DEPARTAMENTO DE INFORMÁTICA Y AUTOMÁTICA
FACULTAD DE CIENCIAS

Intelligence in Formal Machines

VNiVERSiDAD
Ð SALAMANCA

Septiembre 2015

PhD Thesis
TESIS DOCTORAL

Paulo Alexandre Andrade Vieira

Informática y Automática
Universidad de Salamanca

Director
Dr. Juan Manuel Corchado Rodríguez
Dr. Sigeru Omatu

The PhD thesis entitled “Intelligence in Formal” Machines that is presented by Paulo Alexandre Andrade Vieira with the aim to obtain the degree of “Doctor en Informática y Automática por la Universidad de Salamanca” was prepared under the direction of Professor Dr. Juan Manuel Corchado Rodriguez, professor at the Department of ”Informática y Automática de la Universidad de Salamanca” and the Professor Dr. Sigeru Omatu Professor at the Department of Electronics, Information and Communication, Faculty of Engineering, Osaka Institute of Technology.

Salamanca, Septiembre de 2015

Los directores	El doctorando
Fdo: Dr. Juan Manuel Corchado Rodríguez Profesor Titular de Universidad Informática y Automática Universidad de Salamanca	Fdo: Paulo Alexandre Andrade Vieira Assistente do 2º triénio (equiparado) Institute Polytechnic of Guarda, Portugal

Fdo: Dr. Sigeru Omatu
Professor of at the Department of Electronics,
Information and Communication
Faculty of Engineering
Osaka Institute of Technology

To my son Diogo and my wife Fatima with love

Abstract

The aim of this work is to find a way to define and measure Intelligence in computational formal systems. In order to do this I made several types of considerations: first, the concept of "being Intelligent" assumes clear notions and is measured of acceptable way only in human beings; second, there is a growing and savage use of the concept "being Intelligent" for classify the behavior of other biological beings; third, nowadays the word "intelligence" is often used to classify the behavior of non biological beings such as, machines, environments and so on. All of these entities have one thing in common, they are computational systems; fourth, the computational systems are computational formal systems, consequently they can be described using mathematical formalism; fifth, there are a lot of computational formal systems known and there are mathematical theories for working with them that allow to establish relations among them; sixth, the problem of defining and measuring intelligence in formal machines is a problem in the context of following fields: artificial intelligence, theory of computation, complexity theory and category theory; and finally the seventh, in human beings, there is a measure called Intelligence Quotient (IQ) to measure their intelligence.

I gathered all of these considerations and I created a new formalism that is a new formal computational system. I called it Formal Machine (FM). The objective was that all computational formal systems would be rewritten in the new formalism and that in that process they would not lose their mathematical structure. Thus, to satisfy that requirement was necessary to study in depth several computational systems. After, I used the Category Theory and I defined what means to rewrite a computational system to a FMs without lose their mathematical structure. Then I selected the behaviors in humans that are considered intelligent and I used them for doing analogies with Formal Machines. Those analogies served to define and measure "Intelligence" in the new formalism. By doing this for Formal Machines I did this for all computational systems. Thus I created a quotient to measure intelligence in machines, The Machine Intelligent Quotient (MIQ). The MIQ is an analogy of the IQ in humans.

To transform the computational formal systems to the new formalism I created the notion of drives. I wrote algorithms to transform computational formal system to the new formalism. I wrote algorithms for Turing Machines, Push-down Automata, Finite Automata, Neural Networks and other computational systems. I called "drives" to the implementation of these algorithms. I built drives for Finite Automata and Neural Networks.

I also built a software to simulate formal computational systems. This software is called Generator of Universes and Simulator of Formal Machines (GU_SFM). In the moment, in the software is only possible to simulate the behavior of Turing Machines, Push-down Automata and Finite Automata.

To use the new formalism in computation I wrote three APIs that will be used by developers, one for developing in desktops, other for developing with micro-controllers and another for developing in google cloud. I implemented two games, the tic-tac-toe and the four in line game, in each one of the games one the players is the new formalism and the other is a human being. In both implementations it was possible to verify that the new formalism is a good computational formal system to solve engineering problems. I also developed an electronic board and an information system that is able to smell environments. In this information system can be found an implementation of the new formalism in the google cloud.

To validate the MIQ measures I made a statistical study that involved 1000 back-propagation neural networks and a drive projected for them.

Resumen

El objetivo de este trabajo es encontrar una forma de definir y medir la inteligencia en los sistemas formales computacionales. Para hacer esto yo hice varios tipos de consideraciones: en primer lugar, el concepto de “ ser inteligente ” asume nociones claras y se mide de manera aceptable sólo en el ser humano; segundo, hay una creciendo y salvaje uso del concepto “ ser inteligente ” para el comportamiento de otros seres biológicos; tercero, hoy en día la palabra ”inteligencia” a menudo se utiliza para clasificar el comportamiento de los seres no biológicos tales como, máquinas, ambientes, etc. Todas ellas entidades que pueden ser representadas como sistemas computacionales; cuarto, los sistemas computacionales son sistemas formales computacionales y en consecuencia pueden ser descritos con formalismo matemático; quinto, hay una gran cantidad de sistemas formales computacionales conocidos y hay teorías matemáticas para trabajar con ellos y establecer relaciones entre ellos; sexto, el problema de definir y medir la inteligencia en máquinas formales es un problema en el contexto de estas teorías, de la inteligencia artificial, de la teoría de la computación, de la teoría de la complejidad y de la teoría de las categorías; y séptimo y por último, en el ser humano hay una medida, un cociente, llamado cociente de inteligencia para medir la inteligencia, en ingles IQ (Intelligent Quotient).

Reuní todas estas consideraciones y he creado un nuevo formalismo, un nuevo sistema computacional, que llamé Máquinas Formales (FMs). Lo que se pretendía es que todos los sistemas formales computacionales se puedan reescribir en el nuevo formalismo y que en ese proceso no pierden su estructura matemática. Por lo tanto y para satisfacer ese requisito fue necesario estudiar en profundidad una gran cantidad de sistemas formales y recurrir a la teoría de las categorías. Definido el nuevo formalismo yo he seleccionado los comportamientos en los seres humanos que se consideran comportamientos inteligentes y los he utilizado para hacer analogías con Máquinas formales y partiendo de ellos he definido y medido los mismos conceptos en Máquinas Formales y en consecuencia en los sistemas formales computacionales. Estas analogías han servido para definir y medir la ”inteligencia” en el nuevo formalismo. Así he creado un cociente para medir inteligencia en máquinas, el Cociente de Inteligencia de la Máquina, MIQ (en inglés Machine Intelligent Quotient). El MIQ es una analogía al coeficiente de inteligencia en los seres humanos, en inglés IQ (Intelligent Quotient).

Para transformar los sistemas formales computacionales para el nuevo formalismo he creado la noción de drive. Yo he diseñado algoritmos que permiten transformar cada sistema computacional formal para el nuevo formalismo. Yo he proyectado algoritmos para Máquinas de Turing, Automata de Pila, Autómatas finitos, Redes Neuronales etc. La implementación de estés algoritmos llamo drive. Yo he implementado drives para Máquinas de Turing, Automata Finitos y (en inglés) Back Propagation Neural Networks.

Yo también he construido un software para simular sistemas computacionales formales. Este software es llamado, en inglés, Generator of Universes and Simulator of Formal Machines (GU_SFM). En el momento, en lo software, sólo es posible simular el comportamiento de las máquinas de Turing, Push-down Autómatas y Autómatas finitos.

Para utilizar el nuevo formalismo en computación escribí tres APIs que serán usadas por los desarrolladores, una para el desarrollo en ordenadores desktop, otra para desarrollo con microcontroladores y otra para desarrollo en la google cloud. He también implementado dos juegos, el juego tres en raya y el cuatro en línea. En cada uno de los juegos uno de los jugadores es una implementación del nuevo formalismo y el otro es un ser humano. En ambas implementaciones ha sido posible verificar que el nuevo formalismo es un buen sistema formal computacional para resolver problemas de ingeniería. También desarrollé una placa electrónica y un sistema de información que es capaz de oler entornos. En este sistema de información se puede encontrar una implementación del nuevo formalismo en el google cloud.

Para validar las medidas MIQ hice un estudio estadístico en que he usado 1000 redes neuronales "con back propagation" y una drive que he proyectado para ellos.

Acknowledgements

This work was only possible due to a considerable personal effort and the help of a group of people. I thank them citing them here.

I thank to professor Corchado and to the professor Omatu for all the support they gave me throughout this work.

I thank to the professor Adérito Alcaso and to the professor Carlos Carreto, of the Polytechnic Institute of Guarda, by their monitoring in my PhD doctoral instance in School of Management and Technology of the Polytechnic Institute of Guarda.

I thank to professor Noel Lopes of the Polytechnic Institute of Guarda, by their help in the area of Machine Learning.

I thank to the director of the School of Management and Technology of the Polytechnic Institute of Guarda for allowing me to realize my doctoral instance the School of Management and Technology.

I thank to BISITE team, for their support in the phase of the delivery of this work.

I thank everyone in general who directly or indirectly helped me in this work.

Contents

List of Tables

List of Figures

Intelligence in Formal Machines

Paulo Alexandre Andrade Vieira,
vieirapaulo927@gmail.com

1 Introduction. A problem and its solution

This section is divided in 5 subsections; assumptions and objectives, motivation, methodology and work plan, structure of the PhD thesis and a problem and its solution. In assumptions and objectives is indicated the aim of the this work and its assumptions. In motivations is described what motivated the work, in methodology and work plan is described the methodology used. In the subsection structure of the PhD thesis is described how the thesis is structured and by last the subsection a problem and its solution is described what is the problem that are motivated this work and how it was find a solution.

1.1 Assumptions and Objectives

The aim of this work is to define and measure intelligence in computational systems. I considered the solution in the fields of medicine, artificial intelligence, computation theory, complexity theory and category theory. This work presents a solution in these areas.

1.2 Motivation

When I was a student of mathematics at the University of Oporto I studied two disciplines that would mark my future. One of them in third year, Algebraic Theory of Automata, and the other at fourth, Category Theory. The algebraic theory of automata is a mix of Algebra, Computation Theory and Complexity Theory. The Category Theory is, a meta-mathematical theory, about mathematical structures.

In my master degree in the University of Lisbon, I studied a course called Automata Theory, in which among other things was established the connection between the Automata Theory and the Category Theory.

In 2004 I participated in a robotics course at the University of Coimbra and I heard talk about fuzzy mathematics, fuzzy logic and fuzzy control

When I began my doctoral studies in University of Salamanca, in one of the courses I studied a Zadeh's article, about the fuzzy area, in which he wrote about a concept of intelligence to devices, the MIQ (Machine Intelligent Quotient).

When I began my doctoral studies at the University of Salamanca, in one of the courses I studied an article by a Zadeh, on fuzzy area, in which he wrote about a concept of intelligence to devices, the MIQ (Machine Intelligent Quotient).

This work is the consequence of this route. In this route is all my motivation and interest. Gradually grew in me the concept of the problem that existed and what should be done to find a solution.

1.3 Methodology and work plan

In General, the methodology used consisted of intense studies, on computational systems and on the notion of intelligence in humans, and in technology implementation for testing the theoretical solutions created. The intense study of computational systems led me to create a new computational system, that I called Formal Machine (FM). To show that the new formalism was implementable, it was implemented as a player in two games and as manager of alerts in the cloud. The definition and the way to measure intelligence in FMs was built from the intense study that was done of the concepts that, in a human, are considered as intelligence. In order to test some of these measures created to FMs, a statistical study of measures known for machine learning was done, and relations were established between the measures of the two computational systems.

1.4 Structure of the PhD Thesis

The thesis is composed of nine sections and one appendix. The section 1 contains references to the goals and motivations of the work. The problem to solve is identified and is explained how it was resolved. In section 2 is described a short history of computation systems and are given the definitions of the mathematical concepts that will be used in this work. In the section 3 is defined the new formalism, their computation model and is described how it should be designed a problem in the new formalism. The section 4 is a section where is built the data structure of the new formalism, is presented a software that I developed to simulate computational systems, and are described two games in which the players are FMs. In section 5 are presented several theorems about how is possible to build new FMs from others FMs. In section 6 are created several intelligence measures in FMs, the MIQs. In section 7 is presented a work that relates measures of the Machine Learning with measures of the FMs. In section 8 is analyzed the why MCUs can be seen as FMs. In section 9 is presented the conclusion of the work and is described the work to be made in the future. In the appendix is presented the report about my work, in the doctoral instance, elapsed in Guarda Portugal.

1.5 A problem and its solution

In this sub section I present the problem solved and how the solution was found and built.

A Problem: The word intelligence is nowadays used for classify a lot of behaviors. Many of these behaviors are or can be carried out by Formal Computational Systems (FCS)[1]. The problem that was identified and solved in this work was how to define, measure, quantify and compare intelligence in FCSs.

[1]Turing Machines, Pushdown Automata, Transducers, Finite Automata, k-Unbounded Register Machines and Neural Recurrent Networks and so on.

The solution: To solve this problem, I created a new formalism, called the Formal Machine (FM), which makes it possible to rewrite numerous FCSs, in Formal Machines (FMs), without losing their structure. I developed a mathematical notion about what it means to preserve the structure of an FCS. This notion was defined in the Category Theory and makes use of the functor concept, wherein the FCSs and the FMs are written as categories. There should be an embedded functor between these categories, from the FCS category to the FM category. The embedded functor is a concept defined by me and makes it possible to embed one category in another category; this embedding is made by preserving the structure of the FCS. Hence the FCSs are, in fact, FMs with the FCS structure preserved. I present how to construct the Automata, Pushdown Automata, Turing Machines and FMs as categories. As an example of an embedded functor I present an embed functor between a generic Finite Automata defined as a category to a FM also defined as a category. In fact, I present how to embed any Finite Automaton in a FM. I use the word drives to designate the implementation of the algorithms that make it possible to obtain an FM from an FCS.

Without using the Category Theory, I also algebraically rewrote different types of FCSs such as Automata, URM Machines, Pushdown Automata, Turing Machines, Neural Networks, in FMs. This serves to prove that these FCSs are FMs. For each one of these proofs, although they have not been done, is easy to construct the respective embed functor between each FCS and FM. Hence, the FMs obtained from the FCSs preserve the original structure of the FCSs. The proofs of these theorems are algebraic and constructive, they can be used to construct drives for FMs. The drives make it possible to translate an FCS into an FM and to store the new FM in a Database of FMs. For that, to build drives is necessary to know the DataStructure of the FM (page 30). I, also, defined the Data Structure (DS) of a FM[2] which, in turn, maked it possible to build: FM Databases (DBs), FCSs[3] drives, and an interface between the world and the FMs. This interface is a class (in Software Engineering) that implements the DS of a FM. Thus, given an engineering problem to solve, this interface can be used to translate the problem for a FM problem and then it can be solved through FM Technology. Thus, the drives allow that everything which is defined in FMs

[2]The class that implements the DS of a Finite Automata as a FM can be downloaded from http://www.ipg.pt/user/~pavieira/private/SW/index.html (Download the class DS_FA_FM.jar) and the documentation of it can be download from http://www.ipg.pt/user/~pavieira/private/SW/javadoc_ds_fa_fm/index.html

[3]The drive of a Finite Automata for FM, denoted by FA_FM, http://www.ipg.pt/user/~pavieira/private/SW/index.html(Download class FA_FM).
The documentation of the SW structure of the drive FA_FM, public methods and constructors, can be consulted in http://www.ipg.pt/user/~pavieira/private/SW/javadoc_fa_fm/index.html.
An explanation about how to use the drive FA_FM can be found in http://www.ipg.pt/user/~pavieira/private/SW/Use_class_FA_FM.pdf
The DB of a FM that was written in a FM through of the use of the drive FA_FM and stored in a DB can be read in http://www.ipg.pt/user/~pavieira/private/SW/Database_FM_SQL.pdf
An example of transformation of a FA for a FM, through of the use of the drive FA_FM, and its storage in a DB can be found in http://www.ipg.pt/user/~pavieira/private/SW/exampleClassFA_FM.pdf

to be naturally applied in all FCSs. I built a drive for Finite Automata (page 30). Thus any Finite Automaton can be translated and stored as a FM.

From an intense study about the concepts that in the human beings are considered concepts associated with intelligence and from the nature of the FMs I defined several measures of intelligence for FMs. Because of that, and given what a drive allows to transform a FCS in a FM, these measures are therefore naturally applied for all FCSs. I also built a software that simulates the behavior of several FCSs while they performing the tasks. The implementation of the measures referred is ongoing in the mentioned software.

As already mentioned, the way found to define intelligence in FMs was to define it through of many concepts associated with the idea of intelligence. These concepts were selected according to the strong association they have with the analogous idea of intelligence in human beings. To define Intelligence in this way, I made an anthropomorphic analysis of the FMs and built two systems of units: the (FUAP) Fundamental Constitution Units of a Machine and the (FBU) Fundamental Behavior Units. The FUAP is associated with the properties that the machine has with regard to their "anatomy" and "physiology". The FBU is a system of units related to the behavior of the machine. Using this system of units, I constructed two measures to define the capacity of the machine to the computation. One of the measures is their potential computational capacity, and the other is their effective computational capacity. The first is associated with the "anatomy" and "physiology" of the machine and the second is associated with their behavior.

The anatomy of a machine is related with its architecture, its constituents, with answers to the following questions: How is the machine? and, Which are the constituents of the machine?

The physiology of the machines is related with questions as the following, How to work, with each other, the components of a formal machines? The behavior of the machine is related with the tasks that are carried out by the machine. The first analysis made using these types of measurements is an analysis based on Occam's razor, whereby the capacity of the machine is linked with the presentation of easy solutions to simple problems, in machine language is linked to inputs associated with outputs. The outputs are words with short length([Lothaire, 2002]). As result of this was necessary to do a second type of analysis, an analysis to know the capacity of the machine to solve complex tasks. This second analysis of the machine's capacity uses techniques of Complexity Theory such as the small "o", the big "O" and Ω.

After, I defined concepts that are related to the idea of intelligence. These concepts are measured in Formal Machines and are concepts about; their Adaptation to the environment, their Creativity, the Depth and Level of Knowledge that it possesses, the capacity that it presents to communicate using a Language, its capacity to Learn, the capacity which the machine has to store information in their Memory, their Motivation, their capacity to Perceive the Environment, and their capacity to Reason. For each one I called the Machine Intelligence Quotient (MIQ); of the Adaption, MIQ_A, the MIQ_C to measure the creativity of the machine, the MIQ_L to measure its capacity to learn and so on. Thus, to measure the Intelligence of a FM is measuring the different concepts that are associated with intelligence. In Human Beings is used the Quotient of Intelligence (QI) as a measure of intelligence, in FMs there is the Machine Intelligence Quotient (MIQ). However, in machines the MIQ is a dedicated measure for each one of the ideas associated with intelligence.

In this work I defined what the computational structure of a FM is, and I implemented it in software. I called this software the Computational Structure of a FM (CSFM), it is on current version 0.03. CSFM is a software composed of interfaces, abstract classes, classes, methods and variables that should be overwritten and then they are used as an implementation of a FM. The CSFM is to be used by developers who want to solve certain problem(s) using FMs. The developers can find the CSFM in software (SW) and hardware (HW), I implemented both. To the hardware implementation, of the CSFM, I built a library for to be used in the Arduino Technology[4] and in consequence in the Micro controllers (MCUs) implemented in Arduino platforms. This implementation allows that a CSFM can be embedded in the chip, by upload, through of the overwritten of the CSFM's classes and methods. This transforms the chip into an FM that will be able to solve a set of problems. As I can have chips with CSFMs, I am now working on implementing one of them as a player of the Four-In-Line game (FIL game). I already have, in a CSFM, implemented this game as a SW desktop implementation. I have two desktop implementation of two games: the FIL game and the Tic-Tac-Toe game, both through an IDE[5]. Thus, after to constructed the CSFM for SW and HW I am working on writing applications that use the CSFM in HW and in SW, to serve as examples for developers.

In addition I demonstrated several mathematical theorems to construct FMs from FMs, through of the use of operators[6]. All these theorems are constructive theorems because one of the principal aims of them is allow their implementation. Thus, it will be possible to build FMs automatically. I also demonstrated that FMs with the adequate operators are known mathematical structures.

To validate the FM measures I developed an work with Machine Learning (MLs), more concretely Back Propagation Neural Networks (BPNN). I compared some of FMs measures with the measures that are usually used in MLs and using a statistical and inductive reasoning I conjectured several results. From a sample of 1000 BPNNs and using a drive[7] I conjectured some relations between MLs measures and FMs measures. The work to transform the conjecture on a mathematical theorem was left for future work.

As part of my PhD work and with the aim of to obtain the PhD with international mention I was in a PhD instance in Portugal during more than three months in the School of Management and Technology of the Polytechnic In-

[4]`www.arduino.cc`

[5]I use the NetBeans and Java Language

[6]$\cup$, $\cap$, $\_$, $*$ and so on

[7]The drive transforms a BPNN in a FM

stitute of Guarda. In the PhD instance work I built an electronic board, that I called an e-Nose, and the information system that allows to collected smells, store the data values in a database in the cloud and to treat the information obtained. From the information treated the information system is able to send alerts. The system sends two different types of alerts, sensor alerts and diagnostic alerts. The sensor alerts are alerts related with the readings given by the sensors. This alerts are sent when certain reading values, given by the sensors, are exceeded. The diagnostic alerts are alerts about the state of the sensors and about the state of the electronic board. They are sent if the values of the data collected can be indicating that the sensors or someone of them are no calibrated or are damaged or that the board can be damaged. The system of alerts is managed by a FM.

