◇	C ◇	B ◇	< <
2	> >	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮
3	> >	A <	C <	C <	C <	C <	B <	< <
4	> >	A ←	C <	C <	C <	C <	B <	< <
5	> >	C ←	C ←	C <	C <	C <	B <	< <
6	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮	⋮ ⋮
7	> >	C ←	C ←	C ←	C ←	B ←	< ←	< <
8	> >	C ←	C ←	C ←	C ←	B ←	< ←	< <

Figure 10-3

	1	2	3	4
1	> >	A ◇	< <	< <
2	> >	A <	< <	< <
3	> >	A)BCA	< <	< <
4	> >	B ◇	< >CA	< <
5	> >	B ◇	C ◇	< >A <
6	> >	B ◇	C ◇	A ◇ <

Figure 10-4

	1	2	3	4
1	> >	C ◇	< <	< <
2	> >	C <	< <	< <
3	> >	C ←	< <	< <
4	> >	< <→	< <	< <
5	> >	< <	< <	< <

Figure 10-5

Chapter 11

Cellular automata equivalent to DIL systems

11.1. Overview

In the two previous chapters 9 and 10, we have presented an equivalence theorem between cellular automata and PDOL and DOL systems. In this chapter, we are tackling the equivalence between cellular automata and another type of L-system, called DIL systems. DIL systems are deterministic context sensitive systems, where the set of production rules determines the only way in which each symbol in the alphabet can be changed into a word, whenever it appears between certain substrings located at its right and at its left. Therefore, the neighborhood of the equivalent cellular automata must be modified, to take into account the context of the DIL system in that part of the process when the production rules should be applied. In the next sections we will propose an algorithm to solve this problem, and a number of examples will be introduced to better understand the algorithm.

11.2. One dimensional Cellular Automata equivalent to DIL Systems

11.2.1. Informal description

In the previous chapters, we have shown that, for every DOL system S_d , there exists a one-dimensional cellular automaton A_d that generates its first n derivations. Each cell in the A_d cellular automaton's grid is associated to a symbol of the words derived by system S_d . Symbols $<$ and $>$ mark the word's ends, and several signals are used to insert new symbols (if needed) in a given position after applying a derivation rule $(\leftarrow, \rightarrow)$, and to reduce the size of the words (if needed) when λ rules are used $(\leftarrow\leftarrow, \rightarrow\rightarrow, \leftarrow\leftarrow, \rightarrow\rightarrow)$. Each state of A_d has two components: one for the current cell's symbol and another for signals and substrings that should be displaced during the derivation process.

A new component has been added to the cellular automaton states to handle the context: initially, cells containing symbols have an empty displacing substring (\diamond) and a context information equal to themselves, while the filler (g) is used as context for cells containing the markers, as illustrated in figure 11-1.

g	A	A	A	g
>	A	A	A	<
>	\diamond	\diamond	\diamond	<

Figure 11-1: The initial configuration of the cellular automaton

The automaton simulates sequentially (from left to right) the parallel derivation process, and the context must be maintained during the whole derivation to correctly apply context sensitive rules. Thus, some mechanism must be supplied to keep the context information correct while the symbols are displaced to the left when λ rules are applied, and to set the appropriate context information after having finished the whole derivation. The following new symbol (\diamond') and signal ($\diamond' \leftarrow$) will be used.

The behavior of the cellular automaton equivalent to a DIL system is summarized by means of an example.

The (1,1)DIL system is defined thus:

$$S = \{\Sigma = \{A, B, \},$$

$$P = \{AAA ::= AA, AAB ::= BA, BAA ::= BA, BAB ::= AB, gAA ::= B, gAB ::= A, AAg ::= B,$$

$$BAg ::= A, xBy ::= \lambda \forall x, y \in \Sigma\}, AAA\}$$

Figures 11-2 and 11-3, illustrate several steps in the evolution of the automaton that simulates the first derivation of the DIL system. Each row in the figure corresponds to a different step for the cellular automaton. Time increases downwards. As it was mentioned before, every marker cell has the filler (g) as context information; cells associated to symbols have an empty displacing substring (\diamond) and the symbol itself as context information. The behavior can be summarized as follows:

- Signal \leftarrow is propagated from right to left until it reaches the first symbol in the axiom. At that point, the symbol is replaced by the left-hand side of the appropriate production rule, depending on its context (gAA ::= B). See figure 11-2. *The rule context of the applied rule can be obtained from the top of the cell and its neighbors.*

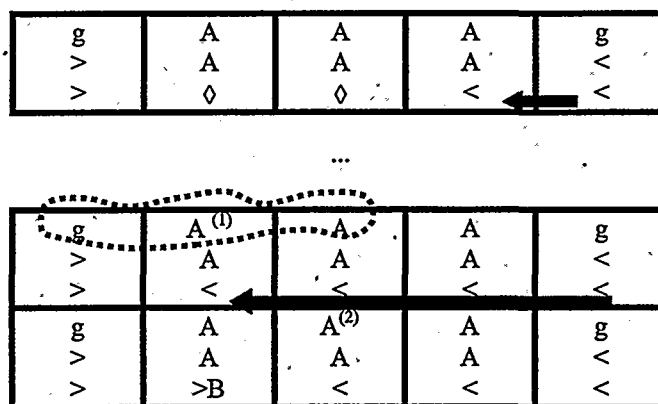


Figure 11-2: At the first step in the derivation, signal \leftarrow is transmitted to the left.

- The first symbol in the displacing substring is allocated in its final position. A new symbol (\diamond') is introduced to indicate that the leftmost symbol has already been derived in this step (maintaining the old contents of the cell, A,

at the top of the state). At the same time the first cell state becomes (A,B, \diamond'), the next cell applies its own rule ($AAA::=AA$), whose right hand-side is added to the remainder of the displacing substring (λ in this case). The process is repeated until the right marker is reached, after the rule $AAg::=B$ has been applied.

- Signal $\diamond' \leftarrow$ is sent to the left to make each cell write its correct context and replace symbol \diamond' by \diamond , to finish the derivation. Figure 11-3 shows these steps.

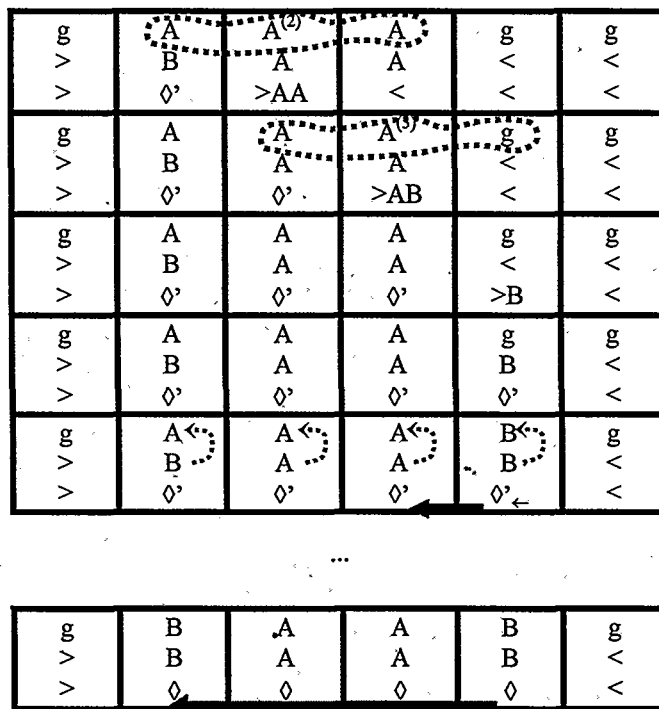


Figure 11-3: Ending the first derivation in the cellular automaton equivalent to the DIL system.

- This configuration is associated with $h(\omega)$, that is, the first word derived from S 's axiom. A new derivation begins now, in the same way described above. Rule $xBy \rightarrow \lambda$ means that the symbol B must be erased from the string wherever it is. When this rule is applied, all the symbols to its right must be displaced one position to the left, because the displacing substring of B 's left neighbor has no symbol. Signal \leftarrow is sent to the right end while displacing each symbol. Each cell traversed by the signal is marked with the symbol \leftarrow . Context information must be handled carefully, because it must be kept unchanged until the whole derivation finishes. If context information is simply displaced to the left on the cell that contains the deleted B , the context will become "gAABg" rather than "gBAABg," which is the correct one. To avoid this mistake, the context of B 's left neighbor is "added" to B 's context as shown in the example.

- When signal \leftarrow reaches the right end it rebounds (turned into \leftarrow) until reaching the cell that originally contained the deleted symbol, while it changes the \leftarrow symbols to the marker $<$ (see figure 11-4).

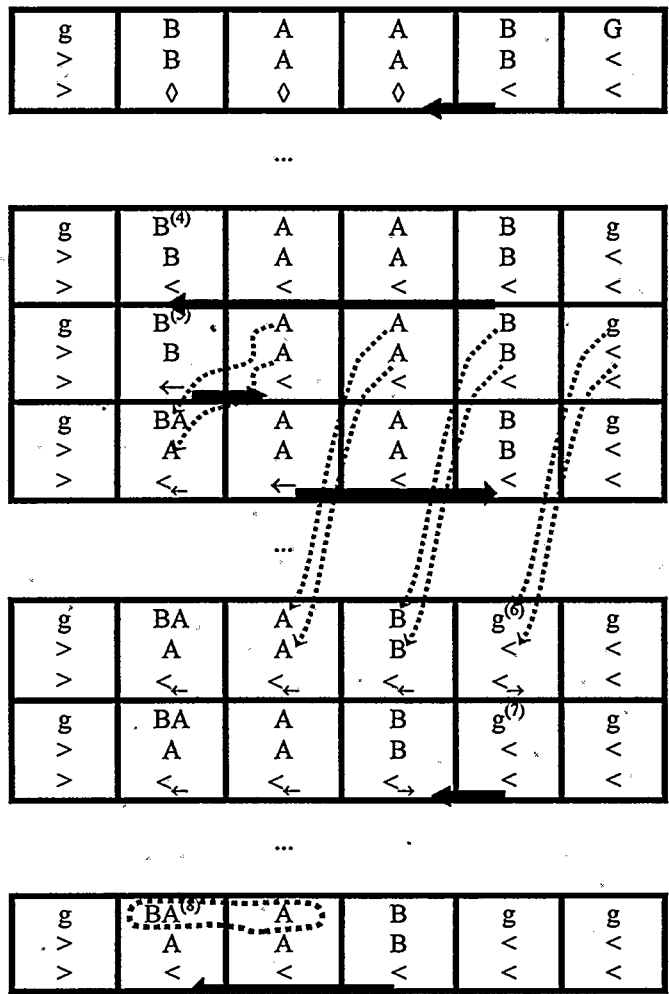


Figure 11-4: The cellular automaton mechanisms to generate the second word derived by the DIL-system.