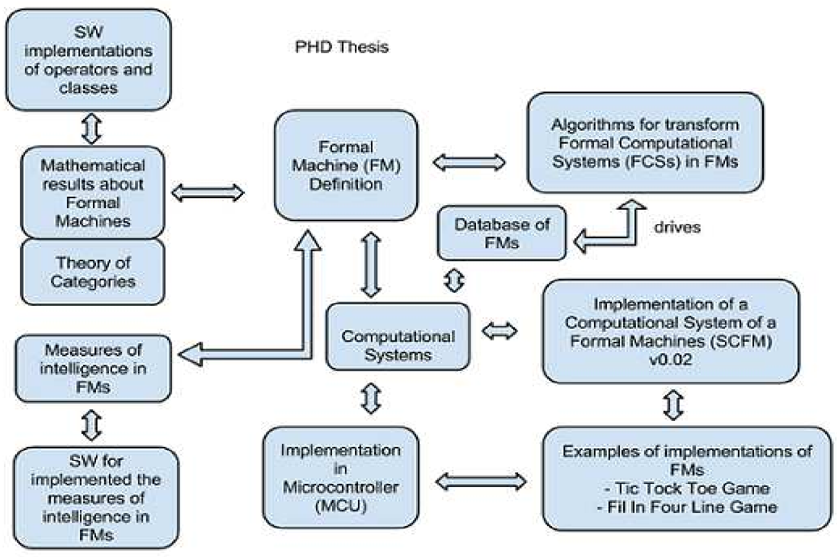

PHD Thesis
SW implementations of operators and classes
Mathematical results about Formal Machines
Theory of Categories
Measures of intelligence in FMs
SW for implemented the measures of intelligence in FMs
Formal Machine (FM) Definition
Algorithms for transform Formal Computational Systems (FCSs) in FMs
Database of FMs
drives
Computational Systems
Implementation of a Computational System of a Formal Machines (SCFM) v0.02
Implementation in Microcontroller (MCU)
Examples of implementations of FMs
- Tic Tock Toe Game
- Fil In Four Line Game

Figure 1: PhD Thesis schematic

2 Preliminary Mathematical Concepts; Introduction

This section consists of two sections. One of them presents a short story about the formal computational systems and the other presents mathematical concepts used in the work.

2.1 Short History about Formal Computational Systems

Now, I will describe briefly the history of formal computational systems

In 1900/02 ([HiD02]) Hilbert published a list of 23 unsolved problems which were at that moment the most important mathematical problems. I would like to focus on two of the 23 problems, the 2^{nd} and the 10^{th}. There is no consensus about whether the 2^{nd} is solved or not; however, the 10^{th} problem is regarded as being solved. Despite the different point of views, the attempts to solve both problems showed that they were seminal problems. In fact both led to the development of a considerable quantity of new concepts, new ideas and new mathematical results associated with the idea of finite sequential methods. Essentially, both are associated with the idea of algorithms. These finite sequential methods are FCSs and each one of the FCSs is as a classification of a certain kind of algorithms. In this work I constructed a new formalism called Formal Machine (FM) that makes it possible to transform all the FCSs into Formal Machines (FMs) without losing their structure. In the past there were many kinds of FCSs that are equivalent to each other. The difference between the FMs and this past experience is that the FM obtained from a Formal Computational System (FCS) is a FM with the structure of the FCS. The FCSs do not lose their mathematical structure. This is consequence of the use of Category Theory. The idea of working with FCSs in Category Theory was introduced by Samuel Eilenberg in 1974 [Eil74] where he rewrote some of the FCSs as categories. In this work is made an innovative use of the Category Theory in the FCSs, and is defined a new concept, an embed functor. I defined what it means to preserve the structure of a FCS, and I showed that it happens in fact, through of the construction of an embed functor between each one of the FCSs and the corresponding FMs, both seen as categories.(See section 2.2.2, page 12).

There is a school that regards the 2^{nd} problem as a problem for which it is necessary to obtain constructive proof about the consistence of Piano Arithmetic Axioms, a proof that is currently considered to be an algorithm ([CaWi07]). The 10^{th} problem, is about finding an algorithm to determine whether a given polynomial Diophantine equation with integer coefficients has an integer solution. Today this problem is solved. It is known that there is not such algorithm. ([CaWi07]) This is a consequence of Matiyasevich's theorem. The 2^{th} and the 10^{th} problems were the trigger of the known Entscheidungsproblem[8]. Thus both problems, the 2^{nd} at least for a certain school, are problems about finding algorithms. Both can be implemented or described in FCSs and both are the trigger for the emergence and development of many FCSs.

[8]The Entscheidungsproblem is a logical problem that consist in to know if it exists a generic algorithm to determine whether any first order logic sentence can be demonstrated.

In 1931 ([RaPa14]), in the attempt to solve the 2^{th} Hilbert problem, Godel demonstrated the two incompleteness theorems and showed, in the first theorem, that in a system that contains the Piano Arithmetic there are prepositions involving natural numbers that are undecidable. With the second theorem he also showed that it is impossible within this system to demonstrate their consistency. In the proof of the incompleteness theorems he defined a new class of functions, the Primitive Recursive Functions. All values of a function that belong to this class of functions are obtained using a finite recursive mechanical method. The functions are defined by recursion, but it was observed that there are functions obtained by this method that are not in the class defined. Thus, in order to solve this difficulty, in the spring of the 1934, during his visit to the Institute for Advanced Study in Princeton, Godel proposed a class of functions that he called the General Recursive Functions.

([ChAl36a]), ([ChAl36b]) In 1936 Alonzo Church presented notes about the Entscheidungsproblem. His reflections about it allowed him to create a type of computation, Turing called (([TAM36]) effective computability, the $\lambda-$calculus ([ChAl85]).

in 1936, in only one paper ([TAM36]), Turing gave a new notion of computation, called computability, presented a new model of computation, referred today as Turing Machines (TMs), and showed the equivalence between computability and the effective computation. He built too a Turing Machine (TM) that is able to compute all the TMs called, referred today, as the Universal Turing Machine (UTM). From the work of Church and Turing arose the Thesis of Church Turing[9]. This thesis is a not demonstrated result but it is accepted by the scientific community. This kind of things are, in mathematics, called principles. Thus, the this result is also called the Principle of Church Turing. A lot of people that work in Computer Science consider the TMs and UMTs to be the theoretical concept of programs (computer programs) and the computers (the machines that performed the programs) respectively. In 1944 the construction of the first computer, the ENIAC, began; it was concluded in 1946. In the ENIAC conception the emphasis was placed on solving the mechanical problems and not in the conception logic and theoretical. Thus, during the construction of the ENIAC, it became clear that a new computer, the EDVAC, should be constructed. In 1945 the mathematician Von Neumann published his famous document "First Draft of a Report on the EDVAC" ([Neu45]) where he describes this new computer.

The conception of the EDVAC is similar of the today's computers. It was a binary computer with a storage zone (the memory) where the data and the instructions are stored. Nowadays the Microcontrollers (MCUs) are divided into MCUs with Von Neumann Architecture (the architecture similar to the EDVAC) and MCUs with Harvard Architecture[10].

[9]In slight language means that all what is performed by an artifact (a device) can be computed by a Turing Machine

[10]In the Von Neumann Architecture the Data Memory and the Program Memory, such as the Data Bus and the Program Bus, are the same physical component in the MCU, and in the Harvard Architecture they are distinct physical components

The three methods, the recursive method (created by Godel), the λ-calculus (created by Church) and the Turing Machines (created by Turing) makes it possible to define the same class of functions. Thus arose a new science, the Computation Theory, and the formal concept of grammar and with them many computational models and grammars. In 1956 Chomsky ([ChNo56]) described three models to characterize a formal language. In 1959 ([ChNo59]) Chomsky presented a classification of formal languages, today known as Chomsky Hierarchy, which organizes formal languages and formal grammars in a hierarchy.

The description mentioned until this point is the description that originated the appearance of the FCSs as consequence of the Hilbert's problems. The FCSs and the hilbert's problems, created synergies among themselves and made people associate the idea of FCS with the idea of computing and the idea of computing with the idea of a machine running. Thus, the FCSs that are finite sequential methods are also finite sequential mechanical methods. From the beginning they have been associated with the idea of formalizing; mechanisms, procedures or tasks done by machines, or with the idea of formalizing the machines themselves. During the development of the computation arose several new computational methods. For example the methods of parallel computing, which represent the formalization of a computational process, here also associated with the idea of FCSs.

The idea of reproducing neural systems as a computational systems came from the field of biological, where people are studying neural systems. A lot of this reproduction was done by construction of computational methods and the FCSs appear as a computational system to simulate neural systems. Thus, the Artificial Neural Networks (ANNs) were created inspired on the central nervous systems of biological beings, principally on the brains. Going further back to describe the relationship between ANNs and computation, I would say that in 1943 Warren McCulloch and Walter Pitts ([WMWP98]) created a computational model for neural networks as an analogy of the functioning of the brain. In the late 1940s Donald Hebb ([HeDO49]) created the hypothesis of learning based on the mechanism of this model, today called Hebbian learning. In the early 1950s Belmont Farley and Wesley Clark ([KPPK]) developed the first digital computer based on artificial neural network. In 1958, Frank Rosenblatt ([RoF58]) created the perceptron, which is an algorithm for patterns recognition based on a two-layer learning computer network. Once created the perceptron there was the attempt to put it to process circuits. It was possible to process the circuits that correspond to simple addition and simple subtractions. However, It was found a simple circuit, the or- exclusive, that the perceptron was not able to process. This situation was a problem that originated some disbelief in the capabilities of this computational system.

Next, the research in Neural Networks stagnated for a time after the Marvin Minsky and Seymour Papert publication ([MiPa69]), in 1969 about machine learning. They discovered two important issues about Neural Networks: one was that single layer networks were not able to process an or-exclusive circuit; and the other was that, at the moment, computers were no able to process neural networks. This last issue caused a setback of the research in neural networks. It was only after a significant increased of the computational power of the computers that the research in neuronal networks was meaningfully increased. In 1974 Paul Werbos ([Wer74]) ([Wer94]) created the backpropagation and this algorithm allowed computing the exclusive-or circuit.

In 1986 ([RuMc86a]), ([RuMc86b]) David E. Rumelhart and James McClelland wrote a text where they described how to use parallel distributed processing to simulate neural processes, a method called connectionism. In the 1990s the statistical learning theory ([Vap95a]), ([Vap95b]) increased in importance and as consequence neural networks were overtaken in popularity. All of this led to a rise in support vector machines ([BSS99]), linear classifiers ([DHS01]) and other similar models. In the 2000s a renewed interest in neural nets arose with the advent of deep learning([YuB09]). Thus, I conclud the brief history of the FCSs.

There are two ways of doing things in computation: through the use of a computational model or ad-hoc. The use of computational models in solving engineering problems has many advantages since many mathematical mechanisms are available for use in the solutions. That is important at the moment that the problem is to be solved, because a many things are known about the formalism chosen and it is possible to use them. It is also important after the problem have been solved, because other researchers can easily add new functionalities/improvements to the solution found once the computational model is known by the scientific community and by the developers.

2.2 Mathematical Concepts

2.2.1 Mathematical Concepts about Words

In this sub subsection some concepts and notations that I will use throughout the PhD thesis are defined, including the *alphabet* as a finite and non-empty set. To denote alphabets it is, preferable to use A, Γ, O. The elements of an alphabet are called *letters*. When a, γ, o, are used alone or with indexes, it is understood that they are letters, respectively, of the alphabets A, Γ, O. A finite, and not empty, sequence of letters of an alphabet is called a *word*. The set of all words of an alphabet, A, is denoted by A^+.

Operations in words

Then [Lot97], I define some operations in words: the concatenation, the operation $+$, $*$ and ω (omega). For $u = a_0a_1...a_n$ and $v = b_0b_1...b_m$, where $a_i, b_j \in A$ with $0 \leq i \leq n$ and $0 \leq j \leq m$,

$$\text{concatenation}, \cdot, u \cdot v = a_0a_1...a_nb_0b_1...b_m, \text{ abbreviated as } uv$$

The concatenation is an associative operation. From the concatenation it is possible to define a new word ϵ, called *empty word*, ϵ, as follows

$$\forall x \in A^+, x\epsilon = \epsilon x = x.$$

Then the operations $+$, $*$ and ω (omega) are defined as

$$\textit{iteration operator}, +, u^+ = \{u^n : n \in \mathbb{N}\} \text{ with } u^n = u \cdot ... \cdot u = \overbrace{u...u}^{\text{n times}}$$
$$\textit{star operator}, *, u^* = \{\epsilon\} \cup \{u^n : n \in \mathbb{N}\}$$
$$\textit{omega operator}, \omega, u^\omega = uuuu...uuu.... \text{ (u is repeated indefinitely)}$$

I also denoted $u(i) = a_i$ and a function $|\,| : A^* \to \mathcal{P}(\mathbb{N})$, wherein $m \in |\,|(w)$ if w is decomposed in m letters of A. By abuse of notation, $|w|$ replaced the notation $|\,|(w)$. Thus, for example, $|\epsilon| = 0$. I define $m \in |w|_B$, with $B \subseteq A$, if w is decomposed in m letters of B(See Example 2.1).

Examples 2.1 *$A = \{0, 10, 100\}$ i) $|100| = \{1, 2\}$, 100 is decomposed in 10, 0 and in 100.*
ii) $|100|_{\{0,1,10\}} = \{2, 3\}$, 100 is decomposed by 0 and 10 in $\{0, 1, 10\}$ and by two 0's and 1 in $\{0, 1, 10\}$. $|100|_{\{10,100\}} = 1$, 100 is decomposed by 100 in $\{10, 100\}$. $|100|_{\{10\}} = 0$, 100 have not any decomposition in $\{10\}$.
iii) $|10|_{\{0,10\}} = 1$, the word 10 is decomposed by 10 in $\{0, 10\}$.
iv) $|100|_{\{0\}} = 0$, the word 100 have not any decomposition $\{0\}$. By abuse of notation, instead of $|100|_{\{0\}} = 0$, is possible to write $|100|_0 = 0$.
v) $|000| = 3$, the word 000 is decomposed by three 0's.$\square$

Operations in sets

The operations above can be generalized and applied to sets. The following sets can be constructed from the alphabet A.

$$\textit{the set } A^+, A^+ = \{a_0a_1...a_n : n \in \mathbb{N}_0, a_i \in A, 0 \le i \le n\},$$
$$\textit{the set } A^*, A^* = A^+ \cup \{\epsilon\},$$
$$\textit{the omega set, of } A, A^w, A^w = \{a^w : a \in A\}.$$

The *concatenation of two sets* of words $L_1 \subseteq A^*$ and $L_2 \subseteq A^*$ can also be defined as

$$L_1 \cdot L_2 = \{u \cdot v : u \in L_1 \text{ e } v \in L_2\}$$

For sets, for example a set D, $|D|$ denotes the cardinal of D, that is the quantity of elements that D possesses.

Let $S_1, S_2, ..., S_N$ be sets, $S = S_1 \times S_2 \times ... \times S_N$ and a N-tuple $s = (s_1, ..., s_i, ..., s_N) \in S$. s_i is denoted (respectively, S_i) as $s(S_i)$ or $s(i)$ (respectively, $S(s_i)$ or $S(i)$). Then $s_i = s(S_i) = s(i)$ (respectively, $S_i = S(s_i) = S(i)$). (See Example 2.2)

Examples 2.2 *$S_1 = \mathbb{N}$, $S_2 = \mathbb{Z}$, $S_3 = \mathbb{Q}$, $S_4 = \{1, 2\}$, $S_5 = \mathbb{N}$. Let $S = S_1 \times S_2 \times S_3 \times S_4 \times S_5$ and $s = (s_1, s_2, s_3, s_4, s_5) = (2, -1, \frac{1}{3}, 2, 5)$. Then $s(S_2) = -1$, $s(S_3) = \frac{1}{3}$, $s(S_4) = 2$ and $S(s_1) = \mathbb{N}$, $S(s_2) = \mathbb{Z}$, $S(s_4) = \{1, 2\}$.*$\square$

The power set (respectively, finite subsets) of a set D is written as $\mathcal{P}(D) = \{E : E \subseteq D\}$ (respectively, $\mathcal{P}_f(D) = \{E \subseteq D : |E| < \infty\}$)

Codes

A subset of A^+, X, is said to be a *code* [BDR10] if, for all $x_0, x_1, ..., x_n \in X$, $y_0, y_1, ..., y_m \in X$ such that $x_0 x_1 ... x_n = y_0 y_1 ... y_m$, implies $n = m$ and ($x_i = y_i$, for all $0 \leq i \leq n$). In this case,

$$X^* = \oplus_{n \geq 0} X^n = X^0 \dot{\cup} X \dot{\cup} X^2 \dot{\cup} ... = \{\epsilon\} \dot{\cup} X \dot{\cup} X^2 \dot{\cup} ... \text{ (disjoint union)}$$

When the alphabet A is a code, $|u|$ is denoted by $|u\rfloor$ (Example 2.3). If A is a code, each $u \in A^+$ has only one decomposition in A. Therefore, $|u\rfloor$=length of the word u in A and $|u\rfloor_B$=length of the word u in B whenever $B \subseteq A$ and $u \in B$.

Examples 2.3 $A = \{0,1\}$ *is a code*, $|100\rfloor = 3$, $|100\rfloor_0 = 2$, $|100\rfloor_1 = 1$.□

2.2.2 Category Theory

A *category*, $\mathcal{C}$, is a 2-tuple $\mathcal{C} = (Obj_{\mathcal{C}}, Morf_{\mathcal{C}})$ where $Obj_{\mathcal{C}}$ is a set, called the *objects* of $\mathcal{C}$, and $Morf_{\mathcal{C}}$ is a family of sets $Morf_{\mathcal{C}}(X,Y)$ with $X, Y \in Obj_{\mathcal{C}}$, called the *morphisms* of $\mathcal{C}$. The category $\mathcal{C}$ has the following properties:
i) There is an operator of composition, $\circ_{\mathcal{C}}$, such that for all $X, Y, Z \in Obj_{\mathcal{C}}$,

$$\circ_{\mathcal{C}} : Morf_{\mathcal{C}}(X,Y) \times Morf_{\mathcal{C}}(Y,Z) \longrightarrow Morf_{\mathcal{C}}(X,Z) \text{ is a function,}$$

ii) $\circ_{\mathcal{C}}$ is associative, for all $f \in Morf_{\mathcal{C}}(X,Y)$, $g \in Morf_{\mathcal{C}}(Y,Z)$, $h \in Morf_{\mathcal{C}}(X,Z)$, $(f \circ_{\mathcal{C}} g) \circ_{\mathcal{C}} h = f \circ_{\mathcal{C}} (g \circ_{\mathcal{C}} h)$, and
iii) If $X = Y$. Then exists a $1_X \in Morf_{\mathcal{C}}(X,X)$ such that for all $f \in Morf_{\mathcal{C}}(X,Z)$ and $g \in Morf_{\mathcal{C}}(Z,X)$, $1_X \circ_{\mathcal{C}} f = f$ and $g \circ_{\mathcal{C}} 1_X = g$

When there's not ambiguity the *composition operator* is only denoted by $\circ$.
A category $\mathcal{D}$ is called a *subcategory* of $\mathcal{C}$ if:
i) $Obj_{\mathcal{D}} \subseteq Obj_{\mathcal{C}}$,
ii) for all $X \in Obj_{\mathcal{D}}$, 1_X, the identity in X, in $\mathcal{D}$ is also the identity of X in $\mathcal{C}$,
iii) $Morf_{\mathcal{D}}(X,Y) \subseteq Morf_{\mathcal{C}}(X,Y)$ for all $X, Y \in Obj_{\mathcal{D}}$, and
iv) $\circ_{\mathcal{C}}|_{Morf_{\mathcal{D}}(X,Y) \times Morf_{\mathcal{D}}(Y,Z)} = \circ_{\mathcal{D}}|_{Morf_{\mathcal{D}}(X,Y) \times Morf_{\mathcal{D}}(Y,Z)}$ for all $X, Y, Z \in Obj_{\mathcal{D}}$.

Next, another mathematical object, called a functor can be defined. A *functor* is a correspondence between two categories, $\mathcal{C}$ for $\mathcal{D}$, that preserves the composition operator, $\circ_{\mathcal{C}}$, of the category $\mathcal{C}$ in the category $\mathcal{D}$ through of the use of the composition operator $\circ_{\mathcal{D}}$. Formally, a functor $\mathcal{F}$ is a 2-tuple correspondence $\mathcal{F} = (\mathcal{F}_{Obj}, \mathcal{F}_{Morf})$ such that
i) $\mathcal{F}_{Obj} : Obj_{\mathcal{C}} \longrightarrow Obj_{\mathcal{D}}$ is such that for all $X \in Obj_{\mathcal{C}}$,

$$\mathcal{F}_{Obj} : X \longrightarrow \mathcal{F}_{Obj}(X) \text{ is a function}$$

ii) $\mathcal{F}_{Morf} : Morf_{\mathcal{C}} \longrightarrow Morf_{\mathcal{D}}$ is such that for all $X, Y \in Obj_{\mathcal{C}}$,

$$\mathcal{F}_{Morf}(X,Y) : Morf_{\mathcal{C}}(X,Y) \longrightarrow Morf_{\mathcal{D}}(\mathcal{F}_{Obj}(X), \mathcal{F}_{Obj}(Y)) \text{ is a function wherein:}$$

ii.1) for all $X \in Obj_{\mathcal{C}}$, $\mathcal{F}_{Morf}(X,X)(1_X) = 1_{\mathcal{F}_{Obj}(X)}$
ii.2) for all $f \in Morf_{\mathcal{C}}(X,Y)$, $g \in Morf_{\mathcal{C}}(Y,Z)$;

$$\mathcal{F}_{Morf}(X,Z)(f \circ_{\mathcal{C}} g) = \mathcal{F}_{Morf}(f)(X,Y) \circ_{\mathcal{D}} \mathcal{F}_{Morf}(g)(Y,Z).$$

The *functor* $\mathcal{F}$ is called *injective* if the functions $\mathcal{F}_{Morf}(X,Y)$ for all $X,Y \in Obj_{\mathcal{C}}$ are injectives. This case also indicates which $\mathcal{F}_{Morf}$ is injective. An injective functor is called a *faithful functor*. I refer to the functor $\mathcal{F}$ as *embedded functor* if it is a faithful functor where $\mathcal{F}_{Obj} : X \longrightarrow \mathcal{F}_{Obj}(X)$ for all $X \in Obj_{\mathcal{C}}$ is injective. If $\mathcal{F}$ is an embedded functor then category $\mathcal{C}$ is said that is *embedded* in category $\mathcal{D}$, and $\mathcal{C}$ is isomorphic to a subcategory of $\mathcal{D}$, $\mathcal{F}(\mathcal{C})$, which is the same category as $\mathcal{C}$. In slight language, the relations that the morphisms of $\mathcal{C}$ have among themselves are the same relations that, with respect to the correspondence of the objects and morphisms, exist among the morphisms of the $\mathcal{F}(\mathcal{C})$. When there is no ambiguity, I use $\mathcal{F}_{Morf}$ instead of $\mathcal{F}_{Morf}(X,Y)$.

The formal definition of FM, presented in the following section, was, also, built from a careful analysis of several FCSs [Hop08][Cut97] and based on the knowledge about how microcontrollers [Atm13] and the processors operate. Some of the analysis done can be observed in the proofs of the two theorems, theorems 4.1 and 4.2 on page 42, which establishes that set out that a number of FCSs are FMs.

3 Formal Machines

In this section I present the definition of a new formalism, the Formal Machine (FM). I present a finite automaton, that recognizes gmail addresses, and a Pushdown Automaton, that verify the corrections of the parenthesis in a syntax of a programming language. For the two automata are showed their translation to FMs. Thus, these two FMs are the first two FMs, that I present here, that are able to solve concrete problems. In this sections I also presented the computational model of a FM, their implementation in serial and in parallel. For last I present how should be conceived a problem in FMs.

3.1 Definition of a Formal Machine

The process of construction of this formalism in which the FCSs are instantiated without modifying their structures or their nature is resulted of considerable reflection and analysis. In this process I analyzed a number of mathematical entities that are considered FCSs. This new formalism will allow to define measures, mathematical entities, and to obtain results about these measures and entities that can be directly applied to FCSs. From the analysis and reflection referred, arose the conviction that the characteristics of a FM, should be as follows.

3.1.1 Generic approach to Formal Machines

The definition of a Formal Machine should include:

i) a declaration on its components, which should be finite in number, and the causal relations or connections among them;

ii) the meaning of the configuration of the machine. The concept must make it possible to know, at any moment, the configuration of the machine. All the components of the machine should be in the configuration and knowing the configuration should tell you the state of each one of the components;

iii) the primitive instructions of the machine. Each instruction is a k-partial operation in the set of the machine configurations. The machine instructions operate in the space of these configurations using the connection that exists among the components. A program is a finite sequence of instructions.

iv) an algorithm that describes how the machine works. That algorithm is called the *Von Neumann's Algorithm*, VN_{Alg}.

3.1.2 Defining a Formal Machine

Next I am going to define a FM. Examples of FMs can be seen in Example 3.1 page 17. A FM, fM, is a 7-tuple

$$\mathrm{fM} = (CompM_B, CompM_R, ConfM, ConfM_i, ConfM_f, InstM, \mathrm{VN}_{Alg}),$$

wherein:

i)
- the set $CompM_B$ is called the set of the *basic machine components*. $CompM_B$ is a finite set,

$$CompM_B = \{C_1, ..., C_n\}.$$

The set $CompM_B$ has the following property:

$$\forall i \in \{1, 2, ..., n\} \exists A_i \neq \emptyset : |A_i| < +\infty, A_i \text{ is a code and } C_i \subseteq (A_i)^*$$

A_i is designated the *alphabet of the component* C_i and is also written as $A(C_i)$, $A(C_i) = A_i$. The existence of the alphabet of the component, without loss of generality, $A_i = A(C_i)$, allows to the machine to have a unique factorization on A_i for the operation of concatenation on A_i^+ at each component C_i.

- the set $CompM_R$, is called the set of the *inner relations of the machine* or simply the *set of the relations of the machine,*

$$CompM_R = \{R_1, ..., R_m\}.$$

The set $CompM_R$ has the following properties:

$$(\forall R_i \in Comp_R)(\exists t_i \in \mathbb{N}) : R_i \subseteq \times_{j=1}^{t_i} C'_j$$

with $C'_j \in CompM_B$. The relations $R \in CompM_R$ can be seen as providing streaming channels between components.

ii) $ConfM \subseteq \times_{i=1}^{n} C_i$ is the set of *configurations of the machine*. A *configuration* $c \in ConfM$ is an n-tuple $c = (c_1, ..., c_n)$. Thus, $c(C_i) = c_i$. Is possible to build, naturally, a code that is the Cartesian product of codes, $A(C_1) \times ... \times A(C_n)$, whereupon can be written the configurations of the machine. In that code, $|c\rfloor = \sum_{i=1}^{n} |c_i\rfloor$, with $|c_i\rfloor =$ length of the word c_i in $A(C_i)$.

$ConfM_i \subseteq ConfM$ is the set of the *initial configurations* of the machine. The initial configurations are the configurations that allow to start the work of the machine.

$ConfM_f \subseteq ConfM$ is the set of the *final configurations* of the machine. A configuration that makes it impossible, in any circumstance, for the machine to shift its configuration is called a *stop configuration*. The final configurations are stop configurations.

iii) $InstM = \{I_1, I_2, ..., I_r\} \cup \{NOP\}$ is a finite set, the *set of primitive instructions*. Each instruction $I_j \neq NOP$ is such that,

$$I_j : \times_{l=1}^{k} (ConfM \setminus ConfM_f) \longrightarrow \mathcal{P}_f(ConfM), \text{ with } k \in \mathbb{N},$$
$$\mathcal{P}_f(ConfM) = \{D \in ConfM : |D| < \infty\}$$

and $c_P = \{c_1,..,c_t\} \subseteq ConfM$, $c_P = I(\vec{c})$ with $\vec{c} = (c^1, c^2, ..., c^k) \in \times_{l=1}^{k} ConfM$. For each $c_i = (c_{i1}, ..., c_{in}) \in c_P$, and c_{iv} with $1 \leq v \leq n$ (therefore $c_i(C_v) = c_{iv}$ and $C_v \in CompM_B$), either
a)[11] $\exists 1 \leq j \leq k$: $c_{iv} = c^j(C_v)$ or
b)[12] $(\exists 1 \leq j \leq k)$ $(\exists 1 \leq r_{c_{iv}}, s_{c_{iv}} \leq n)$ $(\exists R_{c_{iv}} \in CompM_R)$
$(\exists t_1 < ... < t_{r_{c_{iv}}}$ with $c^j_{t_1} \in C_{t_1}(c^j), ..., c^j_{t_{r_{c_{iv}}}} \in C_{t_{r_{c_{iv}}}}(c^j))$
$(\exists w_1 < ... < w_{s_{c_{iv}}}$ with $c_{iw_1} \in C_{w_1}(c_i), ..., c_{iw_{s_{c_{iv}}}} \in C_{w_{s_{c_{iv}}}}(c_i))$
$\exists w_l \in \{w_1, ..., w_{s_{c_{iv}}}\}$ such that
$c_{iv} = c_{iw_l}$, $u = (c^j_{t_1}, ..., c^j_{t_{r_{c_{iv}}}}, c_{iw_1}, ..., \underbrace{c_{iw_l}}_{c_{iv}}, ..., c_{iw_{s_{c_{iv}}}}) \in R_{c_{iv}}$, $R_{c_{iv}}(r_{c_{iv}} + l) = C_v$ (See section 10.1, page 132).

I_j is called a *k-instruction* or simply an *instruction of the machine*. The instructions of the machine operate on subsets of a finite Cartesian product of the set of configurations, $(ConfM)^k$. When there is a subset of configurations obtained from the application of an instruction and this subset has new configurations, is because the instruction uses connections among the components. Each one of those connections are elements of a relation. That relation is a subset of a finite Cartesian product of several components of the machine and is an element of the set $CompM_R$.

NOP is the following instruction:

$$\text{NOP}:\mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(\mathcal{P}(\mathbb{N})) \text{ wherein}$$
$$\text{NOP}(c_P) = \{c'_P : c'_P \subseteq c_P\}.$$

When $c'_P \in NOP(c_P)$ I use a notation more slight and I write $NOP(c_P) = c'_P$.
iv) Von Neumann's Algorithm (VN$_{Alg}$) is an algorithm with a certain structure consisting of the description of how the FM works. The VN$_{Alg}$ is divided in 3 zones; the Definition Zone, the Setting Zone and the Execution Zone. The Definition Zone is the place in the algorithm where the constants, the variables, and all kinds of objects that are used in the algorithm are defined. The meta-objects required to the running of the algorithm are also defined here. In the Setting Zone the initial state of the machine is set. That initial state can be the first perception of the environment, acquired from out, or an input introduced. The Execution Zone is also called the *Von Neumann's Cycle* (VNC). The VNC is a loop with or without a stopping condition. Each execution of the VNC is called a *cycle of machine*[13]. Let's see the overall structure of the VN$_{Alg}$.

[11] c_{iv} is the v^{th} component of some configuration $c^j \in ConfM$ which is in $\vec{c}$

[12] The component of c_i, c_{iv}, is obtained from $I \in InstM$. $c_i \in I(\vec{c})$ and there is a relation R_{iv} and a $u \in R_{iv}$ such that the first elements of u, $u_1,...,u_{r_{c_{iv}}}$ are elements of some configuration c^j of $\vec{c}$ and the other elements $u_{r_{c_{iv}}+1}, ..., u_{r_{c_{iv}}+s_{c_{iv}}}$ are elements of c_i. One of the elements $u_{r_{c_{iv}}+1}, ..., u_{r_{c_{iv}}+s_{c_{iv}}}$, at least one, suppose $u_{r_{c_{iv}}+l}$ is c_{iv} with $R_{iv}(u_{r_{c_{iv}}+l}) = C_v$.

[13] Looking at the datasheet of a MCU, an important measure for an instructions is the number of cycles of machine that are necessary to their execution.

Algorithm 1 Von Neumann's Algorithm

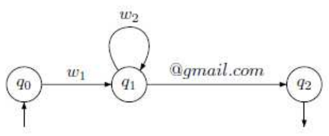
```
/*
// Definition Zone
....
*/
setup()
{
// Setting Zone
.....
}
loop(with or without a stopping condition)
{
// Execution Zone. Cycle of Von Neumann
......
}
```

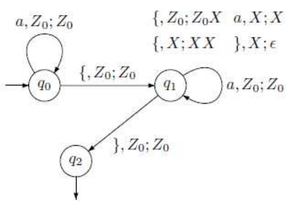
The VN_{Alg} can be viewed as a universal mechanism of how the instantiated formal system works. For example, in the case of the FM obtained from a Finite Automaton, the VN_{Alg} will be, in slight language, the universal mechanism of how any Finite Automaton works. This universality allows to classify the new formalisms from the VN_{Alg} and through of the operation *mod* VN_{Alg}[14](See section 3.1.3 page 19). Any program performed by the machine is always performed, at a low level, by the program that implements the Von Neumann's algorithm of the machine. The characterization of the type of computation that a machine makes must be observed in the VN_{Alg}.□

Examples 3.1 *i) The FM* fM_1 *(page 18), recognizes an e-mail address of the gmail. This FM is obtained from a Finite Automaton* $\mathcal{A}_1$
ii) The FM fM_2 *(page 19), recognizes the brackets well formed. This FM is obtained from a Pushdown Automaton* $\mathcal{PA}_1$ *.*□

In the following figures can be seen the Automaton $\mathcal{A}_1$, the Pushdown Automaton $\mathcal{PA}_1$ and the FMs fM_1 and fM_2. fM_1 is the FM built from $\mathcal{A}_1$ and fM_2 is the FM built from $\mathcal{PA}_1$. An algorithm to transforms a Finite Automata to FMs and Pushdown Automata to FMs can be seen in sub subsection 4.2.2(page 44) and is announced by the theorem 4.1 (page 42).

[14]The definition of the relation *mod* can be seen at the end of the section 3.1.3

Example 3.1 i) Automaton	Formal Machine, fM$_1$
$\mathcal{A}_1$  $A = A_1 \cup A_2 \cup A_3$ is a code $A_1 = \{a, A, \ldots, z, Z\}$ $A_2 = \{1, \ldots, 9\}$ $A_3 = \{@gmail.com\}$ $\delta(q_0, w_1) = q_1, w_1 \in A_1$ $\delta(q_1, w_2) = q_2, w_2 \in A_1 \cup A_2$ $\delta(q_2, @gmail.com) = q_3$	$\rightarrow Comp_B = \{Q, T_I, p_I\}$: $Q = \{q_0, q_1, q_2\}$, $T_I = A^*\flat^\omega$, $p_I = \mathbb{N}$ $\rightarrow Comp_R = \{R_\delta\}$: $(q_0, uw_1v\flat^\omega, \lvert u\rfloor + 1, q_1, \lvert u\rfloor + 2)$, $(q_1, uw_1v\flat^\omega, \lvert u\rfloor + 1, q_1, \lvert u\rfloor + 2)$ $(q_1, u\ @gmail.com\ \flat^\omega, \lvert u\rfloor + 1, q_2, \lvert u\rfloor + 11)$, $\rightarrow ConfM = Q \times A^*\flat^\omega \times \mathbb{N}$ $ConfM_i$: $(q_0, w_1\ v\flat^\omega, 1)$ $ConfM_f$: $(q_2, u\flat^\omega, \lvert u\rfloor + 1)$ $\rightarrow$ Instructions: $I(c_\bullet) = c_{\bullet\delta}$ $c_1 = (q_0, uw_1v\flat^\omega, \lvert u\rfloor + 1)$, $c_{1\delta} = (q_1, uw_1v\flat^\omega, \lvert u\rfloor + 2)$ $c_2 = (q_1, uw_2v\flat^\omega, \lvert u\rfloor + 1)$, $c_{2\delta} = (q_3, uw_2v\flat^\omega, \lvert u\rfloor + 2)$ $c_3 = (q_1, @gmail.com\flat^\omega, \lvert u\rfloor + 1)$ $c_{3\delta} = (q_2, @gmail.com\flat^\omega, \lvert u\rfloor + 11)$ with $u, v \in A^*$ with $w_1 \in A_1, w_2 \in A_1 \cup A_2$

Example 3.1 ii) Pushdown Automaton	Formal Machine, fM$_2$
$\mathcal{PA}_1 = (Q, I, F, A, \Gamma, Z_0, \delta_{\mathcal{AS}_1})$ $\mathcal{PA}_1$  $a, Z_0; Z_0$ $\{, Z_0; Z_0X$ $a, X; X$ $\{, X; XX$ $\}, X; \epsilon$ $\{, Z_0; Z_0$ q_0 q_1 $a, Z_0; Z_0$ $\}, Z_0; Z_0$ q_2 $Q = \{q_0, q_1, q_2\}$ $I = \{q_0\}$, the arrow pointing inwards $F = \{q_2\}$, the arrow points out A = ASCII and $a \in$ ASCII $- \{``\{", ``\}"\}$, A is a code $\Gamma = \{Z_0, X\}$ $\delta(q_0, a, Z_0) = (q_0, Z_0)$, $\delta(q_0, \{, Z_0) = (q_1, Z_0)$ $\delta(q_1, a, Z_0) = (q_1, Z_0)$, $\delta(q_1, a, X) = (q_1, X)$ $\delta(q_1, \{, Z_0) = (q_1, Z_0X)$ $\delta(q_1, \{, X) = (q_1, XX)$ $\delta(q_1, \}, X) = (q_1, \epsilon)$, $\delta(q_1, \}, Z_0) = (q_2, Z_0)$	$u, v \in A^*$ $\rightarrow Comp_B = \{Q, T_I, p_I\}$: $T_I = A^*\flat^\omega, p_I = \mathbb{N}, P = Z_0(\Gamma - Z_0)^*$ $\rightarrow CompM_R = \{R_\delta\}$ $(q_0, uav\flat^\omega, \lfloor u \rfloor + 1, Z_0, q_0, \lfloor u \rfloor + 2, Z_0)$ $(q_0, u\{v\flat^\omega, \lfloor u \rfloor + 1, Z_0, q_1, \lfloor u \rfloor + 2, Z_0)$ $(q_1, uav\flat^\omega, \lfloor u \rfloor + 1, Z_0, q_1, \lfloor u \rfloor + 2, Z_0)$ $(q_1, uav\flat^\omega, \lfloor u \rfloor + 1, Z_0\alpha_0X, q_1, \lfloor u \rfloor + 2, Z_0\alpha_0X)$ $(q_1, u\{v\flat^\omega, \lfloor u \rfloor + 1, Z_0, q_1, \lfloor u \rfloor + 2, Z_0X)$ $(q_1, u\{v\flat^\omega, \lfloor u \rfloor + 1, Z_0\alpha_0X, q_1, \lfloor u \rfloor + 2, Z_0\alpha_0XX)$ $(q_1, u\}v\flat^\omega, \lfloor u \rfloor + 1, Z_0\alpha_0X, q_1, \lfloor u \rfloor + 2, Z_0\alpha_0)$ $(q_1, u\}\flat^\omega, \lfloor u \rfloor + 1, Z_0, q_2, \lfloor u \rfloor + 2, Z_0)$ $\rightarrow ConfM = Q \times A^*\flat^\omega \times \mathbb{N} \times (Z_0(\Gamma - Z_0)^* \cup \{\epsilon\})$ $ConfM_i$: $(q_0, u\flat^\omega, 1, Z_0)$ $ConfM_f$: $(q_2, u\flat^\omega, \lfloor u \rfloor + 1, \alpha)$, $\alpha \in Z_0(\Gamma - Z_0)^* \cup \{\epsilon\}$ $\rightarrow$ Instructions: $I(c_\bullet) = c_{\bullet\delta}$ $c_1 = (q_0, uav\flat^\omega, \lfloor u \rfloor+1, Z_0)$; $c_{1\delta} = (q_0, uav\flat^\omega, \lfloor u \rfloor+2, Z_0)$ $c_2 = (q_0, u\{v\flat^\omega, \lfloor u \rfloor+1, Z_0)$; $c_{2\delta} = (q_1, u\{v\flat^\omega, \lfloor u \rfloor+2, Z_0)$ $c_3 = (q_1, uav\flat^\omega, \lfloor u \rfloor+1, Z_0)$; $c_{3\delta} = (q_1, uav\flat^\omega, \lfloor u \rfloor+2, Z_0)$ $c_4 = (q_1, uav\flat^\omega, \|u\| + 1, Z_0\alpha_0X)$; $c_{4\delta} = (q_1, uav\flat^\omega, \lfloor u \rfloor + 2, Z_0\alpha_0X)$ $c_5 = (q_1, u\{v\flat^\omega, \lfloor u \rfloor + 1, Z_0)$; $c_{5\delta} = (q_1, u\{v\flat^\omega, \lfloor u \rfloor + 2, Z_0X)$ $c_6 = (q_1, u\{v\flat^\omega, \lfloor u \rfloor + 1, Z_0\alpha_0X)$ $c_{6\delta} = (q_1, u\{v\flat^\omega, \lfloor u \rfloor + 2, Z_0\alpha_0XX)$ $c_7 = (q_1, u\}v\flat^\omega, \lfloor u \rfloor + 1, Z_0\alpha_0X)$; $c_{7\delta} = (q_1, u\}v\flat^\omega, \lfloor u \rfloor + 2, Z_0\alpha_0)$ $c_8 = (q_1, u\}v\flat^\omega, \lfloor u \rfloor+1, Z_0)$; $c_{8\delta} = (q_2, u\}v\flat^\omega, \lfloor u \rfloor+2, Z_0)$

One of the common features among today's different FCSs is the fact that all of them can be represented through directed graphs. A directed graph [BoM82] is an entity G that is an ordered pair of sets $G = (V, E)$, where the set V is called the *vertex set*, and the set $E = \{uv : u, v \in V\}$ is called the *edges set*. The set V can be seen as a code if necessary rewriting it. The V can be thought as the set of the components of the graph. E can be seen as a binary relation in V. These are clear reasons to justify the inclusion of the sets $CompM_B$ (set of the components of a FM) and $CompM_R$ (set of the relations of causalities or connections among the components of a FM) in the definition of FM.

3.1.3 Computation on Formal Machines

In this sub subsection I am going to define what is a *program* of a FM, the *computation operator* of a FM and a *task performed* by a FM.

A *program* of the fM[15] is a finite and ordered sequence of instructions of the

[15]Note that fM is a FM, look the definition of the FM.

machine $(\alpha_0, \alpha_1, ..., \alpha_k)$, wherein $k \in \mathbb{N}_0, \alpha_j \in InstM, 0 \leq j \leq k$. The set of all programs is denoted by *progM*, $progM = \{(\alpha_0, \alpha_1, ..., \alpha_k) : k \in \mathbb{N}_0, \alpha_j \in InstM, 0 \leq j \leq k\}$.

The binary operator $\vdash$, called *computational operator*, is defined in the set of all subsets of configurations of the fM,

$$\mathcal{P}(ConfM) = \{A : A \text{ is a set of configurations of the fM}\}.$$

An element of $\mathcal{P}(ConfM)$ is denoted by c_P[16]. For two sets of configurations $c_P, c'_P \in \mathcal{P}(ConfM)$, $c_P \vdash c'_P$ is defined as follows

$$c_P \vdash c'_P \text{ iff } (\exists c''_P \subseteq c_P)(\exists I_j \in InstM - \{NOP\}(\text{k-instruction}))(\exists \{c_1, ..., c_k\} \subseteq c_P) : \\ c'_P = c''_P \cup c'''_P, \vec{c} = (c_1, ..., c_k) \text{ e } c'''_P = I_j(\vec{c}) \text{ or (making use of NOP) } c'_P \subseteq c_P$$

and is said that the instruction I_j *transforms* c_P in c'_P. The operator $\vdash$ occurs only inside of the VNC. A sequence

$$e = c_{(0,P)} \vdash c_{(1,P)} \vdash ... \vdash c_{(n,P)}, \text{ where } c_{(i,P)} \in \mathcal{P}(ConfM) \text{ for } i = 0, 1, ..., n$$

is called an *execution* of the fM, and the sets of configurations $c_{(0,P)}$, $c_{(1,P)}$, ..., $c_{(n,P)}$ are designated the set of *configuration of e*,

$$ConfM_P(e) = \{c_{(i,P)} : 0 \leq i \leq n\}.$$

Examples 3.2 *A computation of the* fM$_1$ *(See page 18) to recognize the mail address paulo@gmail.com. The computation of the task referred is:*
$(q_0, paulo@gmail.com\flat^\omega, 1) \vdash (q_1, paulo@gmail.com\flat^\omega, 2) \vdash ...$
$\vdash (q_1, paulo@gmail.com\flat^\omega, 6) \vdash (q_6, paulo@gmail.com\flat^\omega, 16)$.□

The close transitive of the operator $\vdash$ is denoted by $\vdash^*$. For each execution exists always a program of the machine, which is the sequence of instruction that was used. However, the relation is not one-to-one, since can happen two distinct programs carried out the same execution. This is dependent of the computational structure of the FM. The computation operator, $\vdash$, makes FMs formal systems in which are difficult to do practices of Reverse Engineering. Thus, they are good systems to implement security solutions.