Notice that the context (gBAABg) has been correctly kept during the deletion of symbol B, hence the rule $BAA \rightarrow BA$ can now be correctly applied. Notice also that, when a cell has a string as its context information, the symbol itself (the current content associated to the cell) is located at the last (rightmost) position of the context. The derivation process continues ($AAB \rightarrow BA$) as explained before, until the rule $xBy \rightarrow \lambda$ is applied again at the right end of the preceding derived word. In this case no symbol must be displaced to the left, because there are symbols enough to compensate B's deletion (see figure 11-5).

g	BA	A	B	g	g
>	A	A	B	<	<
>	>BA	<	<	<	<
g	BA	A ⁽⁹⁾	B	g	g
>	B	A	B	<	<
>	∅'	>ABA	<	<	<
g	BA	A	B ⁽¹⁰⁾	g	g
>	B	A	B	<	<
>	∅'	∅'	>BA	<	<
g	BA	A	B	g	g
>	B	A	B	<	<
>	∅'	∅'	∅'	>A	<
g	BA	A	B	g	g
>	B	A	B	A	<
>	∅'	∅'	∅'	∅'	<
g	BA	A	B	A	g
>	B	A	B	A	<
>	∅'	∅'	∅'	∅'←	<

...

g	B	A	B	A	g
>	B	A	B	A	<
>	∅	∅	∅	∅	<

Figure 11-5: The cellular automaton joins the contexts of the deleted dells and keeps it unchanged until the end of the derivation, when a especial signal ($\emptyset' \leftarrow$) is sent to the left, putting the correct context in its place, identical to the new derived word.

In the first row in figure. 11-5, we note that the cellular automaton continues generating the next word derived by the DIL-system. The intermediate stage is obtained when all the cells have the symbol (\emptyset') at the lowest level of the rows in the figure, surrounded by ($g, >, >$) and ($g, <, <$). At this point, signal $\emptyset' \leftarrow$ is sent to the left, to update the context information, and the word $h^2(\omega)$ is obtained. After that, the cellular automaton is ready for a new derivation, and so on.

11.2.2. Formal description

11.2.2.1. Theorem

Given a DIL system $S=(\Sigma, P, \omega)$, where all the components have been previously defined, and being $L(S, n)$ the set of the first n words generated by S , starting at and including ω , there exists a one dimensional cellular automaton whose states contain $L(S, n)$ in the same order. This automaton is n -equivalent to S .

11.2.2.2. Proof

As previously, we build a constructive proof. In the next paragraphs, a one dimensional automaton (A) that simulates the same language generated by the DIL system is built. Notice the following notation:

- $\Sigma_{m_l} = \{>\}$

is the alphabet of the left marker.

- $\Sigma_{m_r} = \{<\}$

is the alphabet of the right marker.

- $\Sigma_m = \{<, >\}$

is the alphabet of both markers.

- $\Sigma_{m_s} = \{<\leftarrow, \leftarrow, \rightarrow, \leftarrow\}$

are the signal symbols used for shortening the string due to the λ rules.

- $\Sigma_{em} = \Sigma \cup \Sigma_m \cup \Sigma_{m_s}$

is the alphabet Σ extended with the markers and the signals.

- $\Sigma_{mc} = \{\diamond \leftarrow\}$

is the signal that is used to set the right context after a derivation is completed .

- $\Sigma_e = \Sigma \cup \Sigma_m \cup \Sigma_{m_s} \cup \{\diamond, \diamond'\}$

- $\Sigma_{e,k} = \bigcup_{i=1}^k \Sigma_e^i$

- $\Sigma_k = \bigcup_{i=1}^k \Sigma^i$

The cellular automaton we want to build is one-dimensional. As indicated in the informal description, its grid (G) is, at least at its right, an infinite vector of finite and deterministic automata.

The predecessor / successor neighborhood

($V_{p/s} = (L+R+1, (-1, -(L-1), \dots, 0, 1, 2, \dots, R-1, R))$) is used.

As indicated in the informal description, each automaton must be able to contain the following information:

- A symbol from the alphabet of the DIL system plus the end markers.
- A displacing string of the same kind of symbols plus the signals needed to adjust the length of the next derived word.
- A symbol or string containing the context, which must be kept the same during the whole derivation process.

Thus, each possible state of the cellular automaton will be a three-fold formally defined as follows

$$Q \subseteq \Sigma_{e_m} \times (\Sigma_{m_l} \cdot (\Sigma_k \cup \{\lambda\})) \cup \Sigma_{m_r} \cup \Sigma_{m_c} \cup \Sigma_m \cup \{\emptyset\} \cup \{\emptyset\}$$

where

- $k \leq \max_{A \in \Sigma} \{|\rho(A)|\} \times |h^n(\omega)|$, where *max* represents the maximum.
- $(g, >, >)$ and $(g, <, <)$ are used respectively as left and right markers that fill the portion of the grid that remains unused.
- $(A, B, <)$, $A \in \Sigma \cup \{g\}$ and $B \in \Sigma$ are used to propagate the right marker.
- (A, B, \emptyset') , $A \in \Sigma$ are used when symbol A reaches its intermediate position.
- (A, B, \emptyset) , $A \in \Sigma$ are used when symbol A reaches its final position.
- $(A, B, > \alpha)$, $B \in \Sigma \cup \{<\} \wedge \alpha \in \Sigma_k$, $A \in \cup_{i=1}^R \Sigma^i$ are used to move the displacing substrings.
- (A, C, D) , $C \in \Sigma \cup \{>\} \wedge \alpha \in \Sigma_k \wedge D \in \Sigma_{ms}$ are used to shorten the derived string.

The cellular automaton shows initially the axiom of the DIL system. 0 will be considered always the index value corresponding to the first symbol in a word. So G_0 is defined as follows:

$$G_{0s,n}[I] = \begin{cases} (g, >, >) & \text{if } i < 0 \\ (s, s, \emptyset) & \forall i \in [1, |\omega|], s = \omega[i] \\ (g, <, <) & \text{Otherwise} \end{cases}$$

The set of final states is empty: $T = \emptyset$

The transition function for all the possible cases is formally defined in table 11-1

	$c[i-1,0]$	$c[i,0]$	$c[i+1,0]$	$f_s(C[i-1], C[i], C[i+1])$
1	$(g, >, >)$	$(g, <, <)$	$(g, <, <)$	$(g, <, <)$
2	$(g, >, >)$	$(g, >, >)$	$(g, <, <)$	$(g, >, >)$
3	$(g, >, >)$	$(g, >, >)$	$(g, <, <_)$	$(g, >, >)$
4	$(g, >, >)$	$(g, >, >)$	(A, B, \leftarrow)	$(g, >, >) \quad \forall A, B \in \Sigma$
5	$(g, >, >)$	$(g, >, >)$	(A, B, \emptyset)	$(g, >, >) \quad \forall A, B \in \Sigma$
6	$(g, >, >)$	$(g, >, >)$	$(A, B, <)$	$(g, >, >) \quad \forall A, B \in \Sigma$
7	$(g, >, >)$	$(g, >, >)$	$(A, B, > \alpha)$	$(g, >, >) \quad \forall A \in \cup_{i=1}^R \Sigma^i, \forall B \in \Sigma, \forall \alpha \in \Sigma^* \mid \alpha \leq k$
8	$(g, >, >)$	(A, B, \emptyset)	(C, D, \emptyset)	$(A, B, \emptyset) \quad \forall A, B, C, D \in \Sigma$
9	(A, B, \emptyset')	(C, D, \emptyset')	(E, F, \emptyset')	$(C, D, \emptyset') \quad \forall A, C, E \in \cup_{i=1}^R \Sigma^i \quad \forall B, D, F \in \Sigma$
10	$(g, >, >)$	(A, B, \emptyset')	$(C, D, > \alpha)$	$(A, B, \emptyset) \quad \forall B, D \in \Sigma, \forall \alpha \in \Sigma \mid \alpha \leq k$ $\forall A, C \in \cup_{i=1}^R \Sigma^i$
11	(A, B, \emptyset')	(C, D, \emptyset')	$(E, F, <_)$	$(C, D, \emptyset') \quad \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
12	$(g, >, >)$	(A, B, \emptyset')	$(C, D, <_)$	$(A, B, \emptyset') \quad \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
13	(A, B, \emptyset')	(C, D, \emptyset')	(E, F, \leftarrow)	$(C, D, \emptyset') \quad \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
14	$(g, >, >)$	(A, B, \emptyset')	(C, D, \leftarrow)	$(A, B, \emptyset') \quad \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
15	(A, B, \emptyset')	(C, D, \emptyset')	$(g, <, > \alpha)$	$(C, D, \emptyset') \quad \forall B, D \in \Sigma, \forall \alpha \in \Sigma \mid \alpha \leq k,$ $\forall A, C \in \cup_{i=1}^R \Sigma^i$
16	$(g, >, >)$	(A, B, \emptyset')	$(g, <, > \alpha)$	$(A, B, \emptyset') \quad \forall B \in \Sigma, \forall \alpha \in \Sigma \mid \alpha \leq k, \forall A \in \cup_{i=1}^R \Sigma^i$
17	(A, B, \emptyset')	(C, D, \emptyset')	$(E, F, <_)$	$(C, D, \emptyset') \quad \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
18	$(g, >, >)$	(A, B, \emptyset')	$(C, D, <_)$	$(A, B, \emptyset'), \forall A, B, C, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
19	(A, B, \emptyset)	(C, D, \emptyset)	$(g, <, <)$	$(C, D, <) \quad \forall A, B, C, D \in \Sigma$
20	$(g, >, >)$	(A, B, \emptyset)	$(C, D, <)$	$(A, B, <) \quad \forall A, B, C, D \in \Sigma$
21	(A, B, \emptyset)	(C, D, \emptyset)	$(E, F, <)$	$(C, D, <) \quad \forall A, B, C, D, E, F \in \Sigma$
22	$(g, >, >)$	(A, B, \emptyset)	$(g, <, <)$	$(A, B, <) \quad \forall A, B \in \Sigma$
23	(A, B, \emptyset)	$(C, D, <)$	$(g, <, <)$	$(C, D, <) \quad \forall A, B, C, D \in \Sigma$
24	(A, B, \emptyset)	$(C, D, <)$	$(E, F, <)$	$(C, D, <) \quad \forall A, B, C, D, E, F \in \Sigma$
25	$(A, B, <)$	$(C, D, <)$	$(E, F, <)$	$(C, D, <) \quad \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
26	$(A, B, <)$	$(C, D, <)$	$(g, <, <)$	$(C, D, <) \quad \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
27	$(g, >, >)$	$(A, B, <)$	$(C, D, <)$	$((A, B, > \rho(B)) \quad \forall A, B \in \Sigma \mid \rho(B) \neq \lambda, \forall C, D \in \Sigma$
28	$(g, >, >)$	$(A, B, <)$	$(g, <, <)$	$(A, B, > \rho(B)) \quad \forall B \in \Sigma \mid \rho(s) \neq \lambda, \forall A \in \cup_{i=1}^R \Sigma^i$
29	$(A, B, > C\alpha)$	$(D, E, <)$	$(F, G, <)$	$(D, E, > \alpha \rho(E)) \quad \forall E \in \Sigma \mid \rho(E) \neq \lambda,$ $\forall A \in \cup_{i=1}^R \Sigma^i, \forall B, C, D, E, F, G \in \Sigma, \forall \alpha \in \Sigma \mid \alpha \leq k$