Examples 3.3 *The 6-tuple (I, I, I, I, I, I) is the program of the* fM$_1$ *(See page 18) for recognize paulo@gmail.com and renato@gmail.com. The same program that works and allows to recognize two distinct tasks.* .□

The pair $\tau = (c_P, c'_P)$, where $c_P \subseteq ConfM_i$ and $c'_P \subseteq ConfM_f$ such that $c_P \vdash^* c'_P$ is called a *task performed* by the fM. If $\tau = (c_P, c'_P)$ then c_P and c'_P are denoted, respectively, by τ_I and τ_O. Therefore, $\tau_I = c_P$ and $\tau_O = c'_P$. $|\tau\rfloor$ is defined

[16]When the set c_P is such that $|c_P| = 1$, c_P contains only one configuration, c, $c_P = \{c\}$. In this case is written, by abuse of notation, $c_P = c$

as $|\tau\rfloor = \sum_{c\in\tau_I} |c\rfloor + \sum_{c\in\tau_O} |c\rfloor$.

The set of the tasks performed by the fM, $\mathcal{L}$(fM), is called the *Language performed* (or *recognized*) by the fM.

$$\mathcal{L}(\text{fM}) = \{(c_P, c'_P) : c_P \subseteq ConfM_i, c'_P \subseteq ConfM_f \text{ and } c_P \vdash^* c'_P\}$$

Examples 3.4 $\mathcal{L}(\text{fM}_1)=\{(c_P, c'_P) : c_P \subseteq (q_0, u\text{b}^\omega, 1) : u \in A_1(A_1 \cup A_2)^* A_3,$
$c'_P \subseteq \{(q_2, u\text{b}^\omega, |u\rfloor + 1) : (q_0, u\text{b}^\omega, 1) \in c_P\}\}$.□

The class of all FMs is called the class $\mathbb{Z}eus$, $\mathbb{Z}eus = \{\text{fM} : \text{fM is a FM}\}$.

The VN$_{Alg}$ of each FM is part of the own FM. Let fM be a FM. fM has a certain VN$_{Alg}$, $\mathcal{P}_1$, and a language performed by it, $\mathcal{L}$(fM). Thus, fM$_{\mathcal{P}_1}$ is a notation to the fM referred. For each VN$_{Alg}$, $\mathcal{P}$, is possible to take the fM referred and change its VN$_{Alg}$ to another algorithm $\mathcal{P}$. Then, the fM$_{\mathcal{P}}$ obtained is not fM it is a new FM. Hence, for each fM a new FM can be obtained from the VN$_{Alg}$.

Now, I can define the relation *mod* in the set of all VN$_{Alg}$. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be a VN$_{Alg}$. I say that $\mathcal{P}_1 mod \mathcal{P}_2$ if and only if for any FM, fM, $\mathcal{L}(\text{fM}_{\mathcal{P}_1}) = \mathcal{L}(\text{fM}_{\mathcal{P}_2})$.

Is easy to demonstrate that the relation *mod* is an equivalent relation. Therefore, I can make a partition of all VN$_{Alg}$. Thus, is obtained a classification of FMs through of the VN$_{Alg}$, the set $\mathbb{Z}$eus/*mod*.

3.2 The Computational Model

The Computational Structure of a Formal Machine (CSFM) is a computational implementation of a FM. I am describing the CSFM in its version 0.03. A CSFM is a 3-tuple CSFM=(VN_{Alg},psm,$\mathcal{A}$,$\vdash$) where: VN_{Alg} is the Von Neumann's Algorithm of the machine, psm is the Physical State of the Machine, $\mathcal{A}$ is a Finite Automaton, and $\vdash$ is the Computational Operator of the FM and it is defined by an algorithm called Computational Operator Algorithm (COA)[17].

The psm is a matrix with rows and columns (See figure 2). Each column is an element of $ConfM \times \{0,1\}$. A column c_i can be seen as a pair $c_i = (c_i^1, c_i^2)$ where $c_i^1 \in ConfM$ is a configuration of the FM and $c_i^2 \in \{0,1\}$. When $c_i^2 = 1$ means that the configuration c_i^1 is active and when $c_i^2 = 0$ means that the configuration c_i^1 is not active. The rows of the psm matrix are split in two types of entries. The firsts rows of the psm, $C_1, C_2, ..., C_n$, are the components of the FM (elements of $CompM_B$) and the last row, denoted by σ_P, is an element of the Cartesian product $\{0,1\}^{|ConfM|}$. σ_P represents, by signalization with the

[17]An implementation of this can be download in `http://www.ipg.pt/user/~pavieira/private/SW/FM_OpComp_v002/FM_OpComp.jar`

number one, the set of configurations that are active. The Finite Automaton referred, in the previous paragraph, is the computational model where the $InstM$ is represented. (See figure 4) The Algorithm of the operator $\vdash$ is a step of computation of the FM and the VN_{Alg} describes how the FM works (See figure 5), here I describe the version 0.03.

The Von Neumann's Algorithm, VN_{Alg}

The Von Neumann's Algorithm, VN_{Alg}, such as already was referred previously, is an algorithm that obeys to a structure that is divided in three spaces, three zones. A definition zone, a setting zone and an execution zone (page 16). The execution zone is also called the Von Neumman's Cycle (VNC). The operator of computation $\vdash$ is used only in the VNC.

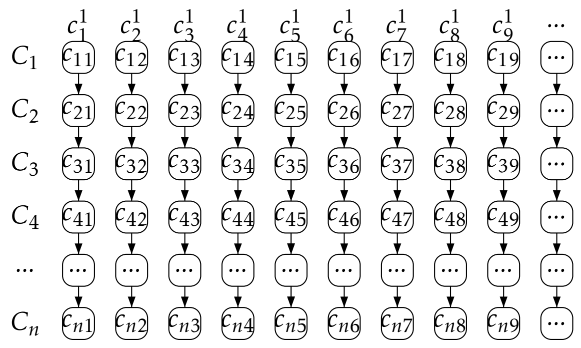
Physical State Machine of the fM, psm_{fM}
P The table psm can be seen as a matrix where the rows are labeled with the components of the fM and with an element σ_P that allows to label the active configurations. The columns are the configurations of the fM and their state is active or not.

$$\begin{tikzcd}
 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & \cdots \\
C_1 & c_{11} \arrow[d] & c_{12} \arrow[d] & c_{13} \arrow[d] & c_{14} \arrow[d] & c_{15} \arrow[d] & c_{16} \arrow[d] & c_{17} \arrow[d] & c_{18} \arrow[d] & c_{19} \arrow[d] & \cdots \arrow[d] \\
C_2 & c_{21} \arrow[d] & c_{22} \arrow[d] & c_{23} \arrow[d] & c_{24} \arrow[d] & c_{25} \arrow[d] & c_{26} \arrow[d] & c_{27} \arrow[d] & c_{28} \arrow[d] & c_{29} \arrow[d] & \cdots \arrow[d] \\
C_3 & c_{31} \arrow[d] & c_{32} \arrow[d] & c_{33} \arrow[d] & c_{34} \arrow[d] & c_{35} \arrow[d] & c_{36} \arrow[d] & c_{37} \arrow[d] & c_{38} \arrow[d] & c_{39} \arrow[d] & \cdots \arrow[d] \\
C_4 & c_{41} \arrow[d] & c_{42} \arrow[d] & c_{43} \arrow[d] & c_{44} \arrow[d] & c_{45} \arrow[d] & c_{46} \arrow[d] & c_{47} \arrow[d] & c_{48} \arrow[d] & c_{49} \arrow[d] & \cdots \arrow[d] \\
\cdots & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] \\
C_n & c_{n1} \arrow[d] & c_{n2} \arrow[d] & c_{n3} \arrow[d] & c_{n4} \arrow[d] & c_{n5} \arrow[d] & c_{n6} \arrow[d] & c_{n7} \arrow[d] & c_{n8} \arrow[d] & c_{n9} \arrow[d] & \cdots \arrow[d] \\
\sigma_P & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & \cdots
\end{tikzcd}$$

Figure 2: Physical state of the machine fM, psm_{fM}

wherein:
i) $C_i \in CompM_B$ with $i = 1, 2, ..., n$, $n = |CompM_B|$,
ii) $c_i = (c_i^1, c_i^2)$, $c_i^1 = (c_{1i}, ..., c_{ni}) \in ConfM$ with $i = 1, 2, ..., n$, $c_i^2 = d_i$ and
iii) $\sigma_P \in \{0, 1\}^{|ConfM|}$.

The behavior of the fM can be studied throughout of iterations or time. For that I define a function, called behavior function, $b(t) = psm(A, \sigma_P(t))$ where $\sigma_P(t)$ label the active configurations at the iteration or moment of time t and A is an invariant matrix



c_1^1 c_2^1 c_3^1 c_4^1 c_5^1 c_6^1 c_7^1 c_8^1 c_9^1 ...
C_1 c_{11} c_{12} c_{13} c_{14} c_{15} c_{16} c_{17} c_{18} c_{19} ...
C_2 c_{21} c_{22} c_{23} c_{24} c_{25} c_{26} c_{27} c_{28} c_{29} ...
C_3 c_{31} c_{32} c_{33} c_{34} c_{35} c_{36} c_{37} c_{38} c_{39} ...
C_4 c_{41} c_{42} c_{43} c_{44} c_{45} c_{46} c_{47} c_{48} c_{49} ...
...
C_n c_{n1} c_{n2} c_{n3} c_{n4} c_{n5} c_{n6} c_{n7} c_{n8} c_{n9} ...

Figure 3: $\mathcal{A}$ is a submatrix of the psm_{fM} matrix

In the theoretical construction of the *psm*, the set of configurations $ConfM$ can be partitioned in two sets. The set $c0_P = \{c_i \in ConfM :\ \text{and } \sigma_P(i) = d_i = 0\}$ and $c1_P = \{c_i \in ConfM :\ \text{and } \sigma_P(i) = d_i = 1\}$

Finite Automaton of the fM, $\mathcal{A}_{\text{fM}} = (Q, I, F, A, \Delta)$ with $\Delta \subseteq Q \times\ A \times Q$

The set of states, Q, of this Finite Automaton is the power set of the set of configurations, $ConfM$, of the fM, $\mathcal{P}(ConfM)$. $Q = \mathcal{P}(ConfM)$ and $A = InstM \cup \{\text{NOP}\}$. The operation NOP is defined in the following way. Suppose $c_P \subseteq ConfM$, from NOP you can obtain any $c'_P \subseteq ConfM$ where $c'_P \subseteq c_P$. Therefore, $(c_P, \text{NOP}, c'_P) \in \Delta$. Thus, there are a reason, through of the operator NOP, for the fact $c_P \vdash c'_P$, in slight notation $NOP(c_P) = c'_P$. $I = \mathcal{P}(ConfM_i)$ and $F = \mathcal{P}(ConfM_f)$. The set of transitions, Δ, is:

$\Delta = \{(c_P, NOP, c'_P) : c'_P \subseteq c_P\} \cup \{(c_P, inst, c'_P)$: $(inst \in InstM - \{NOP\}$ is a k-partial function[18]) $(\exists c_1, ..., c_k \in c_P)$:$\vec{c} = (c_1, ..., c_k)\ inst(\vec{c}) = c'_P\}$

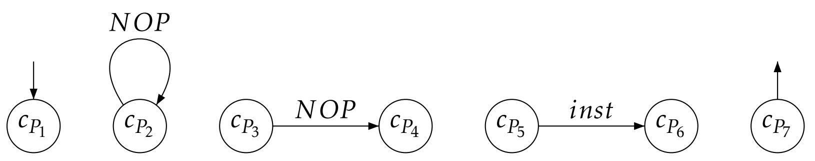


Figure 4: graphical representation of the automaton $\mathcal{A}_{\text{fM}}$

wherein:
i) c_{P_1} is an initial state of the $\mathcal{A}_{\text{fM}}$
ii) c_{P_2} is a loop. It is carried out by the instruction NOP. A loop in one state, as in the figure, is always carried out by the instruction NOP.
iii) $c_{P_3} \supseteq c_{P_4}$, because the edge between c_{P_3} and c_{P_4} is labeled by the instruction

[18] $f : X \longrightarrow Y$ is a partial function if $f|_{domain(f)} : domain(f) \longrightarrow Y$ is a function, the $domain(f) = \{x \in X | \exists y \in Y : f(x) = y\} \subsetneq X$. If exist a set Z such that $X = Z^k$ (f is called a k-partial function). If f is such that $domain(f) = X$, f is called a total function

NOP.
iv) There is an edge, $inst$, between c_{P_5} and c_{P_6}. Suppose $inst$ is a k-partial function and that $c_{P_5} \subseteq dom(inst)$. Then exists $c_1, ..., c_k \in c_{P_5}$ such that $inst(\vec{c}) = c_{P_6}$ with $\vec{c} = (c_1, ..., c_k)$.
v) c_{P_7} is a final state of the $\mathcal{A}_{\mathrm{fM}}$

Computation Operator Algorithm (COA) of the fM, $\vdash$

This algorithm shows, by the existence of the "choices" and by the implementation of the "motives" that the FMs are highly versatile and for this they have high capacity to be a good model for particular engineering problems. The first "motive" is the motive$_0$ this motive serves to introduce in the FMs the number total of configuration of the machine. Now I am going to describe the COA.

// Start the processing The description of the step of computation $c_P \vdash c'_P$[19]
(state q_0) 0. Choose a set of configurations, c_{P_0} such that $c_{P_0} \in ConfM_i$ (motive$_1$). If $c_{P_0} \in ConfM_f$ go to state 9 if not state 1.
(state q_1) 1. Put $\sigma_P \leftarrow null$. Update psm, putting c_{P_0} in the row σ_P, $\sigma_P \leftarrow c_{P_0}$. Go to state 2
(state q_2) 2. $counter \leftarrow 0$. Go to state 3
// A step of computation $\vdash$. From here begins the processing of the operator $\vdash$. This is a loop between the step 3 up to 9.
(state q_3) 3. Read the σ_P (the active configurations) from the psm, $c_{P_i} \leftarrow \sigma_P$. Go to state 4
(state q_4) 4. Are you going to use an instruction? (motive$_2$) If yes go to **instruction** if not { $c_{P_r} = \emptyset$ and state 5}.
(state q_5) 5. Are you going to use a NOP? (motive$_3$) If yes go to **NOP** if not { $c_{P_j} = \emptyset$ and go to state 6}
(state q_6) 6. $counter \leftarrow counter + 1$ and build $c_{P_i} = c_{P_r} \cup c_{P_j}$. Go to state 7
(state q_7) 7. Update psm. Go to state 8
(state q_8) 8. If $c_{P_i} \cap ConfM_f \neq c_{P_i}$ (motive$_4$) {go to state 3 } if not {(motive$_9$)} and go to state 9}
// end the processing
(state q_9) 9. $c'_P \leftarrow c_{P_i}$(motive$_{10}$). End

instruction:
(state $q_{4.i}$) i) Choose an instruction $inst \in InstM$. Suppose without loss of generality that $inst$ is a k-partial function (motive$_5$). Go to state 4.ii)
(state $q_{4.ii}$) ii) Choose k configurations that are elements of c_{P_i}, $c_1, .., c_k \in c_{P_i}$, $\vec{c} = (c_1, .., c_k)$ (motive$_6$). Go to state 4.iii)
(state $q_{4.iii}$) iii) Apply $I(\vec{c}) = c_{P_r}$ (motive$_7$). End instruction.

NOP:

[19]The motive$_0$ has c_P in its arguments

(state $q_{5.i}$) i) Choose a set of configuration, c_{P_j}, such that $c_{P_j} \subseteq c_{P_i}$ (motive$_8$). Go to step 5.ii)
(state $q_{5.ii}$) ii) End NOP

In the algorithm there are eleven subroutines called motives (motive$_0$, motive$_1$, motive$_2$, motive$_3$, motive$_4$, motive$_5$, motive$_6$, motive$_7$, motive$_8$, motive$_9$ and motive$_{10}$). The "motives" are software, abstract classes that are overwrite in the CSFM implemented (See figure 6). The COA algorithm is an algorithm wildly indeterministic. The implementation of the "motives" make the COA a deterministic algorithm. This implementation should be consequence of the problem that you need solve. For a developer that wants to solve some problem using FMs it needs to create the psm table of the problem, give the instructions of the machine and implement the "motives".

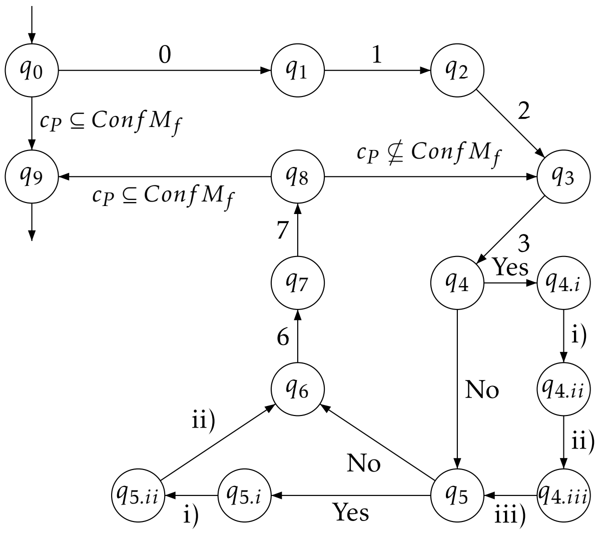


Figure 5: Automaton of the Computation Operator, $\vdash$.

Thus, as fM is a FM, I denoted the Computational Structure of the fM as a 4-tuple, CSFM$_{\text{fM}}$=(psm$_{\text{fM}}$, FA$_{\text{fM}}$, $\vdash_{\text{fM}}$,$VN_{Alg_{\text{fM}}}$) where psm$_{\text{fM}}$ is a psm of the fM, FA$_{\text{fM}}$ is a Finite Automaton of the fM, and $\vdash_{\text{fM}}$ is the computational operator of the fM, and VN_{Alg} is the Von Neumann's Algorithm, in agreement with the previous definition of CSFM in page 27.

The following table shows for each "motive" its arguments (inputs) and the outputs. The table is constructed in the supposition that the system is in the cycle of iteration, or order, k.

The first raw is labeled by I (inputs introduced in the FM from the external world), C (is the number of the cycle), and by c_{P_0}, c_{P_i}, c_{P_j}, c_{P_r}, *inst*, NOP and O. The c_{P_0}, c_{P_i}, c_{P_j}, c_{P_r}, *inst* and NOP are as described in COA Algorithm, O is the output of each "motive". In the column labeled by I (input column) there are variables *confgL* and *envir*. *confgL* is the total number of configurations and

Table 1: Table about inputs and outputs of the motives, cycle of execution k

	I	C	c_{P_0}	C	c_{P_i}	C	c_{P_j}	C	c_{P_r}	C	$inst$	C	NOP	O
$motive_0$	confgL	1												
$motive_1$	envir	1												c_{P_0}
$motive_2$		1	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	boolean
$motive_3$		1	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	boolean
$motive_4$		1	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	boolean
$motive_5$		1	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$InstM$
$motive_6$		1	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	c_{P_i}
$motive_7$		1	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	$< k$	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	c_{P_r}
$motive_8$		1	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$< k$	$\surd$	c_{P_j}
$motive_9$		1	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	answer
$motive_{10}$		1	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	$\leq k$	$\surd$	c_{P_i}

envir is a the set of the environment's data. When the table is

Table 2: excerpt from above Table, cycle of execution k

....	C	c_{P_i}	...	...
motive$_l$	$\leq k$	$\surd$	...	...
motive$_g$	¡ k	$\surd$	...	...
motive$_h$	...	....	...	c_{P_i}

In table 2 the first row of the column (C, c_{P_i}) is $(\leq k, \surd)$. $(\leq k, \surd)$ means which in the argument of the motive$_l$ it is possible to have the set of configurations c_{P_i} that are generated until the cycle (or iteration) k. The second row of the column (C, c_{P_i}) is $(< k, \surd)$. This means that in the argument of the motive$_g$ is possible to have the c_{P_i} that are generated until the cycle or iteration less than k. In the motive$_h$ in column O is the output. This output is an element of $\mathcal{P}(ConfM)$, in this case the set of configuration c_{P_i}.