30	$(A, B, > C)$	$(D, E, <)$	$(F, G, <)$	$(D, E, \leftarrow) \forall E \in \Sigma \rho(s) = \lambda, \forall B, C, E, G \in \Sigma$ $\forall A, D, F \in \cup_{i=1}^R \Sigma^i$
31	$(A, B, < \rightarrow)$	$(C, D, <)$	$(g, <, <)$	$(C, D, <) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
32	(A, B, \leftarrow)	$(C, D, <)$	$(E, F, <)$	$(C, D, \leftarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
33	(A, B, \leftarrow)	$(C, D, <)$	$(g, <, <)$	$(C, D, \leftarrow) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
34	$(g, >, >)$	$(A, B, > C\alpha)$	$(D, E, <)$	$(A, C, \diamond') \forall B, C, D, E \in \Sigma, \forall \alpha \in \Sigma^* \alpha \leq k,$ $\forall A \in \cup_{i=1}^R \Sigma^i$
35	$(g, >, >)$	$(A, B, > C\alpha)$	$(g, <, <)$	$(A, C, \diamond') \forall B, C \in \Sigma, \forall \alpha \in \Sigma^* \alpha \leq k, \forall A \in \cup_{i=1}^R \Sigma^i$
36	(A, B, \diamond')	$(C, D, > E\alpha)$	$(F, G, <)$	$(C, E, \diamond') \forall B, D, E, G \in \Sigma, \forall \alpha \in \Sigma^* \alpha \leq k,$ $\forall A, C, F \in \cup_{i=1}^R \Sigma^i$
37	(A, B, \diamond')	$(C, D, > E\alpha)$	$(g, <, <)$	$(C, E, \diamond') \forall A, B, C, D \in \Sigma, \forall \alpha \in \Sigma^* \alpha \leq k,$ $\forall A, C \in \cup_{i=1}^R \Sigma^i$
38	(A, B, \diamond')	$(g, <, > C\alpha)$	$(g, <, <)$	$(g, C, \diamond') \forall B, C \in \Sigma, \forall \alpha \in \Sigma^* \alpha \leq k, \forall A \in \cup_{i=1}^R \Sigma^i$
39	(A, B, \diamond')	(C, D, \leftarrow)	$(E, F, <)$	$(CE, F, < \leftarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
40	$(A, B, < \leftarrow)$	(C, D, \leftarrow)	$(E, F, <)$	$(E, F, < \leftarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
41	$(A, B, < \leftarrow)$	(C, D, \leftarrow)	$(g, <, <)$	$(Cg, <, < \leftarrow) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
42	(A, B, \diamond')	(C, D, \leftarrow)	$(g, <, <)$	$(Cg, <, < \leftarrow) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
43	$(g, >, >)$	(A, B, \leftarrow)	$(g, <, <)$	$(Ag, <, < \leftarrow) \forall B \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$
44	$(A, B, < \leftarrow)$	$(C, D, < \leftarrow)$	(E, F, \leftarrow)	$(C, D, < \leftarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
45	(A, B, \diamond')	$(C, D, < \leftarrow)$	(E, F, \leftarrow)	$(E, F, \leftarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
46	$(A, B, < \leftarrow)$	$(C, D, < \leftarrow)$	$(E, F, < \rightarrow)$	$(C, D, < \rightarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
47	(A, B, \diamond')	$(C, D, < \leftarrow)$	$(E, F, < \leftarrow)$	$(C, D, < \leftarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
48	$(A, B, < \leftarrow)$	$(C, D, < \leftarrow)$	$(E, F, < \leftarrow)$	$(C, D, < \leftarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
49	(A, B, \diamond')	$(C, D, < \leftarrow)$	$(E, F, < \rightarrow)$	$(C, D, < \rightarrow) \forall B, D, F \in \Sigma, \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
50	(A, B, \diamond')	$(C, D, < \leftarrow)$	$(g, <, < \rightarrow)$	$(C, D, < \rightarrow) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
51	$(A, B, < \leftarrow)$	$(C, D, < \leftarrow)$	$(g, <, < \rightarrow)$	$(C, D, < \rightarrow) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
52	$(A, B, < \leftarrow)$	$(g, <, < \rightarrow)$	$(g, <, <)$	$(g, <, <) \forall B \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$
53	(A, B, \diamond')	$(C, D, < \rightarrow)$	$(E, F, <)$	$(C, D, \leftarrow) \forall D \in \Sigma \rho(D) = \lambda, \forall B, D, E, F \in \Sigma,$ $\forall A, C, E \in \cup_{i=1}^R \Sigma^i$
54	(A, s', x)	$(g, <, <)$	$(g, <, <)$	$(g, <, <) \forall A \in \cup_{i=1}^R \Sigma^i, \forall x \neq \alpha, \alpha \in \Sigma^+,$
55	$(A, B, > C\alpha)$	$(g, <, <)$	$(g, <, <)$	$(g, <, > \alpha) \forall B, C \in \Sigma, \forall \alpha \in \Sigma^+, \forall A \in \cup_{i=1}^R \Sigma^i$
56	$(g, >, >)$	$(A, B, <)$	$(g, <, <)$	$(A, B, \leftarrow) \forall B \in \Sigma \rho(B) = \lambda, \forall A \in \cup_{i=1}^R \Sigma^i$

57	$(g, >, >)$	$(g, <, <_{\rightarrow})$	$(g, <, <)$	$(g, <, <)$
58	(A, B, \diamond')	$(C, D, <_{\rightarrow})$	$(g, <, <)$	$(C, D, > \rho(D)) \forall D \mid \rho(D) \neq \lambda, \forall B, D \in \Sigma,$ $\forall A, C \in \cup_{i=1}^R \Sigma^i$
59	$(A, B, <_{\leftarrow})$	$(C, D, <_{\rightarrow})$	$(g, <, <)$	$(C, D, <) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
60	$(A, <, > B\alpha)$	$(g, <, <)$	$(g, <, <)$	$(g, <, > \alpha) \forall B \in \Sigma, \forall \alpha \in \Sigma^+, \forall A \in \cup_{i=1}^R \Sigma^i$
61	(A, B, \diamond)	(C, D, \diamond)	(E, F, \diamond)	$(C, D, \diamond) \forall A, B, C, D, E, F \in \Sigma$
62	$(g, >, >)$	$(A, B, \diamond'_{\leftarrow})$	(C, D, \diamond)	$(B, B, \diamond) \forall B, C, D \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$
63	(A, B, \diamond')	(C, D, \diamond')	$(Cg, <, <_{\rightarrow})$	$(C, D, \diamond') \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
64	(A, B, \diamond')	$(Cg, <, <_{\rightarrow})$	$(g, <, <)$	$(g, <, <) \forall B \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
65	(A, B, \diamond')	(C, D, \diamond')	$(g, <, <)$	$(C, D, \diamond'_{\leftarrow}) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
66	$(A, B, \diamond'_{\leftarrow})$	(C, D, \diamond)	$(g, <, <)$	$(C, D, \diamond) \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
67	$(g, >, >)$	$(A, B, \diamond'_{\leftarrow})$	$(g, <, <)$	$(B, B, \diamond) \forall B \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$
68	(A, B, \diamond')	$(C, D, \diamond'_{\leftarrow})$	$(g, <, <)$	$(D, D, \diamond), \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
69	(A, B, \diamond')	$(C, D, \diamond'_{\leftarrow})$	(E, E, \diamond)	$(D, D, \diamond), \forall B, D, E \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
70	$(g, >, >)$	$(A, B, <)$	$(C, D, <)$	$(A, B, \leftarrow) \forall C, D \in \Sigma, \forall B \in \Sigma \mid \rho(B) = \lambda,$ $\forall A \in \cup_{i=1}^R \Sigma^i$
71	$(g, >, >)$	(A, B, \leftarrow)	$(C, D, <)$	$(AC, D, <_{\leftarrow}), \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
72	$(g, >, >)$	$(A, B, <_{\leftarrow})$	(C, D, \leftarrow)	$(A, B, <_{\leftarrow}), \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
73	$(g, >, >)$	$(A, B, <_{\leftarrow})$	$(C, D, <_{\leftarrow})$	$(A, B, <_{\leftarrow}), \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
74	$(g, >, >)$	$(A, B, <_{\leftarrow})$	$(C, D, <_{\rightarrow})$	$(A, B, <_{\rightarrow}), \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
75	(A, B, \diamond')	$(C, D, <_{\rightarrow})$	$(E, F, <)$	$(C, D, > \rho(D)), \forall D \mid \rho(D) \neq \lambda, \forall B, D, F \in \Sigma,$ $\forall A, C, E \in \cup_{i=1}^R \Sigma^i$
76	$(g, >, >)$	$(A, B, <_{\rightarrow})$	$(C, D, <)$	$(A, B, \leftarrow) \forall B, D \in \Sigma \mid \rho(B) = \lambda, \forall A, C \in \cup_{i=1}^R \Sigma^i$
77	$(g, >, >)$	$(A, B, <_{\rightarrow})$	$(C, D, <)$	$(A, B, > \rho(B)), \forall B \mid \rho(B) \neq \lambda, \forall B, D \in \Sigma,$ $\forall A, C \in \cup_{i=1}^R \Sigma^i$
78	$(g, >, >)$	(A, B, \leftarrow)	$(C, D, <)$	$(AC, D, <_{\leftarrow}), \forall B, D \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
79	$(A, B, <_{\leftarrow})$	$(Cg, <, <_{\rightarrow})$	$(g, <, <)$	$(g, <, <), \forall B \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
80	$(g, >, >)$	$(A, B, <_{\leftarrow})$	$(Cg, <, <_{\rightarrow})$	$(A, B, <_{\rightarrow}), \forall B \in \Sigma, \forall A, C \in \cup_{i=1}^R \Sigma^i$
81	$(g, >, >)$	$(A, B, <_{\rightarrow})$	$(g, <, <)$	$(A, B, \leftarrow) \forall B \in \Sigma \mid \rho(B) = \lambda, \forall A \in \cup_{i=1}^R \Sigma^i$
82	$(g, >, >)$	$(A, B, <_{\rightarrow})$	$(g, <, <)$	$(A, B, > \rho(B)), \forall B \mid \rho(B) \neq \lambda, \forall B \in \Sigma,$ $\forall A \in \cup_{i=1}^R \Sigma^i$
83	$(g, >, >)$	$(g, >, >)$	$(A, B, <_{\leftarrow})$	$(g, >, >), \forall B \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$
84	$(g, >, >)$	$(g, >, >)$	$(A, B, <_{\rightarrow})$	$(g, >, >), \forall B \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$
85	$(A, B, <_{\leftarrow})$	$(C, D, <_{\rightarrow})$	$(E, F, <)$	$(C, D, <), \forall B, D, F \in \Sigma \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
86	$(A, B, <_{\rightarrow})$	$(C, D, <)$	$(E, F, <)$	$(C, D, <), \forall B, D, F \in \Sigma \forall A, C, E \in \cup_{i=1}^R \Sigma^i$

87	(A, B, \leftarrow)	(C, D, \leftarrow)	(E, F, \leftarrow)	$(C, D, \leftarrow), \forall B, D, F \in \Sigma \forall A, C, E \in \cup_{i=1}^R \Sigma^i$
88	$(A, B, > C \alpha)$	(D, E, \leftarrow)	$(g, \leftarrow, \leftarrow)$	$(D, E, > \alpha \rho(E)) \forall E \in \Sigma \rho(E) \neq \lambda,$ $\forall A \in \cup_{i=1}^R \Sigma^i, \forall B, C, D, E \in \Sigma, \forall \alpha \in \Sigma^+ \alpha \leq k$
89	$(A, B, > C)$	(D, E, \leftarrow)	$(g, \leftarrow, \leftarrow)$	$(D, E, \leftarrow) \forall E \in \Sigma \rho(E) = \lambda, \forall B, C, E \in \Sigma$ $\forall A, D \in \cup_{i=1}^R \Sigma^i$
90	$(g, >, >)$	(A, B, ϕ')	(C, D, ϕ')	$(A, B, \phi'), \forall B, D \in \Sigma \forall A, C \in \cup_{i=1}^R \Sigma^i$
91	$(g, >, >)$	(A, B, ϕ')	$(C, D, \phi' \leftarrow)$	$(A, B, \phi' \leftarrow), \forall B, D \in \Sigma \forall A, C \in \cup_{i=1}^R \Sigma^i$
92	$(g, >, >)$	$(g, >, >)$	(A, B, ϕ')	$(g, >, >), \forall B \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$
93	$(g, >, >)$	$(g, >, >)$	$(A, B, \phi' \leftarrow)$	$(g, >, >), \forall B \in \Sigma, \forall A \in \cup_{i=1}^R \Sigma^i$

Table11-1: Transition function of the cellular automaton equivalent to a DIL system.

11.3. Examples

The algorithm must take into account other cases, besides those previously shown, as we can see in the following examples, where different rows in the transition table are used.

Example 1

The DIL system is defined as the following:

$$S = \{\Sigma = \{A, B\},$$

$$P = \{AAA ::= AA, AAB ::= BA, BAA ::= BA, BAB ::= AB, gAA ::= B, gAB ::= A, g ::= B,$$

$$BAg ::= A, xBy ::= \lambda \quad \forall x, y \in \Sigma\}, BBABB\}.$$

The cellular automaton equivalent to the DIL system (S) is defined as follows:

$$C_{S,n} = (G_{S,n}, G_{0S,n}, V_{S,n}, Q_{S,n}, f_{S,n}, T_{S,n})$$

where

- $G_{S,n}$

is an infinite one-dimensional grid of automata

- $Q_{S,n} = \{(\text{context}, \text{symbol}, \text{displacement})$

where

$$\text{context} \in \{g\} \cup \{\alpha \gamma | \alpha \in \Sigma^*, \gamma \in \{g\}^* \wedge |\alpha \gamma| \leq \max_{i \in \{0, \dots, n\}} \{|h^i(\omega)||\}\},$$

$$\text{displacement} \in \{\leftarrow, \leftarrow\leftarrow, \leftarrow\leftarrow\leftarrow, \leftarrow\leftarrow\leftarrow\leftarrow, \phi, \phi', \phi' \leftarrow\} \cup \{> \alpha | \alpha \in \Sigma^* \wedge |\alpha| \leq \max_{A \in \Sigma} \{|\rho(A)|\} \times$$

$$\max_{i \in \{0, \dots, n\}} \{|h^i(\omega)||\}\}$$



symbol $\in \Sigma \cup \{<, >\}$

- $G_{0S,n}$ is the initial configuration, which is defined as follows:

$$G_{0S,n}[i] = \begin{cases} (g, >, >) & \text{if } i < 0 \\ (s, s, \diamond) & \forall i \in [1, |\omega|], s = \omega[i] \\ (g, <, <) & \text{otherwise} \end{cases}$$

- $V_{S,n} = (3, (-1, 0, 1))$.
- The set of states is

$$Q_c = \left\{ \begin{array}{l} (g, >, >), (g, <, <), (A, A, \diamond), (B, B, \diamond), (A, A, \diamond'), (B, B, \diamond'), (A, B, \diamond'), \\ (BBA, A, \diamond'), (A, A, <), (B, B, <), (BBA, A, > \alpha), (B, B, > \alpha), (B, B, \leftarrow), (A, A, \leftarrow), \\ (BB, B, \leftarrow), (BB, B, \leftarrow), (B, B, \leftarrow), (A, A, \leftarrow), (g, <, \leftarrow), (BBA, A, \leftarrow), (g, <, \leftarrow), \\ (B, B, \leftarrow), (A, A, \leftarrow), (Bg, <, \leftarrow), (B, B, \diamond' \leftarrow), (BBA, A, \diamond' \leftarrow) \end{array} \right\}$$

- $f_{S,n}: Q_{S,n} \times Q_{S,n}^{1+R+1} \rightarrow Q_{S,n}$ is the transition function informally described in the previous section.

The first derivation in this example is shown in figure 11-6.

Example 2

$$S = \{\Sigma = \{A, B\},$$

$$P = \{AAA ::= AA, AAB ::= BA, BAA ::= BA, BAB ::= AB, gAA ::= B, gAB ::= A,$$

$$AAg ::= B, BA g ::= A, xBy ::= \lambda \quad \forall x, y \in \Sigma\}, BBB\}$$

In this example we can see that, after several steps, the first symbol B is deleted and its right context information is accumulated. The deletion continues for all B's until finally the empty word is obtained. The cellular automaton equivalent to this DIL system (S) is shown in figure 11-7.

	1	2	3	4	5	6	7
1	g >(5) >	B B(8) ◇	B B(61) ◇	A A(61) ◇	B B(61) ◇	B B(19) ◇	g <(54) <
2	g >(5) >	B B(8) ◇	B B(61) ◇	A A(61) ◇	B B(21) ◇	B B(23) <	g <(54) <
3	g >(5) >	B B(8) ◇	B B(61) ◇	A A(21) ◇	B B(24) <	B B(26) <	g <(54) <
4	g >(5) >	B B(8) ◇	B B(21) ◇	A A(24) <	B B(25) <	B B(26) <	g <(54) <
5	g >(5) >	B B(20) ◇	B B(24) <	A A(25) <	B B(25) <	B B(26) <	g <(54) <
6	g >(6) >	B B(70) <	B B(25) <	A A(25) <	B B(25) <	B B(26) <	g <(54) <
7	g >(4) >	B B(71) ←	B B(32) <	A A(25) <	B B(25) <	B B(26) <	g <(54) <
8	g >(83) >	BB B(72) ←	B B(40) ←	A A(32) <	B B(25) <	B B(26) <	g <(54) <
9	g >(83) >	BB B(73) ←	A A(44) ←	A A(40) ←	B B(32) <	B B(26) <	g <(54) <
10	g >(83) >	BB B(73) ←	A A(48) ←	B B(44) ←	B B(40) ←	B B(33) <	g <(54) <
11	g >(83) >	BB B(73) ←	A A(48) ←	B B(48) ←	B B(44) ←	B B(41) ←	g <(54) <

12	g > (83) >	BB B (73) <←	A A(48) <←	B B(48) <←	B B(74) <←	Bg <<(79) <→	g < (54) <
13	g > (83) >	BB B(73) <←	A A (48) <←	B B (46) <←	B B (59) <→	g < (54) <	g < (54) <
14	g > (83) >	BB B (73) <←	A A (46) <←	B B (85) <→	B B (31) <	g <<(54) <	g < (54) <
15	g > (83) >	BB B(75) <←	A A(85) <→	B B (86) <	B B(26) <	g <<(54) <	g <<(54) <
16	g > (84) >	BB B (76) <→	A A (25) <	B B (25) <	B B (26) <	g <<(54) <	g <<(54) <
17	g > (4) >	BB B (78) ←	A A (32) <	B B(25) <	B B(26) <	g <<(54) <	g <<(54) <
18	g > (83) >	BBA A(72) <←	A A(40) ←	B B(32) <	B B(26) <	g <<(54) <	g <<(54) <
19	g > (83) >	BBA A (73) <←	B B (44) <←	B B (40) ←	B B(33) <	g <<(54) <	g <<(54) <
20	g > (83) >	BBA A (73) <←	B B (48) <←	B B(44) <←	B B(41) ←	g <<(54) <	g <<(54) <
21	g > (83) >	BBA A(73) <←	B B(48) <←	B B(51) <←	Bg < (80) <→	g <<(54) <	g <<(54) <
22	g > (83) >	BBA A (73) <←	B B (46) <←	B B (59) <→	g < (54) <	g <<(54) <	g < (54) <