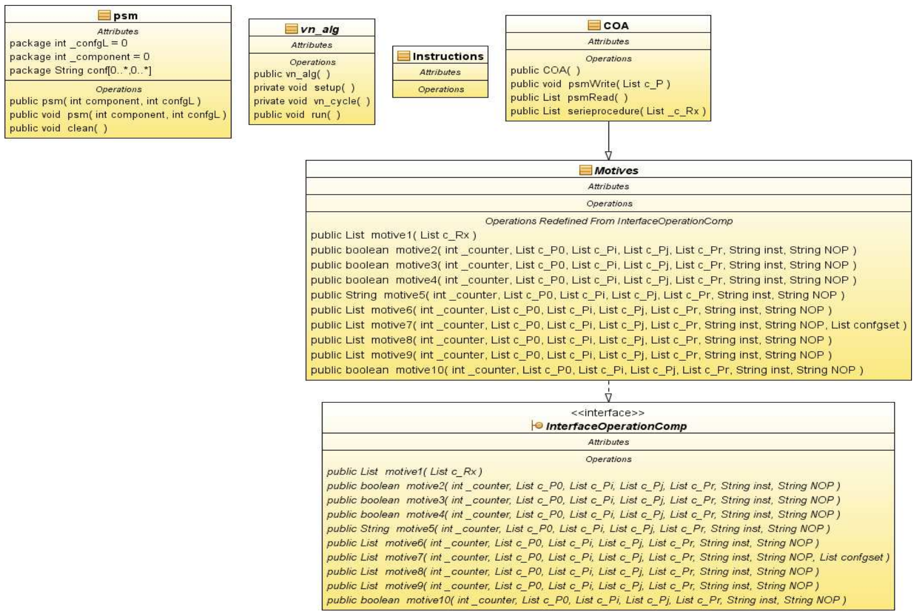

psm
Attributes
package int _confgL = 0
package int _component = 0
package String conf[0..*,0..*]
Operations
public psm(int component, int confgL)
public void psm(int component, int confgL)
public void clean( )
vn_alg
Attributes
Operations
public vn_alg( )
private void setup( )
private void vn_cycle( )
public void run( )
Instructions
Attributes
Operations
COA
Attributes
Operations
public COA( )
public void psmWrite(List c_P)
public List psmRead( )
public List serieprocedure(List _c_Rx)
Motives
Attributes
Operations
Operations Redefined From InterfaceOperationComp
public List motive1(List c_Rx)
public boolean motive2(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public boolean motive3(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public boolean motive4(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public String motive5(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public List motive6(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public List motive7(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP, List confgset)
public List motive8(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public List motive9(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public boolean motive10(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
<<interface>>
InterfaceOperationComp
Attributes
Operations
public List motive1(List c_Rx)
public boolean motive2(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public boolean motive3(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public boolean motive4(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public String motive5(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public List motive6(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public List motive7(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP, List confgset)
public List motive8(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public List motive9(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)
public boolean motive10(int _counter, List c_P0, List c_Pi, List c_Pj, List c_Pr, String inst, String NOP)

Figure 6: UML diagram of the CSFM version 0.03

The FMs have two distinct ways of work. They process the tasks through of a serial procedure or a parallel procedure

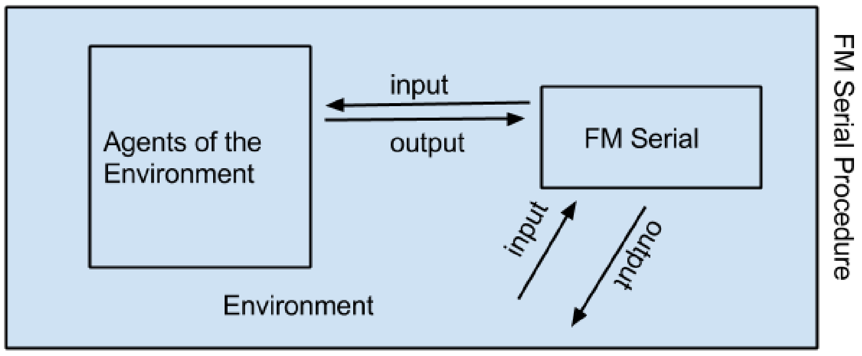


Figure 7: FM serial procedure

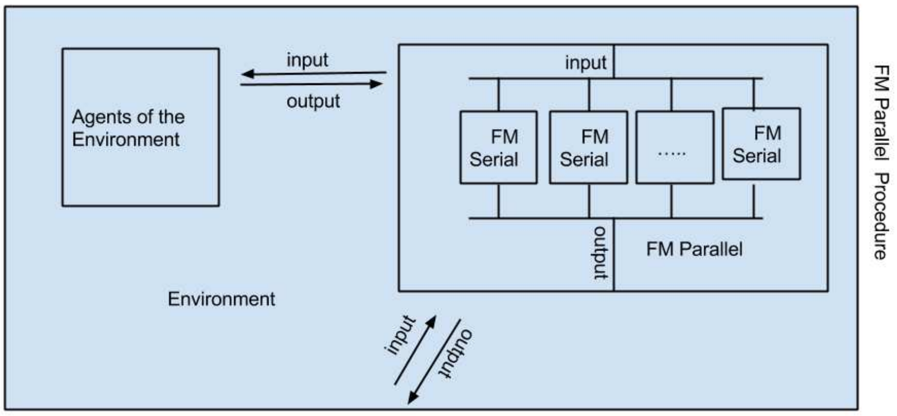


Figure 8: FM parallel procedure

serial procedure: In a FM serial procedure the input is obtained from the environment and/or from an agent. The input, in the machine, is translated to the language of the machine, a set of configuration c_T. In the machine, the processing of c_T is put in a set of configuration $c_P \in ConfM$. After to introduce the input, that was written, the FM runs the CSFM and processes the c_P and produces the output set of configurations c'_P. The c'_P is translated to the environment language and is sent to the environment. Thus, is made a new computation.

parallel procedure: The FM with a parallel procedure receives the input from the environment or agents. This input is translated in the language of the machine, c_P. After that, the machine, nside itself, trigger several threads (suppose k threads) and c_P is translated for each one of the threads. Each one of the threads is a FM Serial procedure. The c_P translation in thread i is c^i_P. Thus, $c^1_P, c^2_P, ..., c^k_P$ are the inputs respectively of the thread $1, 2, ..., k$. Then I have in the FM parallel procedure the output of threads $c'^1_P, c'^2_P, ..., c'^k_P$. The c'^i_P is obtained from the thread i c^i_P.

After this is necessary to produce only one output, c'_P, that is obtained from the output's threads. For last, the c'_P is translated in the language of the environment.

3.3 How to conceive a problem in FMs

To write a problem in FMs you should have in count the following:
- The components of the FM are the components of the problem
- The instructions of the FM are actions to carried out to solve the problem
- In the psm should be possible to appear any possible state of the problem
- NOP is an operator that serves to restrict the number
of active configurations of the machine or to provoke delays in the system with

the machine without make any processing
- In the definition of the operator $\vdash$ is necessary to do some “choices” and implemented the “motives”. This should be done having in attention the problem that is being solving

Examples of problems:
- i) Tic Tac Toe game (TTTGame). In this example the FM appears as one of the players of the game, and the components of the TTTgame are similar to the components of the FIL Game (See Table 16 page 80).
- ii) Four In Line game (FILGame). In this example the FM appears as one of the players of the game (See Table 16 page 80).
- iii) Checker game. A FM that is a Checker player can be projected as follows. The components of the FM are the following components: C1 is the moves of the white checkers, C2 is the moves of the black checkers, C3 is a lot of complete games, each one indexed by the number of defeats to the white checkers, C4 is a lot of complete games each one indexed by the number of defeats to the black checkers.

The inside thought of the machine is made in 4 loops in the COA of the FM. In each loop, in the COA Algorithm, are generated all or almost the possible moves. In the final of the first loop the possible moves generated are divided equally in four threads. From the first loop in each one of the threads are generated the possible moves alternating between the FM and the Human Being, one time playing the FM another playing the Human Being. After all of this are chosen the best moves that the machine should make.

4 Building Technology

4.1 Formal Machine in a Database

This section is divided in 4 sub sections: Formal Machine in a Databases, The FCss and FMs, A software for simulate formal computational systems, games and formal machines. In the subsection Formal Machine in a Databases is presented the data structure of a FM and the structure of a Database of FMs. The presentation is accompanied by examples of a FM obtained from a drive FA_FM. In the subsection The FCSs and FMs are presented theorems that show that several FCSs are FMs, in the follows sub section is presented a software that I developed and that allows to simulate several FCSs. For last I presented two games where the FM are one of the players.

4.1.1 Data Structure to support Formal Machines

For all, what is done in this section, fM is a FM. Whenever fM is obtained from a FCS, $\mathcal{A}$, fM($\mathcal{A}$) denoted the FM obtained from the FCS $\mathcal{A}$. The mathematical construction of how to transform FCSs, a lot of them, in FMs can be seen in the proof of Theorem 4.1 page 42[20]. I begin this section by giving two acronyms, the DA_FA_FM and the FA_FM, that are respectively DataStructure_Finite Automata_Formal Machine and Finite Automata_Formal Machine. The DS_FA_FM [21] is a software class with variables and methods that allows to define the DataStructure (DS) of a FM which is built from a Finite Automaton (FA). The FA_FM [22] is a software class supported pn the DS_FA_FM (See Appendix, section 10.2) that transforms any FA in a FM and store it in a Databases (DB) of FMs. The FA_FM is a drive for Finite Automata (FAs). For a FA, $\mathcal{A}$, the FA_FM class allows to store the fM($\mathcal{A}$) in a BD [23] of FMs.

The DB is defined in agreement with the DS of a FM. The aim of this section is to give the DS of a FM.

The description of the DS, here presented, is written in pseudocode close to Java Language[Sha11] [Laf03]. The description of the DS for a FM will be accompanied by several examples of how it is implemented in FA_FM and how to use its implementation to create the DB of the fM($\mathcal{A}_3$) for the Automaton $\mathcal{A}_3$ (See page 31).

This subsection (Formal Machine in a Database) is divided in seven sub subsections. Since the sub subsection 4.1.2 up 4.1.4 are referred the objects that are necessary to build the DS of a FM. In the sub subsection 4.1.5 up 4.1.8 is showed how to use the DS to build the DBs of FMs. All the text of this section is filled of Examples, that, as in all the thesis has a end mark, a square. For a more easy read of this subsection (Formal Machine in a Database) only here,

[20]The proof begins in subsection 4.2.2 page 44

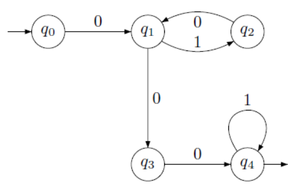
[21]http://www.ipg.pt/user/~pavieira/private/SW/javadoc_ds_fa_fm/index.html□

[22]http://www.ipg.pt/user/~pavieira/private/SW/javadoc_fa_fm/index.html,
http://www.ipg.pt/user/~pavieira/private/SW/exampleClassFA_FM.pdf□

[23]http://www.ipg.pt/user/~pavieira/private/SW/Database_FM_SQL.pdf□

[H]

Figure 9: The Automaton $\mathcal{A}_3$ and the fM($\mathcal{A}_3$)

Automaton $\mathcal{A}_3$	Formal Machine, fM($\mathcal{A}_3$)
$\mathcal{A}_3$  q_0 q_1 q_2 q_3 q_4 0 0 1 0 1 0 $A = \{0,1\}$ is a code $\delta(q_0,0) = q_1$ $\delta(q_1,1) = q_2$ $\delta(q_2,0) = q_1$ $\delta(q_1,0) = q_3$ $\delta(q_3,0) = q_4$, $\delta(q_4,1) = q_4$	$\rightarrow Comp_B = \{Q, T_I, p_I\}$: $Q = \{q_0,q_1,q_2,q_3,q_4\}$, $T_I = A^*\flat^\omega$, $p_I = \mathbb{N}$ $\rightarrow Comp_R = \{R_\delta\}$: $(q_0, u0v\flat^\omega, \lvert u\rfloor+1, q_1, \lvert u\rfloor+2)$, $(q_1, u0v\flat^\omega, \lvert u\rfloor+1, q_3, \lvert u\rfloor+2)$ $(q_1, u1v\flat^\omega, \lvert u\rfloor+1, q_2, \lvert u\rfloor+2)$ $(q_2, u0v\flat^\omega, \lvert u\rfloor+1, q_1, \lvert u\rfloor+2)$, $(q_3, u0v\flat^\omega, \lvert u\rfloor+1, q_4, \lvert u\rfloor+2)$ $(q_4, u1v\flat^\omega, \lvert u\rfloor+1, q_4, \lvert u\rfloor+2)$ $\rightarrow ConfM = Q \times A^*\flat^\omega \times \mathbb{N}$ $ConfM_i$: $(q_0, 0u\flat^\omega, 1)$ $ConfM_f$: $(q_4, u\flat^\omega, \lvert u\rfloor+1)$ $\rightarrow$ Instructions: $I(c_\bullet) = c_{\bullet\delta}$ $c_1 = (q_0, u0v\flat^\omega, \lvert u\rfloor+1)$, $c_{1\delta} = (q_1, u0v\flat^\omega, \lvert u\rfloor+2)$ $c_2 = (q_1, u0v\flat^\omega, \lvert u\rfloor+1)$, $c_{2\delta} = (q_3, u0v\flat^\omega, \lvert u\rfloor+2)$ $c_3 = (q_1, u1v\flat^\omega, \lvert u\rfloor+1)$, $c_{3\delta} = (q_2, u1v\flat^\omega, \lvert u\rfloor+2)$ $c_4 = (q_2, u0v\flat^\omega, \lvert u\rfloor+1)$, $c_{4\delta} = (q_1, u0v\flat^\omega, \lvert u\rfloor+2)$ $c_5 = (q_1, u0v\flat^\omega, \lvert u\rfloor+1)$, $c_{5\delta} = (q_3, u0v\flat^\omega, \lvert u\rfloor+2)$ $c_6 = (q_3, u0v\flat^\omega, \lvert u\rfloor+1)$, $c_{6\delta} = (q_4, u0v\flat^\omega, \lvert u\rfloor+2)$ $c_7 = (q_4, u1v\flat^\omega, \lvert u\rfloor+1)$, $c_{7\delta} = (q_4, u1v\flat^\omega, \lvert u\rfloor+2)$ with $u, v \in A^*$

the Meta-objects, tables and footnotes are also ended with a square. The DB is represented in tables. From now, I am going to define the DS of a FM.

4.1.2 Constants, Variables and Arrays of integers

In this subsections I present the constants and variables necessary to build the FMs.

Examples 4.1 *In FA_FMs the parameters have always the following values:* $|CompM_B| = n_{components} = 3$, $|CompM_R| = n_{relations} = 1$ *and* $|InstM| = n_{instructions} = 1$. *The other parameters do not have fixed values, the* $components[i]$ *with* $0 \leq i \leq 2$, *the* $relation[0]$ *and the* $instructions[0]$.□

4.1.3 The extension and understanding methods

The objects that constitute a FM can be defined by extension, using what I call the *extension* methods, or by understanding, using what I call the *understanding* methods. Thus, any DB of FMs can be built through of these two types of methods. Suppose, without loss of generality, that you have an object, O_{obj}, of a FM that is a set where U is, in a certain sense, the universe of O_{obj}.

When the object, O_{obj}, is defined by *extension*, U is the universe of O_{obj} in the sense that U is a set and $O_{obj} \subseteq U$. When the object O_{obj} is created in the DB by the use of the *extension* method, the element O_{obj} is described in the

Table 3: FM Structure: Constants, Variables and Arrays

Constants, Variables and Arrays of integers	**Description**
final INFINITE=$2^{31}-1$	
final int $n_{components}$	number of components of the fM, $\vert CompM_B\vert = n_{components}$
int[ ] $components = new\ int[n_{components}]$	if $C_i \in CompM_B$ is finite, $components[i] = \vert C_i\vert$, otherwise $components[i]$ =INFINITE
final int $n_{relations}$	number of relations of the fM, $\vert CompM_R\vert = n_{relations}$
int[ ] $relations = new\ int[n_{relations}]$	if $R_i \in CompM_R$ is finite, $relations[i] = \vert R_i\vert$, otherwise $relations[i]$ =INFINITE
final int $n_{instructions}$	number of instructions of the fM $\vert InstM\vert = n_{instructions}$
int[ ] $instructions = new\ int[n_{instructions}]$	if D_{I_i} (domain of I_i) is finite, $I_i \in InstM$, $instructions[i] = \vert D_{I_i}\vert$,
otherwise	$instructions[i]$ = INFINITE .□

DB by the storage, in the DB, of all elements that belong to O_{obj}. Thus, O_{obj} is, through of an *extension* method, completely defined by the elements that belong to it (See Example 4.2).

Examples 4.2 *In* fM($\mathcal{A}_3$) *I can define the object $Q \in CompM_B$ through the extension method*[24] *of the FA_FMs. The universe U of Q is the* ***data_type*** *string. Thus, U is the set of all strings. Therefore, using Set Theory notation, $Q \subseteq string$ (See Table 4).*□

[24]The *extension* method that implemented $Q \in CompM_B$ of the fM($\mathcal{A}_3$) has the following expression, $extensionM_B(“\mathcal{A}_3”,“Q”,“string”,“\underbrace{q0;q1;q2;q3;q4}_{\text{See Appendix 10.2}}”)$.□

Table 4: The table of the DB of fM($\mathcal{A}_3$) after the use of the extension method to create the object Q. pk-primary key.

(pk) FM	(pk) Counting	(pk) N_{C_i}	U_i	Elements/p(x)	Method
$\mathcal{A}_3$	1	Q	string	q_0	extension
$\mathcal{A}_3$	2	Q	string	q_1	extension
$\mathcal{A}_3$	3	Q	string	q_2	extension
$\mathcal{A}_3$	4	Q	string	q_3	extension
$\mathcal{A}_3$	5	Q	string	q_4	extension

□

When the object O_{obj} is defined by *understanding* in the DB not exist explicitly the elements of O_{obj} but a *well formed formula*(wff) of a Propositional Logic (PL), First Order Logic (FOL) or a High Order Logic (HOL). The Logic in use, of PL, FOL or HOL, is denoted by U. Thus, in the use of the *understanding* method, U is the universe of O_{obj} in the sense that U is one of following logics: a PL, a FOL, or a HOL and O_{obj} is a wff of it. When an *understanding* method is in use, there is a set U_{var} that is: i) a set of all propositions of U if U is a PL or ii) is a set of constants, variables and terms of U if U is a FOL or a HOL (See Example 4.3).

Examples 4.3 *An example of the use of the understanding method can be seen in the definition, in* fM($\mathcal{A}_3$)*, of the object* $T_I \in CompM_B$[25]*. The component U of the object* T_I *is a 2-tuple* $(\{0,1,\mathfrak{b}^{\omega}\};\{\epsilon,\cdot,+,^{+},^{*}\})$ *and the expression* $0(10)^*001^*\mathfrak{b}^{\omega}$ *is a wff of U[CoE11] (See Table 5).*□

Table 5: The table of the DB of fM($\mathcal{A}_3$) after the use of the understanding method to create the object T_I. pk- primary key.

(pk) FM	(pk) Counting	(pk) N_{C_i}	U_i	Elements or p(x)	Method
$\mathcal{A}_3$	6	T_I	$\{0,1,\mathfrak{b}^{\omega}\};\{\epsilon,\cdot,+,^{+},^{*}\}$	$0(10)^*001^*\mathfrak{b}^{\omega}$	understanding

□

4.1.4 Meta-objects of a FM

Now, I am going to define the Meta-objects of a FM. These objects are divided into atomic and derived Meta-objects. The atomic Meta-objects are **data_type**s, modifiers and identifiers.

[25]The understanding method of the class FA_FM that implemented $T_I \in CompM_B$ of the fM($\mathcal{A}_3$) has the following expression, *understandingM*$_B$("$\mathcal{A}_3$","T_I","$(\{0,1,\mathfrak{b}^{\omega}\};\{\epsilon,\cdot,+,^{+},^{*}\})$","$0(10)^*001^*\mathfrak{b}^{\omega}$").□

Meta-object 1 *Atomic Meta-objects of the FMs*

data_type=*byte*|*short*|*int*|*long*|*float*|*double*|*char*|*string*|*boolean*|*conf*
modifier={*public, private, protected*}
identifier=$(letters)(letters + digits)^*$
$letters$=_, *A,..., Z, a,..., z (the underscore is taken as a letter)*
$digits$=*0,1,...,9*

(End of Meta-object 1)□

The **data_type** conf, is not a primitive **data_type** of a programming language. Thus, is necessary to define it. The **data_type** conf is a finite Cartesian product as follows.

$$\text{conf}=\{00, 01, 10, 11\} \times (\overbrace{C_1 \times ... \times C_{n_{components}}}^{\in ConfM}),$$

where 00 (respectively, 01, 10, 11) corresponds to a configuration that is neither initial nor final (respectively, is not initial and is final, is initial and not final, and is initial and final). The type of the configurations of an FM is as follows:

Table 6: Type of configurations of an FM

00	the configuration is not initial neither final
01	the configuration is not initial and is final
10	the configuration is initial and isn't final
11	the configuration is initial and final

The **data_type** conf is implemented as a class, the class conf (See Meta-object 2)(See Example 4.4)

Meta-object 2 ***data_type** conf*
public class conf{

```
public conf(String C_0,data_type C_1,...,data_type C_n){
    // where C_0 ∈ {00, 01, 10, 11}
    publ ic i=C_0.charAt(0);
    public f=C_0.charAt(1);
    /*
    Eventually build the objects C_1, ..., C_n
    for use of the methods extension and/or understanding
    */
    ..................... }
```

} *(End of Meta-object 2)*□

Examples 4.4 *An example of the definition of the class* conf *can be seen in FA_FM (See Example 4.5). Examples of objects with the* **data_type** *conf in* fM*($\mathcal{A}_3$) are:*
$(\underbrace{10}_{\text{Table 6}}, \underbrace{q_0, 0ub^{\omega}, 1}_{\text{See } \mathcal{A}_3}), (00, \underbrace{q_1, u0vb^{\omega}, |u| + 1}_{c_2, \text{ See } \mathcal{A}_3}), (\underbrace{01}_{\text{Table 6}}, \underbrace{q_4, ub^{\omega}, |u| + 1}_{\text{See } \mathcal{A}_3})$.□

Examples 4.5 *Implementation of the* ***data_type*** *conf in FA_FM*

```
public void conf{
            public conf(String m, String q, String u, String n){
                // m ∈ {00,01,10,11};
                // q is one of these q0, q1, q2,q3, q4;
                // n is or represent an element of ℕ;
               public i=m.charAt(0);
               public f=m.charAt(1);
               ... } })□
```

After defining the Atomic Meta-objects of the FM, I build derivative Meta-objects such as, for example, the **data_type**s that are finite Cartesian products of **data_type**s, the class timesType (See Meta-object 3). A particular kind of this Meta-object is used to define the **data_type** of the elements of $CompM_R$(See Meta-object 4).