	1	2	3	4	5	6	7
23	g) (83) >	BBA A(74) <←	B B(87) <→	B B(31) <	g <(54) <	g <(54) <	g <(54) <
24	g) (84) >	BBA A(77) <→	B B(86) <	B B(26) <	g <(54) <	g <(54) <	g <(54) <
25	g) (7) >	BBA A(34))AB	B B(29) <	B B(26) <	g <(54) <	g <(54) <	g <(54) <
26	g) (92) >	BBA A(10) ◇'	B B(36))B	B B(88) <	g <(54) <	g <(54) <	g <(54) <
27	g) (92) >	BBA A(90) ◇'	B B(13) ◇'	B B(42) ←	g <(54) <	g <(54) <	g <(54) <
28	g) (92) >	BBA A(90) ◇'	B B(63) ◇'	Bg <(64) <→	g <(54) <	g <(54) <	g <(54) <
29	g) (92) >	BBA A(90) ◇'	B B(65) ◇'	g <(54) <	g <(54) <	g <(54) <	g <(54) <
30	g) (92) >	BBA A(95) ◇'	B B(68) ◇'←	g <(54) <	g <(54) <	g <(54) <	g <(54) <
31	g) (93) >	BBA A(62) ◇'←	B B(66) ◇	g <(54) <	g <(54) <	g <(54) <	g <(54) <
32	g) (5) >	A A(8) ◇	B B(19) ◇	g <(54) <	g <(54) <	g <(54) <	g <(54) <

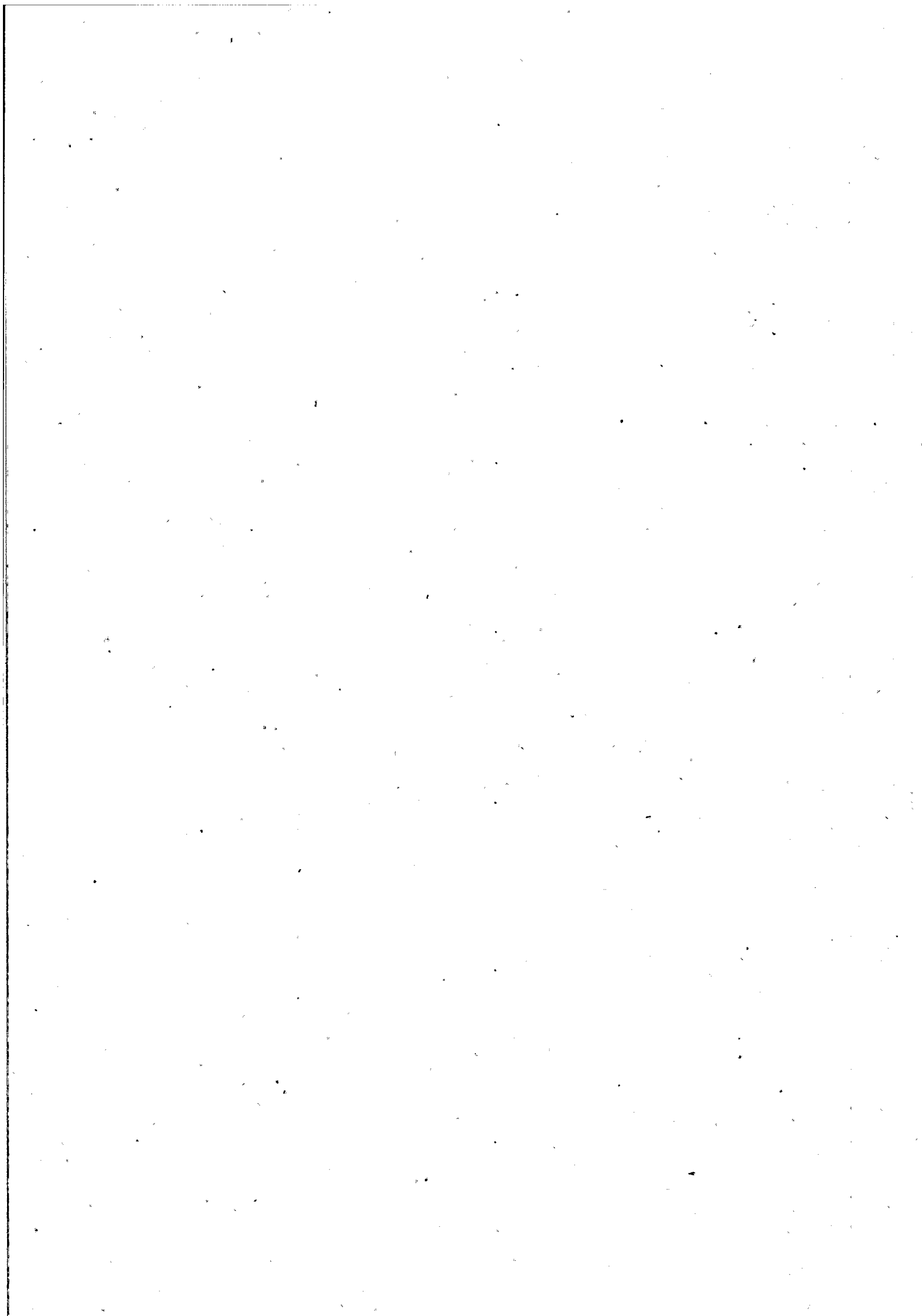
Figure 11-6: Derivation of AB from BBABB in the cellular automaton described in example 1

	1	2	3	4	5	6	7
1	g >(5) >	B(8) B ◇	B(61) B ◇	B(19) B ◇	g(54) < <	g <(54) <	g <(54) <
2	g >(5) >	B(8) B ◇	B(21) B ◇	B(23) B <	g(54) < <	g <(54) <	g <(54) <
3	g >(5) >	B(20) B ◇	B(24) B <	B(26) B <	g(54) < <	g <(54) <	g <(54) <
4	g >(6) >	B(70) B <	B(25) B <	B(26) B <	g(54) < <	g <(54) <	g <(54) <
5	g >(4) >	B(71) B ←	B(32) B <	B(26) B <	g(54) < <	g <(54) <	g <(54) <
6	g >(83) >	BB(72) B ←	B(40) B ←	B(33) B <	g(54) < <	g <(54) <	g <(54) <
7	g >(83) >	BB(73) B ←	B(44) B ←	B(41) B ←	g(54) < <	g <(54) <	g <(54) <
8	g >(83) >	BB B(73) ←	B B(51) ←	Bg <(79) ←	g <(54) <	g <(54) <	g <(54) <
9	g >(83) >	BB B(74) ←	B B(59) ←	g <(54) <	g <(54) <	g <(54) <	g <(54) <
10	g >(84) >	BB B(76) ←	B B(31) <	g <(54) <	g <(54) <	g <(54) <	g <(54) <
11	g >(4) >	BB B(78) ←	B B(33) <	g <(54) <	g <(54) <	g <(54) <	g <(54) <

	1	2	3	4	5	6	7
12	g >(83) >	BBB B(72) ←	B B(41) ←	g <(54) <	g <(54) <	g <(54) <	g <(54) <
13	g >(83) >	BBB B(80) ←	Bg <(79) ←	g <(54) <	g <(54) <	g <(54) <	g <(54) <
14	g >(84) >	BBB B(81) ←	g <(54) <	g <(54) <	g <(54) <	g <(54) <	g <(54) <
15	g >(4) >	BBB B(43) ←	g <(54) <	g <(54) <	g <(54) <	g <(54) <	g <(54) <
16	g >(3) >	BBBg <(57) ←	g <(54) <	g <(54) <	g <(54) <	g <(54) <	g <(54) <
17	g >(2) >	g <(1) <	g <(54) <	g <(54) <	g <(54) <	g <(54) <	g <(54) <

Figure 11-7: Derivation of λ from BBB in the cellular automaton described in example 2

Conclusions & Future Work



Conclusions

One of the main problems in modern *Theoretical Computer Science*, as explained in the introduction (*Motivation and plan of the thesis*), is the design and programming using new programming paradigms and complex systems, such as Lindenmayer Systems and Cellular Automata (CA). The use of genetic techniques is not a new approach to solve this task. Our research group is interested in developing tools which are independent of the domain considered, thus making easier the design of L-systems and CA that exhibit a given behavior.

Several previous works have applied Genetic Algorithms (GA) to the design of L-systems in particular domains. One of the objectives of this thesis is the generalization of these works. We show that it is possible to consider the automatic design of L-systems by genetic techniques as a particular case of genetic programming, and for the first time have applied to L-systems a general-purpose genetic programming tool (Grammatical Evolution, GE) that allows evolutionary automatic programming in an arbitrary language.

As previously explained, GE adds, to standard GA, a grammar directed genotype-phenotype mapping. In this way, the genotype representation and the search engine are independent components of the system. This thesis describes an application that could have been solved by means of standard GA; however, even in this simple case, the genotype-phenotype mapping of GE does not add a significant overhead to the performance of classic GA.

Our group has highlighted the usefulness of inferring interesting properties of a real system from the study of the formal model that simulates it. In [Alf01a][Alf00a] an algorithm was proposed to estimate the fractal dimension of a family of initiator-iterator fractals, from the study of the D0L-systems that represent them. Following this work, this thesis shows that it is possible to solve non-trivial tasks with both theoretical and practical interest (for example, the design of a curve with a given fractal dimension that could be used in the fractal antenna industry) by means of genetic programming techniques applied through the use of grammars. To achieve this objective, a modification of GE was needed, because the original description of GE uses Chomsky grammars instead of Lindenmayer grammars. The fitness of the population in evolution is measured by means of the algorithm mentioned above that estimates the fractal dimension of the curves without the need of drawing them.

The use of GA on binary CA's has been described in detail. Most of these works are, in general, interested in solving a given particular problem (e.g. classification tasks, synchronization problems, etc...). This thesis suggests that GA can be considered a general tool for programming and studying CA's. In our case, Conway's game of life (a bi-dimensional computationally complete binary CA) has been obtained by means of standard GA. We are interested on further studying and analyzing the behavior of bi-dimensional CA's, and this result is a first step.