Meta-object 3 *// Finite Cartesian product of* ***data_types***

```
modifier class timesType{
               modifier multiType(data_type T1,...,data_type Tn){
               ..................... } } (End of Meta-object 3) □
```

4.1.5 On $CompM_B$

The elements of $CompM_B$, $CompM_B = \{C_1, C_2, ..., C_n\}$, are defined through the methods:

$extension(FM, N_{C_i}, U_i, \{v_1, ..., v_{components[i]}\})$ or

$understanding(FM, N_{C_i}, U_i, p(x))$,

wherein U_i and N_{C_i} are, respectively, the universe and the alias of C_i. N_{C_i} is an **identifier**.

In the use of the *extension* method U_i is a **data_type**, seen as the set of all instantiations of its **data_type**. Thus, $C_i \subseteq U_i$. The *extension* method can only be used if mathematically $|C_i| < \infty$. The construction for *extension*, mathematically, represents the set C_i, $C_i = \{v_1, ..., v_{components[i]}\}$.

In the use of the understanding method U_i is a PL, a FOL or a HOL and $p(x)$ is a wff of U_i. Mathematically $understanding(FM, N_{C_i}, U_i, p(x))$ is the set $\{x \in U_{i,var} : p(x)\}$. Thus, $understanding(FM, N_{C_i}, U_i, p(x)) = \{x \in U_{i,var} : p(x)\}$.

The set $CompM_B$ in a DB, it will appear as in Table 7[26]. The *extension* and *understanding* methods, in $CompM_B$, act on the table and write there. The *extension* method adds new rows to the table, one for each element v_i. The *understanding* method puts the logic U_i and a wff $p(x)$ of the U_i into the table. This allows to an element, x, of the $U_{i;var}$ that it belongs to the set C_i if it verifies the wff $p(x)$ (See Table 7).

Table 7: Table of the set $CompM_B$.

FM	Counting	N_{C_i}	U_i	Elements/p(x)	Method
FM	1	C_i	...	v_1	extension
FM	2	C_i	...	....	extension
FM	$n_{components}$	C_i	...	$v_{n_{components}}$	extension
FM	$n_{components}$ +1	C_j	...	$p(x)$	understanding
FM	...	...	...	...	...

□

Examples of the implementations of *extension* and *understanding* methods on elements of the set $CompM_B$, can be seen in the Examples 4.2, 4.3.

4.1.6 On $CompM_R$

I begin this sub subsection by defining a new object. This object is an array of distinct **data_type**s, the object productU (See Meta-object 4). An example of achievement of that object in FA_FM is the Example 4.7.

Meta-object 4 ***data_type*** *of the elements* $CompM_R$

```
public class productU{
        public productU(data_type U_(i,1),...,data_type U_(i,t)){
        ..................... }
}
```
productU $U_{(i,1)} \times ... \times U_{(i,t)}$ = *new productU*$(U_{(i,1)}, ..., U_{(i,t)})$
(End of Meta-object 4)□

The **Meta-object 5** is a particular case of the **Meta-object 4** but as the **Meta-object 5** is only to define the **data_type** of the elements of $CompM_R$. These

[26]In the following document is presented, the schema of the tables, 4 tables, for FA_FMs (Formal Machines obtained from Finite Automata).This tables constitute the DB of a FA_FM http://www.ipg.pt/user/~pavieira/private/SW/Database_FM_SQL.pdf

objects are very important in FMs, I should define these Meta-objects separately(See Example 4.6).

Examples 4.6 *The class productU should be built to allow to define the elements of $CompM_R$. In FA_FM, $|CompM_R| = 1$. Thus, for FA_FM, it is only necessary to build one class productU(See Example 4.7).*□

Examples 4.7 ***data_type*** *of the elements $CompM_R$ of a FA_FM*
public class productU{
    public productU(string _Q1, string _T_I, int _p_I1, string _Q2, int _p_I2){
    .... } }(See Example 4.8).□

Examples 4.8 *After obtaining the class productU, one of its elements is instantiated to define the universe of the relation R_δ of a FA_FM. This universe is an universe in the sense that is used in an extension* method. *So the universe of R_δ is prodR*

productU prodR;
prodR[27]*=new productU(string _Q1, string _T_I, int _p_I1, string _Q2, int _p_I2);*

□

The elements of $CompM_R$, $CompM_R = \{R_1, R_2, ..., R_m\}$, can be defined by:

$extension(FM, N_{R_i}, U_{(i,1)} \times ... \times U_{(i,t)}, \{v_1, ..., v_{relations[i]}\})$ or

$understanding(FM, N_{R_i}, U_{(i,1)} \times ... \times U_{(i,t)}, p(x))$,

wherein
$U_{(i,1)} \times ... \times U_{(i,t)}$ and N_{R_i} are, respectively, the universe and the alias of the relation R_i. The $(U_{(i,1)} \times ... \times U_{(i,t)})_{var} = U_{(i,1),var} \times ... \times U_{(i,t),var}$. In the use of the *extension* method each $v_j \in U_{(i,1)} \times ... \times U_{(i,t)}$. In the definition by extension, mathematically, the relation R_i corresponds to $R_i = \{v_1, ..., v_{relations[i]}\}$. In the definition by understanding, $understanding(N_{R_i}, U_{(i,1)} \times ... \times U_{(i,t)}, p(x))$ is mathematically the set $R_i = \{x \in (U_{(i,1)} \times ... \times U_{(i,t)})_{var} : p(x)\}$ and in the use of the *understanding* method, it is necessary to define a logic where $p(x)$ is a wff.

$$understanding(N_{R_i}, U_{(i,1)} \times ... \times U_{(i,t)}, p(x)) = \{x \in (U_{(i,1)} \times ... \times U_{(i,t)})_{var} : p(x)\},$$

$R_i \subseteq U_{(i,1)} \times ... \times U_{(i,t)}$ and $p(x)$ is a wff in $U_{(i,1)} \times ... \times U_{(i,t)}$.

For DBs on $CompM_R$ two tables can be created, as in Table 8 and Table 9. The $extension$ and $understanding$ methods, in $CompM_R$, act on Table 9.

[27] $prodR = \_Q1 \times \_T\_I \times \_p\_I1 \times \_Q2 \times \_p\_I2$.□

In some applications, as in FA_FM, Table 8 has only one entry. Thus, is only necessary to use only one table, the Table 9(See Example 4.9).

Table 8: Table on $CompM_R$. Define the different **data_type**s of the kind productU. pk - primary key.

(pk) $productU$ definition	U_1	...	U_t	R_i
p_1	...	...	...	...
...	...	...	...	...

□

Table 9: Table on $CompM_R$. pk-primary key.

(pk) FM	(pk) Counting	(pk) N_{R_i}	productU	Relation	Method
FM	...	...	p_i	v	extension
FM	...	...	p_j	$p(x)$ with $x \in p_{j,var}$	understanding
FM	...	...	p_j	...	...

□

Examples 4.9 *In Table 10 the set $CompM_R$ of* fM($\mathcal{A}_3$) *is implemented. In this implementation only the extension method (See Table 10) is used. It is used as follows:*
extensionM_R($\mathcal{A}_3$,R_δ,string, $(q_1, u0vb^\omega, |u\rfloor+1, q_3, |u\rfloor+2)$;$(q_1, u1vb^\omega, |u\rfloor+1, q_3, |u\rfloor+2)$; $(q_2, u0vb^\omega, |u\rfloor+1, q_1, |u\rfloor+2)$;$(q_3, u0vb^\omega, |u\rfloor+1, q_4, |u\rfloor+2)$;$(q_4, u1vb^\omega, |u\rfloor+1, q_4, |u\rfloor+2)$*).*□

Table 10: The table of the DB of $\mathcal{A}_3$ after the use of the extension method to create the object $CompM_R$. pk-primary key.

(pk) FM	(pk) Counting	(pk) N_{R_i}	productU	Relation	Method
$\mathcal{A}_3$	1	R_δ	$prodR$	$(q_0, u0vb^\omega, |u\rfloor+1, q_1, |u\rfloor+2)$	extension
$\mathcal{A}_3$	3	R_δ	$prodR$	$(q_1, u0vb^\omega, |u\rfloor+1, q_3, |u\rfloor+2)$	extension
$\mathcal{A}_3$	4	R_δ	$prodR$	$(q_1, u1vb^\omega, |u\rfloor+1, q_3, |u\rfloor+2)$	extension
$\mathcal{A}_3$	5	R_δ	$prodR$	$(q_2, u0vb^\omega, |u\rfloor+1, q_1, |u\rfloor+2)$	extension
$\mathcal{A}_3$	6	R_δ	$prodR$	$(q_3, u0vb^\omega, |u\rfloor+1, q_4, |u\rfloor+2)$	extension
$\mathcal{A}_3$	7	R_δ	$prodR$	$(q_4, u1vb^\omega, |u\rfloor+1, q_4, |u\rfloor+2)$	extension

□

4.1.7 On the machine configurations $ConfM$, $ConfM_i$, $ConfM_f$

The implementation of the set of configurations of a FM, $ConfM$, is done using the **data_type** conf. The **data_type** conf will be also treated as a set, the set conf[28] and its elements will be called configurations[29].

$$extension(FM, \overbrace{N\_c1; N\_c2; ...., N\_c_k}^{\text{alias of the configurations}}, \overbrace{c_1; c_2; ...; c_k}^{\text{elements of the \textbf{data\_type} conf}}),$$

$$understanding(FM, U, p(x)),$$

wherein $c_j = (m_j, \overbrace{c_{j1}, c_{j2}, ..., c_{jn_{components}}}^{\text{element of } ConfM}) \in$ conf, $m_j \in \{00, 01, 10, 11\}$, N_{c_j} is an alias of a configuration c_j and $c_j(i) = c_{ji} \in C_i$ for $1 \le j \le k$.

In the definition of the understanding method, $understanding(FM, U, p(x))$ with U_{var} = conf is, mathematically, the set $understanding(FM, U, p(x)) = \{x \in$ conf $: p(x)\}$.

In tables, the storage of the conf in the DB can be seen in Table 11. The *extension* and the *understanding* methods, in conf, act on the table. The *extension* method creates rows in the table putting element of $ConfM$ and its classification in each row (See Table 11). The *understanding* method puts, in the table, the property that the elements of conf need to verify. In a configuration, an element of conf, is a pair where the 2nd element of the pair is an elements of $ConfM$ and the 1st is its classification(See Example 4.10).

Table 11: Table of the conf, $c_j = (m_j, c_{c_j})$ with $m_j \in \{00, 01, 10, 11\}$ and m_j is the classification of the $c_{c_j} \in ConfM$.

FM	N_C	m	c_1	...	$c_{n_{components}}$	Method
FM	c_j	00	c_{j1}	...	$c_{jn_{components}}$	extension

□

Examples 4.10 *For* fM($\mathcal{A}_3$) *the storage of the set* $ConfM$ *in DBs is done only through the extension method*[30] *(See Table 12).*□

[28]As set, conf=$\{00, 01, 10, 11\} \times ConfM$.□

[29]This can cause some ambiguity with the elements of $ConfM$, also called configuration, but it will be solved by the context.□

[30]The extension method in fM($\mathcal{A}_3$) to define the set $ConfM$ is:

$$extensionConf(FM, \text{“}c1; c2; c3; c4; c5; c6; c7\text{”}, \text{“}(00, q_0, u0v\flat^{\omega}, |u\rfloor + 1)\text{”}; \text{“}(00, q_1, u0v\flat^{\omega}, |u\rfloor + 1); (00, q_2, u1v\flat^{\omega}, |u\rfloor + 1); (00, q_2, u0v\flat^{\omega}, |u\rfloor + 1); (00, q_1, u0v\flat^{\omega}, |u\rfloor + 1); (00, q_3, u0v\flat^{\omega}, |u\rfloor + 1); (00, q_4, u1v\flat^{\omega}, |u\rfloor + 1)\text{”}).\square$$

Table 12: Table of fM($\mathcal{A}_3$) to define the set conf. pk-primary key.

(pk) FM	(pk) N_C	m	Q	T_I	p_I	Method
$\mathcal{A}_3$	c1	00	q_0	$u0vb^\omega$	$\lfloor u\rfloor+1$	extension
$\mathcal{A}_3$	c2	00	q_1	$u0vb^\omega$	$\lfloor u\rfloor+1$	extension
$\mathcal{A}_3$	c3	00	q_2	$u1vb^\omega$	$\lfloor u\rfloor+1$	extension
$\mathcal{A}_3$	c4	00	q_2	$u0vb^\omega$	$\lfloor u\rfloor+1$	extension
$\mathcal{A}_3$	c5	00	q_1	$u0vb^\omega$	$\lfloor u\rfloor+1$	extension
$\mathcal{A}_3$	c6	00	q_3	$u0vb^\omega$	$\lfloor u\rfloor+1$	extension
$\mathcal{A}_3$	c7	00	q_4	$u1vb^\omega$	$\lfloor u\rfloor+1$	extension

$\square$

4.1.8 On *InstM*

I start, this sub subsection, by defining an object, confK, to build arrays of k-tuples of configurations, k-tuples of **data_type**s conf.

Meta-object 5 *The class of a k-tuples of configurations, confK*
public class confK{
    public confK(conf c^1,...,conf c^k){
    .... } *}(End of Meta-object 5).*$\square$

The methods used to create instructions are:

$extension(FM;I,\vec{c},c_P)$ (See Example 4.11)

$understanding(FM,I,U,d_I,I(\vec{c}))$,

wherein $I \in InstM$ is an instruction of the FM, $\vec{c}$ is a k-tuple of configurations and c_P is a set of configurations that is obtained after using the instruction I over the k-tuple of configurations $\vec{c}$. U is the universe of the instruction I, in the sense of an understanding method. So, U is a LP, FOL, HOL and U_{var} = conf^k. d_I is a wff of U necessary to define the *domain of I, domain of I*= $\{\vec{c} \in U_{var} : d_I(\vec{c})\}$ and $I(\vec{c})$ is a function in U.

The *extension* and *understanding* methods, in *InstM*, act on the table as the Table 13.

Table 13: Table on $InstM$, $c_P = \{c_{P_1}, ..., c_{P_t}\}$. pk-primary key.

(pk) FM	(pk) Counting	(pk) N_I	U	domain	k-tuple of configurations	c_P	Method
FM	...	I_1	...	...	$\vec{c}$	c_{P_1}	extension
FM	...	I_1	...	...	$\vec{c}$	c_{P_2}	extension
FM	...	I_1	...	...	$\vec{c}$	....	...
FM	...	I_1	...	...	$\vec{c}$	c_{P_t}	extension
FM	...	I	...	...	$\vec{c}$	$I(\vec{c})$	understanding
FM	...	...	...	...	...	...	...

□

Examples 4.11 *In $\mathcal{A}_3$ the definition of an instruction $I \in InstM$ is done only through the extension method*[31] *(See Table 14).*□

Table 14: Table of fM($\mathcal{A}_3$) to define the instruction $I \in InstM$. pk-primary key.

(pk) FM	(pk) Counting	(pk) N_I	m_I	Q_I	T_I_I	p_I_I	m_F	Q_F	T_I_F	p_I_F	Method
$\mathcal{A}_3$	1	I	00	q_0	$u0v\flat^{\omega}$	$\|u\rfloor+1$	00	q_1	$u0v\flat^{\omega}$	$\|u\rfloor+2$	extension
$\mathcal{A}_3$	2	I	00	q_1	$u0v\flat^{\omega}$	$\|u\rfloor+1$	00	q_3	$u0v\flat^{\omega}$	$\|u\rfloor+2$	extension
$\mathcal{A}_3$	3	I	00	q_1	$u1v\flat^{\omega}$	$\|u\rfloor+1$	00	q_2	$u1v\flat^{\omega}$	$\|u\rfloor+2$	extension
$\mathcal{A}_3$	4	I	00	q_2	$u0v\flat^{\omega}$	$\|u\rfloor+1$	00	q_1	$u0v\flat^{\omega}$	$\|u\rfloor+2$	extension
$\mathcal{A}_3$	5	I	00	q_1	$u0v\flat^{\omega}$	$\|u\rfloor+1$	00	q_3	$u0v\flat^{\omega}$	$\|u\rfloor+2$	extension
$\mathcal{A}_3$	6	I	00	q_3	$u0v\flat^{\omega}$	$\|u\rfloor+1$	00	q_4	$u0v\flat^{\omega}$	$\|u\rfloor+2$	extension
$\mathcal{A}_3$	7	I	00	q_4	$u1v\flat^{\omega}$	$\|u\rfloor+1$	00	q_4	$u1v\flat^{\omega}$	$\|u\rfloor+2$	extension

4.2 The FCSs are FMs

This subsection is divided in four sub subsections, the first three of them are dedicated to demonstrate the theorems 4.1 and 4.2. The fourth subsections is a subsection where for several FCSs I say how to relate the tasks and languages performed and recognized by them with the tasks and languages performed and recognized by the FM obtained from the respectives FCSs. The 1^{st}, 2^{nd}, 3^{rd} and 4^{th} subsection are respectively called: Some of the current FCSs, Proving Theorem 4.1, Proving Theorem 4.2 and Tasks and performed Languages of the FCSs.

In this subsection, in the following two theorems, I claim that a number of FCSs [Hop08] [Sip13] such as deterministic machines: k-Turing Machines, Pushdown Automata, Transducers, Finite Automata, [Cut97] k-unbounded Register Machines; and [SiS94] Recurrent Neural Network, are FMs. Both th

[31]The method used is the extension method. It defines the instruction $I \in InstM$, $I(c)$ for each configuration c. We leave here one example, in FA_FM, of the use of the method for $\mathcal{A}_3$.

$$extensionInst(\text{“}\mathcal{A}_3\text{”}, \text{“}Inst\text{”}, \text{“}(00, q_0, u0vb^w, |u\rfloor+1)\text{”}, \text{”}(00, q_1, u0vb^w, |u\rfloor+2)\text{”})$$

proofs presented are algebraic and constructive and are made in the sub subsections that are follow. Then the proofs can be used as algorithms with the aim to implement in software these transformations.

Theorem 4.1 *Deterministic machines: k-Turing Machines, Transducers, Pushdown Automata, Finite Automata and k-unbounded Register Machines are FMs.*

Theorem 4.2 *Recurrent Neural Networks are FMs.*

4.2.1 Some of the current FCSs

In this sub subsection I present the definitions of several FCSs as they are defined in Mathematics and Computer Sciences [ArB09], [Cut97], [Hop08], [Sip13][32].

Definition of deterministic; k-Turing Machine, Pushdown Automaton, Transducer, Finite Automaton, unbounded Register Machine

i) definition of a k-Turing Machine $\mathcal{TM}_k$
A k-Turing Machine, $\mathcal{TM}_k$, is an 8-tuple,

$$\mathcal{TM}_k = (Q, A, \Gamma, O, \delta_{\mathcal{T}}, \mu_{\mathcal{T}}, I, F),$$

wherein
- Q is a finite set, called the *set of the states* of $\mathcal{TM}_k$,
- A is a finite set, called the *input alphabet* of $\mathcal{TM}_k$,
- Γ is a finite set, called the *processing alphabet* of $\mathcal{TM}_k$,
- O is a finite set, called the *output alphabet* of $\mathcal{TM}_k$,
- $\delta_{\mathcal{TM}} : (Q-F)\times(A\cup\{\flat\})\times(\Gamma\cup\{\flat\})^k \to Q\times\Gamma^k\times\{L,R,S\}^{K+1}$, is a partial function, called the *transition function* of $\mathcal{TM}_k$
- $\mu_{\mathcal{TM}_k} : (Q-F)\times(A\cup\{\flat\}\times(\Gamma\cup\{\flat\})^k \to O\times\{R,S\}$ is a partial function, called the *output function* of $\mathcal{TM}_k$
- $I \subseteq Q$ and $|I| = 1$, I is called the set of the *initial states* of $\mathcal{TM}_k$
- $F \subseteq Q$, F is called the set of the *final states* of $\mathcal{TM}_k$,
- $\flat \notin I\cup\Gamma\cup O$, $\hat{\imath} \notin I\cup\Gamma\cup O$.

Note 4.1 *i) A $\mathcal{TM}_k$ has k processing tapes, a tape for input and another for output. Sometimes I denote the k-Turing Machine only as $\mathcal{TM}$. That happens, in general, when it is not important the number of processing tapes of the machine or it is clear by the context how many processing tapes the machine has.*
ii) $k \in \mathbb{N}_0$.
ii) A Turing Machine, as defined here, does not do any processing from the final states.