Our group is also interested in the formal properties of Lindenmayer Systems and Cellular Automata. The directors of this thesis have previously proven that there are Lindenmayer Systems able to simulate any given CA. This thesis is a long step to prove the reverse result: that there are CA's able to generate the same language as any L-System of the following families:

- PD0L
- D0L
- DIL

The length of the strings generated by a PD0L system increases with the number of derivations. They cannot become shorter, because PD0L systems do not allow λ rules (erasing rules). Therefore, the number of symbols may grow exponentially. It is not trivial to decide if it is possible to simulate this derivation process with a finite set of states, because the number of derived words is potentially infinite, all of which belong to the language generated by the PD0L system. Any CA that tries to simulate a PD0L system must provide a mechanism to displace symbols from left to right with a finite set of different states. Such CA's must have a number of different states (big enough, but finite) to represent all possible "displacing substrings". This thesis shows how the problem can be solved by defining, for all n , the "n-equivalence" relationship: a given cellular automaton is n-equivalent to a given PD0L system if and only if it is able to generate (in the same order) the first n words generated by the PD0L system.

Once the equivalence between CA's and PD0L systems is proven, we considered allowing erasing rules (and so proved the equivalence between CA's and D0L systems), and finally took in consideration context-sensitive L-grammars (and so proved the equivalence between CA's and DIL systems).

Open lines of work

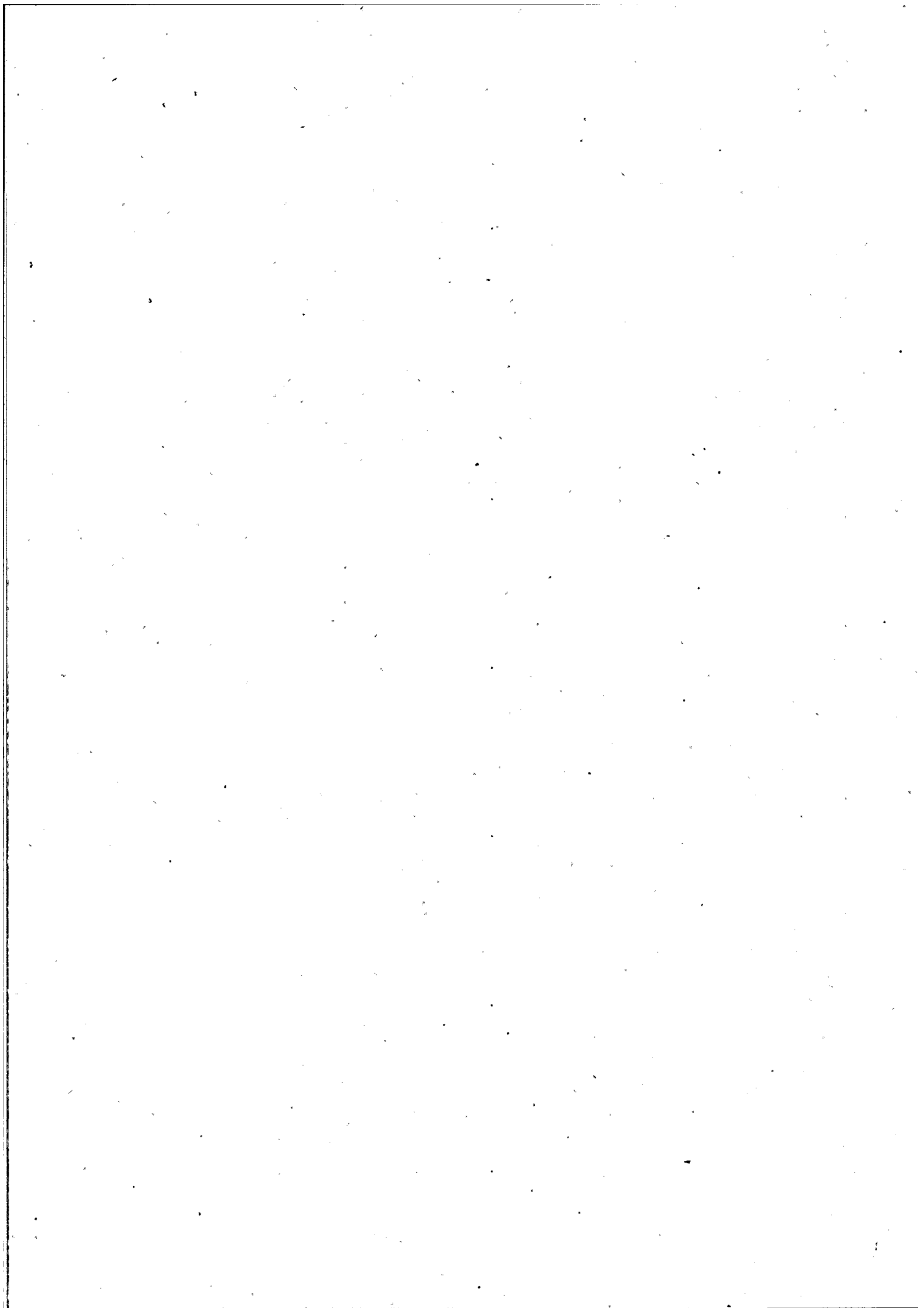
Our first experiment with GE and L-systems could be useful in an industrial environment. There are several companies that currently manufacture so called "fractal antennas". They are really finite approximations to some well-known initiator-iterator fractals (e.g. von Koch snowflake, Sierpinski gasket or Peano's curve) which are only a few among many others automatically generated by our algorithm. Our fitness function is based on the fractal dimension, estimated from the DOL systems that represent the fractals. It seems possible to design a different fitness function based on the electromagnetic parameters of these curves when they are used as antennas.

We hope that GE will offer its real benefits with syntactically more complex L-systems, because it reduces the possibility of generating wrong phenotypes, when there exists a context free grammar that describes them. This approach should also be applied to the design and programming of other formal computational models, such as DNA inspired computing devices, and other bio-inspired formal systems.

Different authors have exhaustively studied the behavior of one-dimensional binary CA's. Sometimes it is possible to characterize and predict it by calculating a single numerical parameter (such as Langton's λ parameter). Our group plans to use GA as a tool to study similar properties and parameters in more complicated CA's (bi-dimensional, for instance). It should be possible to design genetic searches to find CA's with complex behavior and simultaneously estimate several candidate parameters to check their validity for task under study.

Besides the families of L-systems considered in this thesis, whose equivalence to CA's has been proved, there are additional classical families (L systems with tables and extensions) for which those equivalences must still be proved. We plan to face in the future the design of CA's that simulate L-systems with tables and extensions, as well as other non-classic families of L grammars.

**References
&
Bibliography**





REFERENCE & BIBLIOGRAPHY:

[Abr02] Patrice Abry, Richard Baraniuk, Patrick Flandrin, Rudolf Riedi, Darryl Veitch 2002. "The Multiscale Nature of Network Traffic: Discovery, Analysis, and Modelling" IEEE Signal Processing Magazine vol 19, no 3, pp 28-46.

[Abd03a] Abu Dalhoum. A, Ortega Alfonso, Alfonseca. M., 2003.
"Cellular Automata equivalent to PD0L Systems"
International Arab Conference on Information Technology
(ACIT 2003), 20-23 Dic. 2003, Alexandria, Egipto. Pub: Proceedings, pp.819-825.

[Abd03b] Abu Dalhoum. A, Ortega Alfonso, Alfonseca. M., 2003
"Cellular automata equivalent to D0L systems." WSEAS Transactions on Computers, Vol. 4:2, p.1159-1167, Oct. 2003.

[Alf97] Alfonseca. M, Ortega, A., 1997
"A study of the representation of fractal curves by L systems and their equivalences", IBM Jr. of Res. and Dev., Vol. 41:6, p.727-736, Nov.

[Alf99] Alfonseca, M. 1999
"Programming Cellular Automata in APL2", APL Quote Quad (ACM SIGAPL), Vol. 30:1, p. 27-30, Sep. 1999

[Alf00a] Alfonseca. M and Ortega. A. 2000
"Using APL2 to Compute the Dimension of a Fractal Represented as a Grammar", *APL Quote Quad*, 30, No. 4, 13-23 .

[Alf00b] M. Alfonseca. M, Ortega. A., 2000
"Representation of some cellular automata by means of equivalent L system's", *Complexity International*, ISSN: 1320-0682, Volume 7, p.1-16, Feb. 2000.

[Alf01a] Alfonseca, M ; Ortega , A, 2001
"Determination of fractal dimensions from equivalent L systems".
IBM Jr. of Res. and Dev., Vol. 45:6, p. 797-805, Nov.

[Alf01b] Alfonseca. M, Dalhoum. A. Abu, Ortega. A., 2001
"Evolving the game of life with a genetic algorithm", Proceedings of the 3rd Middle East Symposium on Simulation and Modelling (MESM'2001), SCS Publications, p.165-169, ISBN: 1-56555-230-X.

[Alf03] Alfonseca. M.; Ortega, A.; Suárez, A., 2003
"Cellular automata and probabilistic L systems: An example in Ecology, en Grammars and Automata for String Processing: from Mathematics and Computer Science to Biology", and Back, ed. C. Martin-Vide & V. Mitrana, Taylor and Francis Publishers. pp. 111-120 Marzo 2003. ISBN: 0415298857.

- [Bar88] Bransley, M., 1988.
 "Fractals every where". Academic press, San Diego.
- [Bau99] Bauer, W and Mackenzie, C.D., 1999
 "Cancer Detection via determination of fractal cell dimension,"
 Workshop on computational and theoretical biology .
- [Boe91] Boerlijst M. and Hogeweg P. 1991
 Self-structuring and selection: Spiral waves as a substrate for prebiotic evolution. In
 C. G. Langon, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*,
 pages 255-276, Addison-Wesley.
- [Bur70] Burks, A.W., 1970
 "Essays on Cellular Automata". (Univ. of Illinois Press, Urbana, IL, 1970).
- [Cap01] Capcarrere, M.S, Sipper, M, 2001
 "Necessary conditions for density classification by Cellular automata":
Physical Review E, Vol. 64, No 3, September 2001. pp1-4.
- [Cha97] Chaudhuri, P.E et al. 1997
 "Additive cellular automata theory and applications Vol.1 IEEE press advances and in
 circuits and systems series. IEEE press, Piscataway, NJ.
- [Con] Conway, J.H., Berlekamp, E.R., Guy, R.K.,
Winning ways for the mathematical plays. New York: Academic Press, Vol 2, Cap.25
- [Cul90] Culik II.K, Hurd .L.P, Yu. S, 1990.
 "Computation theoretic aspects of cellular automata". *Physica D*, 45:396-403, 1990.
- [Dan02] Danzer, K; van Staden J.F and Burns, D.T, 2002
 "Concepts and Applications of the Term 'Dimensionality' in Analytical Chemistry",
Pure Applied Chemistry, 74, No. 8, 1479-1487.
- [Dao98] Daoudi, K, Vehel, J. Levy and Meyer, Y., 1998.
 "Construction of continuous functions with prescribed local regularity", *Journal of
 Constructive Approximation*, 14, No. 3 349-385 (1998).
- [Del98] Delorme, M., 1998.
 "An Introduction to Cellular Automata", *Research Report*, No. 98-37. Ecole Normale
 Supérieure de Lyon.
- [Erm93] Ermentrout, G.B, Edelstein-Keshitein, L. 1993.
 "Cellular automata approaches to biological modeling". *Journal of theoretical Biology*,
 160:97-133.
- [Fla98] Flake, K., 1998.
 "The computational beauty of Nature. Computer exploration of fractals, chaos, complex
 systems, and adaptation". The M.I.T Press
- [Goe91] Goel, N. S., Rozanel, I, 1991.
 "Some non-biological applications of L-systems" *Int. J. General systems*, Vol.18.

- [Gol89] Goldberg.D, 1989
 "Genetic Algorithms in Search, Optimization, and Machine Learning". Addison-Wesley Reading, MA.
- [Gol89] Goldberg.D, 2002
 "The Design of innovation", Kluwer Academic Publishers.
- [Gui98] Guiheneuf. B, Jaffard.S and Vehel. J.Levy, 1998.
 "Two Results Concerning Chirps and 2-microlocal Exponents Prescription", *Applied and Computational Harmonic Analysis*, 5, 487-492 (1998).
- [Har01] Harte,D.S, 2001 N
 "Multifractals:theory and applications". Chapman and Hall/ CRC, Boca Raton.
- [Hol67] Holland J H, 1967.
 "Nonlinear environment permitting efficient adaptation". Academic Press, New-York,N.Y.
- [Hol75] Holland.J.H, 1975.
 "Adaption of Natural and Arti_cial Systems". University of Michigan Press, Ann Arbor, Michigan.
- [Hor03] Hornby.G.S, 2003.
 "Generative Representations for Evolutionary Design Automation". A dissertation presented to the Faculty of the Graduate School of Arts and Sciences, Brandeis University Department of Computer Science (advisor: Jordan B. Pollack), in partial fulfillment of the requirements for the Ph.D. degree .
- [Hor99] Hordijk, W, 1999
 "Dynamics, Emergent Computation, and Evolution in Cellular automata" PhD thesis New Mexico University, Albuquerque, New Mexico.
- [Ian96] Iannaccone. P.M and Khokha .M,1996
 eds., "Fractal Geometry in Biological Systems: An Analytical Approach", CRC Press, Boca Raton, Fla.
- [Ima98] Imai.K, and Morita.K.K,1998
 "A computation-universal two-dimensional 8-state triangular reversible cellular automaton", Proc. of the Second colloquium on Universal Machines and Computations, Vol II 90-99.
- [Int] <http://calsses.yale.edu/fractals/randomfrac/>
- [Jac94] Jacob.C, 1994.
 "Genetic L-System Programming." PPSN III – Parallel Problem Solving from Nature, International Conference on Evolutionary Computation, Springer-Verlag, Berlin, Lecture Notes in Computer Science 866, 334 – 343 .
- [Jac96a].Jacob.C, 1996.
 "Evolving evolution programs:genetic programming and L-systems".

Proc. First Annual Conference on Genetic Programming, Stanford, USA (1996), MIT Press, pp. 107-115.

[Jac96b] Jacob, C, 1996.

Evolution Programs Evolved in: H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.), Proc. PPSN-IV, Parallel Problem Solving from Nature IV, Lecture Notes in Computer Science 1141, Berlin, GERMANY, Springer, 1996, pp. 42-51.

[Jaf95] Jaffard, S. 1995

"Functions with Prescribed Hölder Exponent", *Applied and Computational Harmonic Analysis*, 2, No. 4, 400-401 (1995).

[Kar95] Kari, J., 1995

"Cellular Automata. An Introduction, in: G. Paun, ed., *Artificial Life: Grammatical Models*" (Black Sea Univ. Press, Bucharest, 1995).

[Koz89] Koza, J. R., 1989.

Hierarchical genetic algorithms operating on populations of computer programs. In proceedings of 11 International Joint Conference on Artificial Intelligence, San Mateo, CA, Morgan Kaufmann.

[Koz92] Koza J.R., 1992

"Genetic Programming: On the Programming of Computers by Means of Natural Selection". MIT Press, Cambridge, Massachusetts.

[Kru78] Krudyumov, G. P. G., L Levin. 1978

"One dimensional uniform arrays that wash out finite islands" probl. Peredachi Inform

[Law66] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. 1966.

Artificial intelligence through Simulated Evolution. John-Wiley, New-York.

[Lan84] Langton G. G. 1984

Self-reproduction in cellular automata. *Physica D*, 10:135

[Lan90] Langton, G.G., 1990

"Computation at the edge of chaos : Phase transitions and emergent computation" *Physica D*, vol.4, pp.12-37.

[Lin68] Lindenmayer, A., 1968

Mathematical Models for Cellular Interactions in Development (two parts), *J. Theor. Biol.* 18, 280-315.

[Lin90] Lindgren, K. and Nordahl, M., 1990

Universal computation in simple one dimensional cellular automata, *Complex Systems*, 4, 299-318.

[Li97] Li Hai, Ray Liu K. J., Shih-Chung B. Lo, 1997

"Fractal modeling and Segmentation for the Enhancement of Microcalcifications in Digital Mammograms" *IEEE Trans. Medical Imaging*, 16(6):785—798.

- [Maj99] Majumdar S., Lin J., Link T., Millard J., Augat P. 1999.
 "Fractal analysis of radiographs: Assessment of trabecular bone structure and prediction of elastic modulus and strength". *Med.Phys.* 26(7).
- [Man77] Mandelbrot.B.B,1977.
 "The Fractal Geometry of Nature", W.H.Freeman and Company, New York.
- [Man90]Manneville. P, Boccara. N, Vichniac.Y, and Bidaux. ,1990.
 "Cellular automata and modeling of complex physical systems". Springer- Verlag,1990.
 Volume 46 of Springer-Verlag proceedings in Physics.
- [Mar86]Margolus. N, Toffoli.T, Vichniac. G.1986.
 "Cellular automata supercomputers for fluid -dynamics modeling". *Physics Review Letters*, 56 (16):1694-1696, 1986.
- [Maz99] Mazoyer. J; Terrier V., 1999.
 Signals in one-dimensional cellular automata. *Theoretical Computer Science* 217 (1999) 53-80
- [Mel01] Mela,K ;Louie.J.N;2001
 "Correlation length and Fractal dimension interpretation from seismic data using variograms and power spectra," *Geophysics*, 66, No. 5, 1372-1378.
- [Mes99] Messina.J.P, Walsh.S.J, Valdivia.G, Taff. G,1999
 The application of cellular automata modeling for enhanced landcover classification in the Ecuadorian Amazon, IV International Conference on GeoComputation (1999).
- [Mit93] Mitchell. M, Harber. P. T, Crutchfield. J. P, 1993
 "Revisiting the edge of chaos: Evolving cellular automata to perform computations" complex systems, pp.89-130 ,no. 7 .
- [Mit94a] Mitchell. M, Crutchfiel. J. P, Harber. P. T. 1994
 Evolving Cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361-391, 1994.
- [Mit94] Mitchel.M, Harber .P.T, Crutchfield. J .P, 1994
 " Dynamics , Computation ,and the edge of chaos : A re-examination " In G.Cown ,D.Pines and D.Melzner,(eds), *Integrative Themes ,Santa Fe Institute Studies in the study of complexity ,Proceedings Volume 19.*Reading ,MA: Addison-Wesley.
- [Mit96] Mitchell,M., 1996
 An introduction to genetic algorithms, MIT press.
- [Mon01] Mondoloni.S and Liang.D, 2001
 "Airspace Fractal dimension and applications," *3rd USA/Europe ATM R& D Seminar* .
- [Mor92] Morita. K,1992.
 Computation-universality of one-dimensional one-way reversible cellular automata, *Inform. Process. Lett.*, 42 (1992) 325-329.

- [Nag94] Nagel.K and Scheicher,.A ,1994.
Microscopic traffic modeling on parallel high performance computers, *Parallel Computing*, 20,125:146.
- [Neu66] Neumann. J .von, 1966
Theory of self-reproducing automata, edited and completed by A. W. Burks (University of Illinois Press, Urbana, IL).
- [Neil03] O'Neill M. and Ryan C., 2003 "Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language" Kluwer Academic Publishers. 2003
- [Neil01] O'Neill M and Ryan,C.2001
"Grammatical Evolution". *IEEE Transactions on Evolutionary Computation*. 5, Np.4, 349-358 (2001).
- [Neil99a] O'Neill,M and Ryan. C .1999.
"Evolving Multi-line Compilable C Programs". In *Proceedings of the Second European Workshop on Genetic*, Berlin, Germany: Springer-Verlag, *Lecture Notes in Computer Science* 1598, 83-92 (1999).
- [Neil99b] O'Neill, M and Ryan, C. 1999
"Under the Hood of Grammatical Evolution". In *GECCO '99: Proceedings of the Genetic and Evolutionary Computation Conference 1999*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. San Mateo, CA: Morgan Kaufmann, 2, 1143-1148 (1999).
- [Neil99c] O'Neill M and Ryan C. 1999
"Genetic code degeneracy: implications for grammatical evolution and beyond". In *ECAL'99: Proceedings of the Fifth European Conference on Artificial Life*, Lausanne, Switzerland, 149-153 (1999).
- [Nor89] Nordahl. M.G,1989.
Formal languages and finite cellular automata.*Complex systems*, 3:63-78,1989
- [Och98] Ochoa.G, 1998.
"On Genetic Algorithms and Lindenmayer Systems." *Proc. PPSN IV*, Amsterdam, Springer-Verlag, *Lecture Notes in Computer Science* 1498, 335-343.
- [Ort03a] A. Ortega, A.A. Dalhoum, M. Alfonsoeca, 2003
"Cellular Automata equivalent to DIL Systems"
5th Middle East Symposium on Simulation and Modelling(MESM 2003), Eurosim, 5-7 Ene. 2004, Sharjah, Emiratos Arabes Unidos.Pub: Proceedings, ISBN: 90-77381-06-6, pp.120-124.
- [Ort02a] A.Ortega.A,Cruz..M.de la,Alfonseca. M, 2002
" Parametric 2-dimensional L Systems and recursive fractal images: Mandelbrot set, Julia sets and biomorphs", *Computers and Graphics* (Elsevier, *SCI JCR* 0.438), vol. 26, p.143-149.