[32]Although I am referred several traditional books on Computation Theory, the definitions presented here of deterministic machines: k-Turing Machines, Pushdown Automata, Transducers, Finite Automata and k-unbounded Register Machines do not follow any in particular. They are essentially equivalent definitions of those presented in the books referred.

ii) Definition of a Pushdown Automaton $\mathcal{AS}$

A Pushdown Automaton, $\mathcal{AS}$, is a 7-tuple

$$\mathcal{AS} = (Q, I, F, A, \Gamma, Z_0, \delta_{\mathcal{AS}}),$$

wherein
- Q is a finite set, called the *set of the states* of $\mathcal{AS}$,
- I is a subset of Q and $|I| = 1$, called the *set of initial states* of $\mathcal{AS}$,
- F is a subset of Q, called the *set of final states* of $\mathcal{AS}$,
- A is a finite set, called the *input alphabet* of $\mathcal{AS}$,
- Γ is a finite set, called the *stack alphabet* of $\mathcal{AS}$,
- O is a finite set, called the *output alphabet* of $\mathcal{AS}$,
- Z_0 is a letter, the letter that is in the back of the stack, called the *initial symbol of the stack* of $\mathcal{AS}$. $Z_0 \in \Gamma$ and Z_0 only occurs at the start, in the base, of the stack
- $\delta_{\mathcal{AS}} : Q \times (A \cup \{b\}) \times \Gamma \to Q \times \Gamma^*$ is a partial function, called the *transition function* of $\mathcal{AS}$

iii) definition of a Transducers, $\mathcal{AT}$

A transducer, $\mathcal{AT}$, is a 7-tuple,

$$\mathcal{AT} = (Q, I, F, A, O, \delta_{\mathcal{AT}}, \mu_{\mathcal{AT}})$$

wherein:
- Q is a finite set, called the *set of the states* of $\mathcal{AT}$,
- I is a subset of Q and $|I| = 1$, called the *set of the initial states* of $\mathcal{AT}$,
- F is a subset of Q, called the *set of final states* of $\mathcal{AT}$,
- A is a finite set, called the *input alphabet* of $\mathcal{AT}$,
- O is a finite set, called the *output alphabet* of $\mathcal{AT}$,
- $\delta_{\mathcal{AT}} : Q \times (A \cup \{b\}) \to Q$, is a partial function, called the *transition function* of $\mathcal{AT}$
- $\mu_{\mathcal{AT}} : Q \times (A \cup \{b\}) \to O$, is partial function, called the *function of output* of $\mathcal{AT}$

iv) definition of a Finite Automaton $\mathcal{A}$

A Finite Automaton, $\mathcal{A}$, is a 5-tuple,

$$\mathcal{A} = (Q, I, F, A, \delta_{\mathcal{A}}),$$

wherein
- Q is a finite set, called the *set of the states* of $\mathcal{A}$,
- I is a subset of Q and $|I| = 1$, called the *set of initial states* of $\mathcal{A}$,
- F is a subset of Q, called the *set of final states* of $\mathcal{A}$,
- A is a finite set, called the *input alphabet* of $\mathcal{A}$,
- $\delta_{\mathcal{A}} : Q \times (A \cup \{b\}) \to Q$ is a partial function, called the *transition function* of $\mathcal{A}$

v) definition of a k-unbounded Register Machine $\mathcal{RM}_k$

A k-unbounded Register Machine is a k+3-tuple

$$\mathcal{RM}_k = (I, p_I, \underbrace{R_0, R_1, ..., R_k}_{\text{registers}}),$$

wherein
- for all $1 \leq i \leq k$, the R_i is called the *register i of the machine* and $R_i = \mathbb{N}$,
- r_i denotes the content of the register R_i
- I is the input tape. The input tape is where the program that will be executed by the machine is stored.
- $p_I = \mathbb{N}$, is the pointer of the input tape. The content of the pointer p_I, is denoted by r_I, and indicates the instruction of the program that is active.
- R_0 is the the Accumulator Register, $R_0 = \mathbb{N}^3$. The register R_0 consists of 3 parts: the high part R_{00}, the middle part R_{10} and the low part R_{20}. The content of each one of the parts of the register R_0 is denoted, respectively, by r_{00}, r_{10}, r_{20}, (r_{00}, r_{10}, r_{20}). A k-unbounded Register Machine has four instructions, Z, S, C and J, which will appear in the program through $\alpha(r_I)$, $\alpha(r_I) = Z$, $\alpha(r_I) = S$, $\alpha(r_I) = C$ or $\alpha(r_I) = J$.

The instruction Z is a 1-partial function, $Z : \mathbb{N} \longrightarrow \mathbb{N}$, where $Z(n)$, with $n = r_{20}$. If $n \neq 0$ then, $R_n \longleftarrow 0$ and $p_I \longleftarrow r_I + 1$, else $R_{20} \longleftarrow 0$ and $p_I \longleftarrow r_I + 1$.

The instruction S is a 1-partial function, $S : \mathbb{N} \longrightarrow \mathbb{N}$, where $S(n)$, with $n = r_{20}$. If $n \neq 0$, then $R_n \longleftarrow r_n + 1$ and $p_I \longleftarrow r_I + 1$, else $R_{20} \longleftarrow r_{20} + 1$ and $p_I \longleftarrow r_I + 1$.

The instruction C is a 2-partial function, $C : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, where $C(n, m)$ such that $n = r_{10}$ and $m = r_{20}$. $C(n, m)$. If $n = r_{10} \neq 0$ then, $R_{r_{10}} \leftarrow r_{r_{20}}$ and $p_I \longleftarrow p_I + 1$, else, If $n = r_{10} = 0$ then, $R_{20} \leftarrow r_{r_{20}}$ and $p_I \longleftarrow p_I + 1$.

The instruction J is a 3-partial function, $J : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, where $J(l, n, m)$ such that $l = r_{00}$, $n = r_{10}$ and $m = r_{20}$. $J(l, n, m)$ compares the content of the registers $R_n = R_{10}$ with $R_m = R_{20}$ if $r_{10} = r_{20}$ then, $p_I \longleftarrow r_{00}$ else $p_I \longleftarrow p_I + 1$. The content of the register R_{00}, r_{00} indicates the number of the instruction, in the sequence of instructions of the program which, in the case of $r_n = r_m$, should be the next instruction to be carried out.

4.2.2 Proving the Theorem 4.1

The proof of the theorem 4.1 will be given from the instantiation of each one of FCSs to FMs. The mathematical definitions of the FCSs, that are instantiated here as FMs, are in the previous section. To prove the theorem I begin to taking one FCS, without loss of generality, for each one of different kinds of FCSs that are defined above and rewrite it as a FM.

The proof of Theorem 4.1 is a constructive proof and is divided in five lemmas. Lemma 4.1, Lemma 4.2, 4.3, 4.4, 4.5 where respectively, and for each one of the FCSs enunciated in the Theorem 4.1, $CompM_B$, $CompM_R$, $ConfM$, $InstM$, VN_{Alg} are instantiated.

Lemma 4.1 *The component $CompM_B$ is instantiated for each one of the FCSs in Theorem 4.1.*

Proof 4.1 *I start by instantiating the set $CompM_B$ for the FCSs under study.*

i) The set of components of the k-Turing Machine is a set with 2k+5 elements,

$$CompM_B(\mathcal{T}) = \{Q, T_I, T_1, ..., T_k, T_O, p_I, p_1, ..., p_k, p_O\},$$

wherein
- Q is the set of the states
- T_I is the input tape, $T_1, ..., T_k$ are the processing tapes respectively the 1^{th}, 2^{nd}, ..., k^{th}, T_O is the output tape. Each tape consists of a concatenated sequence, infinite and countable, of squares (cells) from a first square.
- p_I is the pointer in the input tape T_I, and $p_1, ..., p_k$ are pointers, respectively, of the tapes $T_1, ..., T_k$. p_O is the pointer of the output tape T_O. Each pointer indicates the active square, which is the state of the component on the respective tape.
Each one of the components is the following set, $T_I = A^*\flat^w$, $T_1 = ... = T_k = \Gamma^*\flat^w$, $T_O = O^*\flat^w$ and $p_I = p_1 = ... = p_k = p_O = \mathbb{N}$.

ii) The set of the components of the Pushdown Automaton is a set with 4 elements,

$$CompM_B(\mathcal{AS}) = \{Q, T_I, p_I, P\},$$

wherein
- Q is the set of the states
- T_I is the input tape. The tape consist of a concatenated sequence, infinite and numerable, of squares from the first square.
- p_I the pointer of the input tape T_I indicates which is the active square and which is the state of the component T_I.
- P is called the stack. P is a temporary storage space. The capacity of storage, in storable characters, is infinite and countable.

Each one of the components is the following set, $T_I = A^*\flat^w$, $p_I = \mathbb{N}$ and $P = Z_0(\Gamma - Z_0)^*$.

iii) The set of the components of the Transducer is a set with 4 elements,

$$CompM_B(\mathcal{AT}) = \{Q, T_I, T_O, p_I, p_O\},$$

wherein
- Q is the set of the states
- T_I is the input tape and T_O is the output tape. Each tape consists of a concatenated sequence, infinite and countable, of squares from a first square.
- p_I, p_O are, respectively, the pointers of the input tape T_I and of the output tape T_O and indicates, for each one of the respectively tapes, which is the active square, which is the state of component, respectively, T_I and T_O.

Each one of the components is the following set, $T_I = A^*\flat^w$, $T_O = O^*\flat^w$ and $p_I = p_O = \mathbb{N}$.

iv) The set of the components of the Finite Automaton is a set of 3 elements,

$$CompM_B(\mathcal{A}) = \{Q, T_I, p_I\},$$

wherein
- Q is the set of the states
- T_I is the input tape. The tape consists of a concatenated sequence, infinite and countable, of squares from the first square.
- p_I is the pointer of the input tape T_I and indicates which is the active square, which is the state of component T_I.

Each one of the components is the following set, $T_I = A^*\mathfrak{b}^w$ and $p_I = \mathbb{N}$.

v) The components of the k-unbounded Register Machine[33]*, $\mathcal{RM}_k$, is a set with k+3 elements*

$$CompM_B(\mathcal{RM}_k) = \{I, p_I, R_0, R_1, R_2, ..., R_k\}$$

each one of the components is the following set, $I = \{S, Z, J, C\}^*.\mathfrak{b}^w$, $p_I = \mathbb{N}$, $R_0 = \mathbb{N}^3$, $R_1 ... = R_k = \mathbb{N}$.$\square$

Then, for the same machines, I will instantiate the set $CompM_R$. The elements that belong to the set $CompM_R$ are relations among the components of the FM. An element that belongs to $CompM_R$ establishes a concrete causal relation, inside the relation, among the components of the FM.

Lemma 4.2 *The component $CompM_R$ is instantiated for each one of the FCSs in Theorem 4.1.*

Proof 4.2 *i) The set $CompM_R$ of the k-Turing Machine*
$CompM_R = \{R_{\delta_{\mathcal{TM}}}, R_{\mu_{\mathcal{TM}}}\}$
the relation $R_{\delta_{\mathcal{TM}}}$ is a subset of the set
$(Q - F) \times A^*\mathfrak{b}^w \times (\Gamma^*\mathfrak{b}^w)^k \times \mathbb{N} \times \mathbb{N}^k \times Q \times (\Gamma^*\mathfrak{b}^w)^k \times \mathbb{N} \times \mathbb{N}^k$.

An element of $R_{\delta_{\mathcal{TM}}}$ is a tuple
$(q, u_1 a u_1' \mathfrak{b}^w, (\alpha_1 \gamma_1 \alpha_1' \mathfrak{b}^w, ..., \alpha_k \gamma_k \alpha_k' \mathfrak{b}^w), n_I, (n_1, ..., n_k),$
$q', (\alpha_1 \gamma_1' \alpha_1' \mathfrak{b}^w, ..., \alpha_k \gamma_k' \alpha_k' \mathfrak{b}^w), n_I', (n_1', ..., n_k')) \in$
$(Q - F) \times A^*\mathfrak{b}^w \times (\Gamma^*\mathfrak{b}^w)^k \times \mathbb{N} \times \mathbb{N}^k \times Q \times (\Gamma^*\mathfrak{b}^w)^k \times \mathbb{N} \times \mathbb{N}^k$,
wherein $n_I = \lfloor u_1 a \rfloor$, $a \in A \cup \{\mathfrak{b}\}$ (when $a = \mathfrak{b}$, then $u_1' = \epsilon$) and $n_i = \lfloor \alpha_i \gamma_i \rfloor$, $\gamma_i \in \Gamma \cup \{\mathfrak{b}\}$ (when $\gamma_i = \mathfrak{b}$, then $\alpha_i' = \epsilon$).

$$(q', (\gamma_1', ..., \gamma_k'), r_I, (r_1, ..., r_k)) \in$$
$$\delta(q, (u_1 a u_1' \mathfrak{b}^w)(n_I), ((\alpha_1 \gamma_1 \alpha_1' \mathfrak{b}^w)(n_1), ..., (\alpha_k \gamma_k \alpha_k' \mathfrak{b}^w)(n_k)),$$

[33]This is an unbounded Register Machine of k registers.

with $q' \in Q$, $\gamma_i' \in \Gamma \cup \{\flat\}$.

For $r_I = R$, $r_I = S$ and $r_I = L$ we have, respectively, $n_I' = n_I + 1$, $n_I' = n_I$ and $n_I' = n_I - 1$. In the case of each $1 \leq i \leq k$, wherein $r_i = R$,$r_i = S$ and $r_i = L$, we have, respectively, $n_i' = n_i + 1$, $n_i' = n_i$ and $n_i' = n_i - 1$.

The relation $R_{\mu_{\mathcal{TM}}}$ is a subset of

$$(Q - F) \times A^*\flat^w \times (\Gamma^*\flat^w)^k \times \mathbb{N} \times \mathbb{N}^k \times O^*\flat^w \times \mathbb{N}.$$

An element of $R_{\mu_{\mathcal{TM}}}$ is a tuple

$$(q, u_1 a u_1' \flat^w, (\alpha_1\gamma_1\alpha_1'\flat^w, ..., \alpha_k\gamma_k\alpha_k'\flat^w), n_I, (n_1, ..., n_k), \alpha_O o \flat^w, n_O') \in$$
$$(Q - F) \times A^*\flat^w \times (\Gamma^*\flat^w)^k \times \mathbb{N} \times \mathbb{N}^k \times \mathbb{N} \times O^*\flat^w \times \mathbb{N}$$

where $\alpha_O \in O^*, o \in O \cup \{\flat\}$ and by convention $|\flat|$=0.

$$(\Theta_O, r_O) \in \mu(q, (u\flat^w)(n_I), ((\alpha_1\gamma_1\alpha_1'\flat^w)(n_1), ..., (\alpha_k\gamma_k\alpha_k'\flat^w)(n_k)))$$

if $o = \flat$ (respectively, $o \neq \flat$), then $n_O' = |\alpha_O\rfloor + 1$ and $r_O = S$ (respectively, $n_O' = |\alpha_O o\rfloor + 1$ $r_O = R$).

ii) The set $CompM_R$ of the Pushdown Automaton

$CompM_R = \{R_{\delta_{\mathcal{AS}}}\}$

The relation $R_{\delta_{\mathcal{AS}}}$ is a subset of

$$Q \times A^*b^w \times \mathbb{N} \times \Gamma^* \times Q \times \mathbb{N} \times \Gamma^*$$

An element of $R_{\delta_{\mathcal{AS}}}$ is a tuple

$$(q, u_1 a u_1' \flat^w, n_I, Z_0\alpha_0\gamma, q', n_I + 1, Z_0\alpha_0\beta), \textit{ with}$$

$$(q', \beta) \in \delta_{\mathcal{AS}}(q, (u_1 a u_1' \flat^w)(n_I), \alpha),$$

wherein $\beta \in \Gamma^*$ and $n_I = |u_1 a\rfloor$ (if $a = \flat$, then $u_1' = \epsilon$). If $Z_0\alpha_0\gamma = Z_0$, then we have $Z_0\alpha_0\gamma = Z_0 = \epsilon$,$Z_0\alpha_0\gamma = Z_0$ or $Z_0\alpha_0\gamma = Z_0(\Gamma - Z_0)^+$.

iii) The set $CompM_R$ of the Transducer

$CompM_R = \{R_{\delta_{\mathcal{AT}}}, R_{\mu_{\mathcal{AT}}}\}$

The relation $R_{\delta_{\mathcal{AT}}}$ is a subset of

$$Q \times A^*b^w \times \mathbb{N} \times Q \times \mathbb{N}$$

An element of $R_{\delta_{\mathcal{AT}}}$ is a tuple

$$(q, u_1 a u_1' \flat^w, n_I, q', n_I + 1)$$

wherein $n_I = |u_1 a\rfloor$, $q' \in \delta(q, (u_1 a u_1' \flat^w)(n_I))$ (if $a = \flat$, then $u_1' = \epsilon$).

The relation $R_{\mu_{\mathcal{AT}}}$ is a subset of

$$Q \times A^*b^w \times \mathbb{N} \times O^*\flat^w \times \mathbb{N}$$

An element of $R_{\mu_{\mathcal{AT}}}$ is a tuple

$$(q, u_1 a u_1' \flat^w, n_I, v o \flat^w, n_O'),$$

wherein $n_I = \lfloor u_1 a \rfloor$, $n_O = \lfloor v b \rfloor$, $n_O' = n_O + 1$ *and* $o \in \mu(q, (u_1 a u_1' \flat^w)(n_I))$ *(if* $a = \flat$, *then* $u_1' = \epsilon$*).*

iv) The set $CompM_R$ *of the Finite Automaton*

$CompM_R = \{R_{\delta_{\mathcal{A}}}\}$,
$R_{\delta_{\mathcal{A}}} = \{(q, u_1 a u_1' \flat^w, n_I, q', n_I + 1) \in Q \times A^* \flat^w \times \mathbb{N} \times Q \times \mathbb{N} : q' \in \delta_{\mathcal{A}}(q, (u_1 a u_1' \flat^w)(n_I))\}$,
if $a = \flat$, *we have* $u_1' = \epsilon$.

v) The set $CompM_R$ *of a k-unbounded Register Machine*

$CompM_R = \{R_{\mathcal{S}}, R_{\mathcal{Z}}, R_{\mathcal{C}}, R_{\mathcal{J}}\}$
The relation R_S *is a subset of*[2]

$$I \times \mathbb{N} \times (\mathbb{N})^3 \times \mathbb{N}^k \times I \times \mathbb{N} \times \mathbb{N}^3 \times \mathbb{N}^k$$

An element of R_S *is a tuple*

$$(\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k), \alpha, r_I', (r_{00}, r_{10}, r_{20}'), (r_1', ..., r_k')),$$

wherein $\alpha(r_I) = S$, $S(r_{20})$, $r_I' = r_I + 1$. *For* $1 \leq i \leq k$, *if* $i = r_{20}$, *we have* $r_i' = r_i + 1$ *and, if* $i \neq r_{20}$, *then* $r_i' = r_i$. *For the case where* $r_{20} = 0$, *we have* $r_{20}' = r_{20} + 1$ *and, if* $r_{20} \neq 0$, *then* $r_{20}' = r_{20}$.

The relation R_Z *is a subset of*

$$I \times \mathbb{N} \times (\mathbb{N})^3 \times \mathbb{N}^k \times I \times \mathbb{N} \times \mathbb{N}^3 \times \mathbb{N}^k$$

An element of R_Z *is a tuple*

$$(\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k), \alpha, r_I', (r_{00}, r_{10}, r_{20}'), (r_1', ..., r_k')),$$

wherein $\alpha(r_I) = Z$, $Z(r_{20})$, $r_I' = r_I + 1$. *For* $1 \leq i \leq k$, *if* $i = r_{20}$, *we have* $r_i' = 0$ *and, if* $i \neq r_{20}$, *then* $r_i' = r_i$. *For the case where* $r_{20} = 0$, *we have* $r_{20}' = 0$ *and, if* $r_{20} \neq 0$, *then* $r_{20}' = r_{20}$.

The relation R_C *is a subset of*

$$I \times \mathbb{N} \times (\mathbb{N})^3 \times \mathbb{N}^k \times I \times \mathbb{N} \times \mathbb{N}^3 \times \mathbb{N}^k$$

An element of R_C *is a tuple*

$$(\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k), \alpha, r_I', (r_{00}, r_{10}, r_{20}'), (r_1', ..., r_k')),$$

wherein $\alpha(r_I) = C$, $C(r_{10}, r_{20})$, $r_I' = r_I + 1$. *For* $1 \leq i \leq k$, *if* $i = r_{10}$, *we have* $r_i' = r_{r_{20}}$ *and, if* $i \neq r_{10}$, *then* $r_i' = r_i$. *For the case where* $r_{10} = 0$, *we have* $r_{20}' = r_{r_{20}}$ *and, if* $r_{10} \neq 0$, *then* $r_{20}' = r_{20}$.