- [Ort03b] A. Ortega, A.A. Dalhoum, M. Alfonseca, 2003
 "Using grammatical evolution to design curves with a given fractal dimension"
 Proceedings of the 5th International Conference on Enterprise Information Systems
 (ICEIS 2003), p. 395-398; ISBN: 972-98816-1-8.
- [Ort03c] Ortega.A, Dalhoum.A.A, Alfonseca.M,2003
 "Grammatical evolution to design fractal curves with a given dimension", IBM Jr. of
 Res. and Dev. (SCI JCR 2.560), Vol. 47:4, p. 483-493 July. 2003.
- [Otr] Ortega A,2000.
 Equivalencias entre algunos sistemas complejos: Fractales, Autómatas Celulares y
 Sistemas de Lindenmayer. Universidad Autónoma de Madrid,Ph.D Thesis.
- [Ort02b] Ortega. A, de la Cruz. M, Alfonseca. M. 2002.
 "Parametric 2-dimensional L Systems and recursive fractal images: Mandelbrot set,
 Julia sets and biomorphs"; Computers and Graphics (Elsevier), 26 No. 1, 143-149.
- [Pac88] Packard et al ,1988
 "Adaptation Toward The Edge Of Chaos", Dynamic patterns in complex systems "world scientific
 ,Singapore , pp.293-301
- [Pap80] Papert, S. 1980.
 Mindstorms: children, computers, and powerful ideas. New York: Basic Books.
- [Par94a] Paredis.,J., 1994a,
 Steps towards Coevolutionary Classification Neural Networks, Proceedings Artificial
 Life IV, R. Brooks, P. Maes (eds), MIT Press / Bradford Books.
- [Par94b] Paredis,J.,1994b.
 Coevolutionary Constraint Satisfaction, Proceedings of the Third Conference on Parallel
 Problem Solving from Nature (PPSN 94), Lecture Notes in Computer Science, vol. 866,
 Davidor, Y., Schwefel, H-P., Manner, R. (eds.), Springer Verlag.
- [Par97a] Paredis.J; Westra.J.R, 1997.
 Coevolutionary Computation for Path Planning, Proceedings 5th European Congress on
 Intelligent Techniques and Soft Computing (EUFIT 97), H.-J. Zimmermann (ed.),
 Verlag Mainz, Aachen .
- [Par97b] Paredis. J ,1997
 "Co-evolving cellular automata : Be a ware of the red queen" in Proc, 7th Int. Conf
 Genetic Algorithms ,T Back ,Ed .San Mateo ,CA :Morgan Kaufmann , PP.393 – 400.
- [Par98] Paredis,J. 1998.
 Coevolutionary Process Control., Proceedings of the International Conference on
 Artificial Neural Networks and Genetic Algorithms (ICANN97), G. D. Smith (ed.),
 Springer, Vienna 1997.
- [Pru98]Prusinkiewicz.P,Lindennayer.A, Hanan.J.1998.
 Developmiental models of herbaceous plants for computer imagery purposes. Computer
 Graphics.22(4).

[Pru94] Prusinkiewicz P., James M., Mech R. 1994.
Synthetic Topiary. Computer graphics, ACM SIGGRAPH Annual conference series. (Orland, Florida). pp.351-358

[Pru95] Prusinkiewicz P., Hammal M., Mech R., Hanan Jim 1995.
The artificial life of plants, ACM SIGGRAPH, 95, notes, pages 1-1-1-38 ACM press

[Pru90] Prusinkiewicz P., Lindenmayer A. 1990.
The algorithmic beauty of plants. Springer-Verlag, New York, With J.S. Hanan, F.D. Frachhia, D.R. Fowler, M.J.M de Boer, and L. Mercer.

[Ray98a] Ryan C., Collins J.J., and O'Neill M. 1998
"Grammatical Evolution: Evolving Programs for an Arbitrary Language". In EuroGP'98: Proceedings of the First European Workshop on Genetic Programming. Berlin, Germany: Springer-Verlag, Lecture Notes in Computer Science 1391, 83-95 (1998).

[Ray98b] Ryan C., O'Neill M. "Grammatical Evolution: A Steady State Approach". In Proceedings Joint Conference on Information Sciences, Research Triangle Park, NC, 419-423 (1998).

[Ray98c] Ryan C., O'Neill M. "Grammatical Evolution: A Steady State Approach". In Genetic Programming: Proceedings of the 3rd Annual Conference, J. R. Koza, W. Banzhaf, L. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba and R. L. Riolo, Eds Cambridge, MA, 180-185 (1998).

[Ray98d] Ryan C., O'Neill M., and Collins J.J. 1998
"Grammatical Evolution: Solving Trigonometric Identities." In Mendel'98: Proceedings of the 4th International Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, and Rough Sets. Brno, Czech Republic: Tech. Univ. Brno, 111-119.

[Rec73] Rechenberg I. 1973.
Evolutionsstrategien: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Fromman-Holzboog, Verlag Stuttgart.

[Rec65] Rechenberg I. 1965.
Cybernetic solution path of an experimental problem. Royal Aircraft Establishment Library Translation. 1122.

[Rel02] Reljin I.S.; Reljin Branimir D, 2002
"Fractal geometry and multifractal in analyzing and processing medical data and images" Archive of Oncology; 10(4):283-93. Institute of Oncology Sremska Kamenica, Yugoslavia.

[Rib01] Ribeiro V.J.; Riedi R.H and Baraniuk R.G, 2001.
Wavelets and Multifractals for network traffic modeling and inference Proceedings ICASSP Salt Lake City, Utah.

[Rie99] Riedi R.H.; Crouse M.S, Ribeiro V.J, and Baraniuk R.G, 1999.
A multifractal Wavelet Model with Application to Network Traffic. IEEE Special Issue on Information Theory, 45, 992-1018.

-
- [Sar98]Sarker.P, Burua. R 1998
"The set of reversible 90/150 cellular automata is regular" *Discrete Appl. Math* vol.84. pp.199-213
- [Sar00]Sarkar .P , March 2000
"A Brief history of cellular automata" *ACM Computing Surveys* , pp 80-107 , Vol 32,No .1.
- [Sar01]Sarvotham.S.;Riedi.R.H, and Baraniuk.R.G ,2001.
Network Traffic Analysis and Modeling at the Connection Level.
Proceedings IEEE/ACM SIGCOMM Internet Measurement Workshop 2001,
San Francisco, CA.
- [Sch93]Schadschneider and Schreckenberg.M, 1993.
Cellular automaton models and traffic flow, *J. Phys. A: Math. Gen.* 26 (1993)
L679-683.
- [Ser90]Serra. M et al . 1990
"The analysis of one dimensional cellular automata and their aliasing properties "
IEEE Trans CAD/ICAS vol.9 pp.767-778.
- [Sip96a] Sipper.M, Ruppin. E 1996
"Co-evolving architectures for cellular machines" *Physica D*,1996.
- [Sip96b] Sipper .M 1996
"Co- evolution non-uniform cellular automata to perform computations".
Physica D,92:193-208,
- [Smi84] Smith .A 1984. Plants, fractals and formal languages. *Computer Graphics* 18
(July) 1-10.
- [Tay02] R.P. Taylor, B. Spehar, C.W.G Clifford and B.R Newell,2002
"The Complexity of Pollack's Dripped Fractals" *Proceedings of the International
Conference of Complex Systems* .
- [Terr99] Terrier V., 1999.
Two-dimensional cellular automata recognizer. *Theoretical Computer Science* 218
(1999) 325-346.
- [Terr91] Terrier V., 1991.
Decidabilite en arithmetiques faibles. Temps reel sur automates cellulaires. PhD. Thesis
- [Tof77] Toffoli. T,1977
Computation and construction universality of reversible cellular automata, *J. Comput.
Syst. Sci.*, 15 :213-231.
- [Tor01] F.Torrens.2001.
"Fractals Hybrid orbitals in Protein Models," *Complexity International*, 8 .
- [Tra96]Traxler. G, Gervautz. M. 1996
"Using Genetic Algorithms to Improve the Visual Quality of Fractal Plants generated
with CSG-PL Systems." *Proceedings of The Fourth International Conference in Central
Europe on Computer Graphics and Visualization'96* Ed. N.Magnenat-Thalmann

V.Skala, University of West Bohemia, Plzen, Czech Republic.

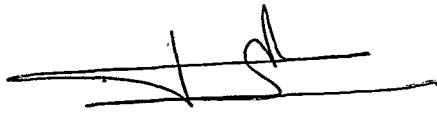
[Vin01] Vinoy.K.J;Jose.K.A;Varadan.V.K and Varadan,V.V, 2001
"Hilbert Curve Fractal Antenna: a Small Resonant Antenna for VHF/UHF
Applications," Microwave and Optical Technology Letters, 29 No.4, 215-219 .

[War99] Ward.D.P, 1999
An optimized cellular automata approach for sustainable urban development in rapidly
urbanizing regions, IV International Conference on GeoComputation .

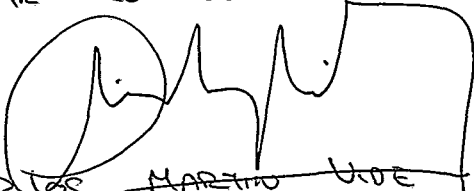
[Wei91] Weisbuch. G,1991. Complex systems dynamics. A lecture notes volume in the
Santa Fe Institute studies in the sciences of complexity (Addison-Wesley, Reading,
Mass.

[Wo194] Wolfram. S, 1994.Cellular Automata and complexity.Addison-Wesley,1994.

Reunido el tribunal que suscribe en el día
de la fecha, acordó calificar la presente Tesis
doctoral con SOBRESALIENTE CUM LAUDE (POR UNANI-
Madrid, 26-4-2004 MIDAD)



FDO: PILAR RODRIGUEZ MARIN



FDO: CARLOS MARTIN UDE

Victor R. Mira

FDO: VICTOR MIRAMA



FDO: BLANCA CASES GUTIERREZ

A. S. Urrecho

FDO: ALEJANDRO SIERRA URRECHO