The relation R_J *is a subset of*

[2] $r_0 = (r_{00}, r_{10}, r_{20})$

$$I \times \mathbb{N} \times (\mathbb{N})^3 \times \mathbb{N}^k \times I \times \mathbb{N} \times \mathbb{N}^3 \times \mathbb{N}^k$$

An element of R_J is a tuple

$$(\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k), \alpha, r'_I, (r'_{00}, r_{10}, r_{20}), (r_1, ..., r_k)),$$

wherein $\alpha(r_I) = J$, $J(r_{00}, r_{10}, r_{20})$. Se $r_{r_{10}} = r_{r_{20}}$, then $r'_I = r_{00}$ and if $r_{r_{10}} \neq r_{r_{20}}$, then $r'_I = r_I + 1$.□

Lemma 4.3 *The component $ConfM$ is instantiated for each one of the FCSs in Theorem 4.1.*

Proof 4.3 *The idea that should be associated to a configuration of a machine is that it must indicate the state of each one of its components.*

i) The set of configurations of machine of the k-Turing Machine is a subset of the set

$$Q \times A^*\mathrm{b}^w \times (\Gamma^*\mathrm{b}^w)^k \times O^*\mathrm{b}^w \times \mathbb{N} \times (\mathbb{N})^k \times \mathbb{N}$$

A configuration of machine is an element

$$(q, u\mathrm{b}^w, (\alpha_1\mathrm{b}^w, ..., \alpha_k\mathrm{b}^w), v\mathrm{b}^w, n_I, (n_1, ..., n_k), |v| + 1)$$

initial configurations: $(q, u\mathrm{b}^w, (\alpha_1\mathrm{b}^w, ..., \alpha_k\mathrm{b}^w), v\mathrm{b}^w, 1, (1, ..., 1), 1)$ com $q \in I$
final configurations: $(q, u\mathrm{b}^w, (\alpha_1\mathrm{b}^w, ..., \alpha_k\mathrm{b}^w), v\mathrm{b}^w, |u| + 1, (n_1, ..., n_k), n_O)$ com $q \in F$

ii) The set of configurations of machine of the Pushdown Automaton is a subset of the set

$$Q \times A^*\mathrm{b}^w \times \mathbb{N} \times Z_0(\Gamma - \{Z_0\})^*$$

A configuration of machine is an element

$$(q, u\mathrm{b}^w, n_0, Z_0\alpha_P)$$

with $\alpha_P \in (\Gamma - \{Z_0\})^*$.
initial configurations: $(q, u\mathrm{b}^w, 1, Z_0)$ with $q \in I$
final configurations: $(q, u\mathrm{b}^w, |u| + 1, Z_0\alpha_P)$ with $q \in F$, $v \in \Gamma^*$

iii) The set of configurations of machine of the Transducer is a subset of

$$Q \times A^*\mathrm{b}^w \times O^*\mathrm{b}^w \times \mathbb{N} \times \mathbb{N}.$$

A configuration of machine is an element

$$(q, u\mathrm{b}^w, v\mathrm{b}^w, n_I, n_O)$$

initial configurations: $(q, u\mathrm{b}^w, v\mathrm{b}^w, 1, 1)$ com $q \in I$
final configurations: $(q, u\mathrm{b}^w, v\mathrm{b}^w, |u| + 1, |v| + 1)$ com $q \in F$

iv) The set of configurations of machine of the Finite Automaton is a subset of

$$Q \times A^* b^w \times \mathbb{N}$$

A configuration of machine is an element,

$$(q, u b^w, n)$$

initial configurations: $(q, u b^w, 1)$ *com* $q \in I$
final configurations: $(q, u b^w, |u| + 1)$ *com* $q \in F$

v) The set of configurations of machine of the k-unbounded Register Machine is a subset of

$$I \times \mathbb{N} \times \mathbb{N}^3 \times \mathbb{N}^k$$

A configuration of machine is an element

$$(\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k))$$

initial configurations: $(\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k))$ *with* $r_I = 1$
final configurations: $(\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k))$ *with* $\alpha(r_I) = `.'$ $\square$

Lemma 4.4 *The component* $InstM$ *is instantiated for each one of the FCSs in Theorem 4.1.*

Proof 4.4 *i) Instructions of the k-Turing Machine*
$I_\delta : ConfM - ConfM_f \longrightarrow ConfM$, $c \longmapsto c_\delta = I_\delta(c)$ *such that* $c_\delta(T_I) = c(T_I)$, $c_\delta(T_O) = c(T_O)$, $c_\delta(p_O) = c(p_O)$ *and*

$$(c(Q), c(T_I), c(T_1), ..., c(T_k), c(p_I), c(p_1), ..., c(p_k),$$
$$c_\delta(Q), c_\delta(T_1), ..., c_\delta(T_k), c_\delta(p_I), c_\delta(p_1), ..., c_\delta(p_k)) \in R_{\delta_{\mathcal{TM}}}$$

$I_\mu : ConfM - ConfM_f \longrightarrow ConfM$, $c \longmapsto c_\mu = I_\mu(c)$ *such that* $c_\mu(Q) = c(Q)$, $c_\mu(T_I) = c(T_I)$, $c_\mu(p_I) = c(p_I)$, $c_\mu(p_O) = c(p_O)$, *and for* $1 \leq i \leq k$ $c_\mu(T_i) = c(T_i)$, $c_\mu(p_i) = c(p_i)$ *and*

$$(c(Q), c(T_I), c(T_1), ..., c(T_k), c(p_I), c(p_1), ..., c(p_k),$$
$$c_\mu(T_O), c_\mu(p_O)) \in R_{\mu_{\mathcal{TM}}}$$

$$I : ConfM - ConfM_f \times ConfM \longrightarrow ConfM, \ (c_\delta, c_\mu) \longmapsto I(c_\delta, c_\mu) = c',$$

wherein $c'(Q) = c_\delta(Q)$, $c'(T_I) = c_\delta(T_I)$, *for all* $1 \leq i \leq k$ $c'(T_i) = c_\delta(T_i)$, $c'(p_i) = c_\delta(p_i)$, $c'(T_O) = c_\mu(T_O)$ *and* $c'(p_O) = c_\mu(p_O)$.

ii) Instruction of the Pushdown Automaton
$I_\delta : ConfM - ConfM_f \longrightarrow ConfM$, $c \longmapsto c_\delta = I_\delta(c)$ *such that* $c_\delta(T_I) = c(T_I)$ *and*

$$(c(Q), c(T_I), c(p_I), c(P), c_\delta(Q), c_\delta(p_I), c_\delta(P)) \in R_{\delta_{\mathcal{AS}}}$$

iii) Instructions of the Transducer
$I_\delta : ConfM - ConfM_f \longrightarrow ConfM$, $c \longmapsto c_\delta = I_\delta(c)$ *such that* $c_\delta(T_I) = c(T_I)$, $c_\delta(T_O) = c(T_O)$, $c_\delta(p_O) = c(p_O)$ *and*

$$(c(Q), c(T_I), c(p_I), c_\delta(Q), c_\delta(p_I)) \in R_{\delta_{\mathcal{AT}}}$$

$I_\mu : ConfM - ConfM_f \longrightarrow ConfM$, $c \longmapsto c_\mu \in I_\mu(c)$ *such that* $c_\mu(Q) = c(Q)$, $c_\mu(p_I) = c(p_I)$, $c_\mu(T_I) = c(T_I)$ *and*

$$(c(Q), c(T_I), c(p_I), c_\mu(T_O), c_\mu(p_O)) \in R_{\mu_{\mathcal{AT}}}$$

$$I : ConfM - ConfM_f \times ConfM \longrightarrow ConfM,\ (c_\delta, c_\mu) \longmapsto I(c_\delta, c_\mu) = c'$$

such that $c'(Q) = c_\delta(Q)$, $c'(T_I) = c_\delta(T_I)$, $c'(p_I) = c_\delta(p_I)$, $c'(T_O) = c_\mu(T_O)$ *and* $c'(p_O) = c_\mu(p_O)$.

iv) Instructions of the Finite Automata
$I_\delta : ConfM - ConfM_f \longrightarrow ConfM$, $c \longmapsto c_\delta\ = I_\delta(c)$ *such that* $c_\delta(T_I) = c(T_I)$

$$(c(Q), c(T_I), c(p_I), c_\delta(Q), c_\delta(p_I)) \in R_{\delta_{\mathcal{A}}}$$

v) Instructions of the k-unbounded Register Machines

The instructions of the machine for the k-unbounded Register Machine are:
- $S(n)$ *increments a value at the content of the register,* R_n, $R_n \leftarrow r_n + 1$, *and the pointer of the input tape goes to the right. When* $n = 0$, *the operation is carried out in the accumulator* R_0, *in the part* R_{20}, $R_{20} \leftarrow r_{20} + 1$ *of the register.*
- $Z(n)$ *puts the value 0,* $R_n \leftarrow 0$, *in the register* R_n *and the pointer of the input tape goes to the right. When* $n = 0$, *the operation is carried out in the accumulator* R_0, *in the part* R_{20}, $R_{20} \leftarrow 0$ *of the register.*
- $C(n, m)$ *copies the content of the register* R_m *for the register* R_n, $R_n \leftarrow r_m$, *and the pointer of the input tape goes to the right. When* $n = 0$, *the content of the register* R_m *is copied for the accumulator* R_0, *for the part* R_{20}, $R_{20} \leftarrow r_m$ *of the register.*
- $J(l, n, m)$, *if the content of the register* R_n *is equal to the content of the register* R_m ($r_n = r_m$), *then the pointer of the input tape should be put at the position* l *in the sequence of instructions. Otherwise it should be put at the following instruction.*

The instructions of the FCS considered are 4, I_S,I_Z,I_C *and* I_J. *For a configuration* c,

$$c = (\alpha, r_I, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k)),$$

wherein, if $\alpha(r_I) = S$ *(respectively,* $\alpha(r_I) = Z$, $\alpha(r_I) = C$, $\alpha(r_I) = J$*),* $S(r_{20})$ *(respectively,* $Z(r_{20})$, $C(r_{10}, r_{20})$, $J(l, r_{10}, r_{20})$*), then* $I_S(c) = c_S$ *(respectively,* $I_Z(c) = c_Z$, $I_C(c) = c_C$, $I_J(c) = c_J$*)*[34]. $c_S = S(r_{20})$ *(respectively,* $c_Z = Z(r_{20})$, $c_C = C(r_{10}, r_{20})$, $c_J = J(r_{00}, r_{10}, r_{20})$*).*$\square$.

Next I am going to describe Von Neumann's Algorithm, although only parts 2 and 3, which, by abuse of language, I still call Von Neumann's Algorithm, for each one of the machines under discussion.[35]

[34] $\alpha(r_I)$ with $\alpha = I_1 I_2 ... I_t$ and $I_j \in \{Z_0, S, C, J, `.'\}$ indicates that $\alpha(1) = i_1, \alpha(2) = i_2, ..., \alpha(t) = i_t$

[35] The Cycles of Von Neumann, CVNs, described here are only the CNV for deterministic machines. By ordination of the symbols involved in the definition 7 of a FM, it is possible to create a partial relation order, the quasi lexicographic order, and to use that to create Von Neumann algorithms for machines that are not deterministic. However, that is outside of the scope of this article and will be presented in a later paper.

Lemma 4.5 *The VN_{Alg} can be described for each one of the FCSs in Theorem 4.1.*

Proof 4.5 *The proof of this Lemma consists in giving the CVN for each one of the FCSs.*

Algorithm 2 CVN.i) The CVN of a deterministic k-Turing Machine

```
i.0) do
i.1) while($\tau_{input}$ == empty) // waiting to receive an input task, $\tau_{input}$, to perform
i.2) load the input task, $\tau_{input}$, on the input tape
i.3) set the machine on the adequate initial configuration, $c_0$, $c \leftarrow c_0$.
i.4) do
i.5) read the state of the machine and obtain the configuration, $c$, that is active.
i.6) carry out the instruction $I_\delta(c)$ to obtain the configuration $c_\delta$
i.7) carry out the instruction $I_\mu(c)$ to obtain the configuration $c_\mu$
i.8) carry out the instruction $I(c_\delta, c_\mu)$ to obtain the configuration $c'$
i.9) put the configuration $c'$ in the place of the configuration $c$, $c \leftarrow c'$
i.10) while($not(c(Q) \in F$ and $c(T_I)(p_I) == \flat$))
```

From the previous Von Neumann algorithm, in the case of k-Turing Machines, the cycle of Von Neumann is translated in terms of the execution of tasks in the following steps of computation.

$$c \vdash c_\delta \vdash \{c_\delta, c_\mu\} \vdash c', \textit{ where } c_\delta, c_\mu \in ConfM : c_\delta \in I_\delta(c), c_\mu \in I_\mu(c) \textit{ e } c' \in I(c_\delta, c_\mu)$$

Algorithm 3 CVN.ii) CVN of a deterministic Pushdown Automaton recognized by final states

```
ii.0) do
ii.1) while($\tau_{input}$ == empty) // waiting to receive an input task, $\tau_{input}$, to perform
ii.2) load the input task, $\tau_{input}$, on the input tape
ii.3) set the machine on the adequate initial configuration, $c_0$, $c \leftarrow c_0$.
ii.4) do
ii.5) read the state of the machine and obtain the configuration, $c$, that is active.
ii.6) carry out the instruction $I_\delta(c)$ to obtain the configuration $c_\delta$
ii.7) $c \longleftarrow c_\delta$
ii.8) while($not(c(Q) \in F$ and $c(T_I)(p_I) == \flat$))
```

In the case of the Pushdown Automata, the execution of CVN is described in the above steps. Additionally,

$$c \vdash c' \textit{ if and only if } c' \in I_\delta(c)$$

Algorithm 4 CVN.iii) CVN of a deterministic Transducer

```
iii.0) do
iii.1) while($\tau_{input}$ == empty) // waiting to receive an input task, $\tau_{input}$, to perform
iii.2) load the input task, $\tau_{input}$, on the input tape
iii.3) set the machine on the adequate initial configuration, $c_0$, $c \leftarrow c_0$.
iii.4) do
iii.5) read the state of the machine and obtain the configuration, $c$, that is active.
iii.6) carry out the instruction $I_\delta(c)$ to obtain the configuration $c_\delta$
iii.7) carry out the instruction $I_\mu(c)$ to obtain the configuration $c_\mu$
iii.8) carry out the instruction $I(c_\delta, c_\mu)$ to obtain the configuration $c'$
iii.9) put the configuration $c'$ in the place of the configuration $c$, $c \leftarrow c'$
iii.10) while($not(c(Q) \in F$ and $c(T_I)(p_I) == \flat$))
```

In the case of the transducers, the execution of the CVN is translated in the following computational steps:

$$c \vdash c_\delta \vdash \{c_\delta, c_\mu\} \vdash c', \text{ where } c_\delta, c_\mu \in Conf M \text{ such that}$$
$$c_\delta \in I_\delta(c), c_\mu \in I_\mu(c) \text{ e } c' \in I(c_\delta, c_\mu)$$

Algorithm 5 CVN.iv) CVN of a Finite Automaton

```
iv.0) do
iv.1) while($\tau_{input}$ == empty) // waiting to receive an input task, $\tau_{input}$, to perform
iv.2) load the input task, $\tau_{input}$, on the input tape
iv.3) put the machine on the adequate initial configuration, $c_0$, $c \leftarrow c_0$.
iv.4) do
iv.5) read the state of the machine and obtain the configuration, $c$, that is active.
iv.6) carry out the instruction $I_\delta(c)$ to obtain the configuration $c_\delta$
iv.7) $c \leftarrow c_\delta$
iv.8) while($not(c(Q) \in F$ and $c(T_I)(p_I) == \flat$))
```

In the case of Finite Automata, the execution of CVN is translated in the following computation steps:

$$c \vdash c' \text{ if and only if } c' \in I_\delta(c)$$

Algorithm 6 CVN.v) CVN of an k-unbounded Register Machine

```
v.0) do
v.1) while($program$ == empty) // waiting to receive an input task, $program = I_1 I_2 ... I_k$ where each $I_j$ is an instruction of the machine
v.2) load the program, $\tau_{input}$, on the input tape
v.3) put the machine on the adequate initial configuration, $c_0$, $c \leftarrow c_0$.
v.4) do
v.5) read the state of the machine and obtain the configuration, $c$, that is active. $c = (program, r_I, r_0, r_1, ..., r_k)$
v.6) carry out the instruction of the machine that is pointed in the input tape, $program(r_I)$, for the accumulator register $r_{20}$
v.7) decode the execution of an instruction of the machine $program(r_I)$
v.8) carry out the instruction of machine $program(r_I)$ and obtain a new configuration $c'$, $c' = (program(r'_I), r'_0, r'_1, ..., r'_k)$
v.9) $c \longleftarrow c'$
v.10) while($program(r_I)! =$ “.”)
```

In the case of the Register Machines, $c \vdash c'$ results from an execution of one of the instructions S, Z, C and J.□.

Theorem4.1 *The deterministic machines: k-Turing Machines, Transducers, Push-down Automata, Finite Automata and k-unbounded Register Machines are FMs.□.*

Proof 4.6 ***Proof of the Theorem 4.1*** *Of the Lemmas 1, 2, 3, 4 and 5 it is proved that the FCSs referred to are FMs.□*

4.2.3 Proving the Theorem 4.2

Theorem 4.2 *Recurrent Neural Networks are FMs*

Proof of the Theorem 4.2

Proof 4.7 *Let $\mathcal{N}$ be as defined in [SiS94], a Recurrent Neuronal Network. Let $\mathcal{N}$ be instantiate as a FM. In $\mathcal{N}$ neurons, Neu, are defined as*

$$Neu = \{\vec{n} = (n_1, ..., n_N) : n_i \in \mathbb{R}\},$$

a family of vectors of input, U, where each one has M components,

$$U = \{\vec{u}(t) = (u_1(t), u_2(t), ..., u_M(t)) : \ e\ t \in \mathbb{N}, u_i(t) \in \mathbb{R}\},$$

and a component $T = \mathbb{N}$ that is seen as the interaction

---

$CompM_B = \{\vec{U}, Neu, T\}$

$CompM_R = \{R_U, R_{Neu}\}$ such that

$$R_U = \{(\vec{u}, \vec{n}, t, \vec{u'}, t+1) : \vec{u} = \vec{u}(t), \vec{u'} = \vec{u'}(t+1) \in U \text{ and } \vec{n} \in Neu, t \in T\},$$

$$R_U \subseteq U \times Neu \times T \times U \times T$$

$$R_{Neu} = \{(\vec{u}, \vec{n}, t, \vec{n'}) : \vec{u} = \vec{u}(t) \in U, \vec{n}, \vec{n'} = (n'_1, ..., n'_M) \in Neu\},$$

$$R_{Neu} \subseteq U \times Neu \times T \times Neu$$

$ConfM = \{(\vec{u}(t), \vec{n}, t) : \vec{u}(t) \in U, \vec{n} \in Neu, t \in T\} \subseteq U \times Neu \times T$
$ConfM_i = \{(\vec{u}(t), \vec{n}, 0) : \vec{u}(0) \in U, \vec{n} \in Neu\}$
$ConfM_f = \{(\vec{u}(t), \vec{n}, t_f) : \vec{u}(t_f) \in U, \vec{n} \in Neu\}$, *for some* $t_f \in T$

$(\vec{u}(t+1), \vec{n'}, t+1) \in Inst(\vec{u}(t), \vec{n}, t)$, *such that*
$\vec{n'} = (n'_1, ..., n'_N)$ *with* $n'_i = c(\sum_{i=1}^{N} a_{ij} n_j + \sum_{j=1}^{M} b_{ij} u(t) + c_i)$ *and such that*

$$\sigma(x) = \begin{cases} 0 & \textit{if } x < 0, \\ x & \textit{if } x \leq 0 \leq 1, \\ 1 & \textit{if } x > 1. \end{cases}$$

$\square$.

4.2.4 Tasks and performed Languages of the FCSs

After to prove in Theorem 4.1, that the FCSs are FMs, I seek a description of the tasks and the language performed for each one of the FCSs seen as FMs. To do that, I establish a relation between the notion of the language performed by each one of the FCSs, seen only as FCSs, and the notion of the language performed by a FM.

i) A task carried out, τ, for the k-Turing Machine, is a pair $\tau = (c, c')$, such that

$$c = (q, u\flat^w, \alpha_1\flat^w, ..., \alpha_k\flat^w, \flat^w, 1, 1, ..., 1, 1), \text{ and}$$
$$c' = (q', u\flat^w, \alpha'_1\flat^w, ..., \alpha'_k\flat^w, v\flat^w, n'_I, n'_1, ..., n'_k, n'_O);\ n'_I = |u\rfloor + 1,\ n'_O = |v\rfloor + 1,$$

with $c \in ConfM_i$ and $c' \in ConfM_f$. To simplify the notation, in the formalism presented, I denote the task above by the notation (u,v). That simplification is possible because I am able to define a correspondence $\phi : L(\mathcal{TM}) \longrightarrow A^* \times O^*$ where, for each (c,c') such as denoted above, $\phi(c,c') = (u,v)$ and, because this correspondence, in deterministic Turing Machines, ϕ is injective. The simplification of the notation allows to use the usual notation of a word recognized by k-Turing Machines as a task performed in FMs.

ii) The task carried out, τ, for Pushdown Automata, is a pair $\tau = (c,c')$, such that

$$c = (q, u\flat^w, 1, Z_0), \text{ and}$$
$$c' = (q', u\flat^w, n'_0, Z_0\alpha'_P),\ n'_0 = |u\rfloor + 1$$

with $\alpha'_P \in (\Gamma - Z_0)^*$ and $c \in ConfM_i$ and $c' \in ConfM_f$. To simplify the notation, in the formalism presented, I denote the task referred to above by the notation u. This simplification is possible because I am able to define a correspondence $\phi : L(\mathcal{AS}) \longrightarrow Z_0A^*$, wherein for (c,c') such as denoted above, $\phi(c,c') = u$ because this correspondence, in deterministic Pushdown Automata, ϕ is injective. The simplification of the notation allows to use the usual notation of a word recognized by Pushdown Automata as a task performed in FMs.

iii) The task carried out, τ, by a Transducer, is a pair $\tau = (c,c')$, such that:

$$c = (q, u\flat^w, \flat^w, 1, 1), \text{ and}$$
$$c' = (q', u\flat^w, v\flat^w, n'_I, n'_O);\ n'_I = |u\rfloor + 1,\ n'_O = |v\rfloor + 1$$

with $c \in ConfM_i$ and $c' \in ConfM_f$. To simplify the notation, in the formalism presented, I denote the task above by the notation (u,v). The simplification of the notation is possible because I am able to define a correspondence $\phi : L(\mathcal{T}) \longrightarrow A^* \times O^*$ such that for (c,c') as denoted above, $\phi(c,c') = (u,v)$ because this correspondence, in deterministic Machines, ϕ is injective. The simplification of the notation allows to use the usual notation of a word recognized by Transducers as a task performed in FMs.

iv) A task carried out, τ, for the Finite Automaton is a pair $\tau = (c,c')$, such that

$$c = (q, u\flat^w, 1), \text{ and}$$
$$c' = (q', u\flat^w, n');\ n' = |u\rfloor + 1,$$

with $c \in ConfM_i$ and $c' \in ConfM_f$. To simplify, I use the notation u. This simplification is possible because we are able to define a correspondence $\phi : L(\mathcal{T}) \longrightarrow A^*$ such that, for (c,c') such as denoted above, $\phi(c,c') = u$ and because this correspondence, in deterministic Machines, ϕ is injective. The simplification of the notation allows to use the usual notation of a word recognized by Finite Automaton as a task performed in FMs.

v) A task carried out, τ, for the k-unbounded Register Machine, is a pair $\tau = (c, c')$, such that

$$c = (\alpha, 1, (r_{00}, r_{10}, r_{20}), (r_1, ..., r_k)), \text{ and}$$
$$c' = (\alpha, r_I', (r_{00}', r_{10}', r_{20}'), (r_1', ..., r_k')),$$

such that $\alpha(r_I') =$ "."

4.3 A Software for Simulate Formal Computational Systems

In this subsection I present the software that I developed and that allows to generate universes and simulate computational systems.

The generation of universes consists in to define an alphabet, the letters, and the work words. That words are all the words, of an alphabet, with a length inferior at a certain n (See figure 12).

The simulation of the behavior of a computational system consisr in the computational system recognize the words that are generated by the generator of universes (See figure 11). This software is denoted by GU_SFM as an acronym of Generator of Universes and Simulator of Formal Machines. Now, I am going to describe the interface of the GU_SFM[36].

The interface has 4 tabs; file tab, configurations tab, Partial Orders tab and Help tab. In following I describe each one of them (See figure 10). In the interface can be found three file text boxes that are setting in the configuration and in Partial Order tabs, after to click the button SET. After all of this, it is possible to generate all the words of the universe, respecting the ordination given by the partial order. This generation is made in finite mode, it generates all the words of the universe and the process ends, or in continuous mode, making a continuous cycle that goes from the simplest words of the universe to the words of greater length and returning from these to the simplest. If the option is to develop the universe in finite generation is possible yet to compare two words of the universe. The software says between two words which one is the smallest and which is the largest.

[36]download it from `www.ipg.pt\user\~pavieira\private\sw\GU_SFM`

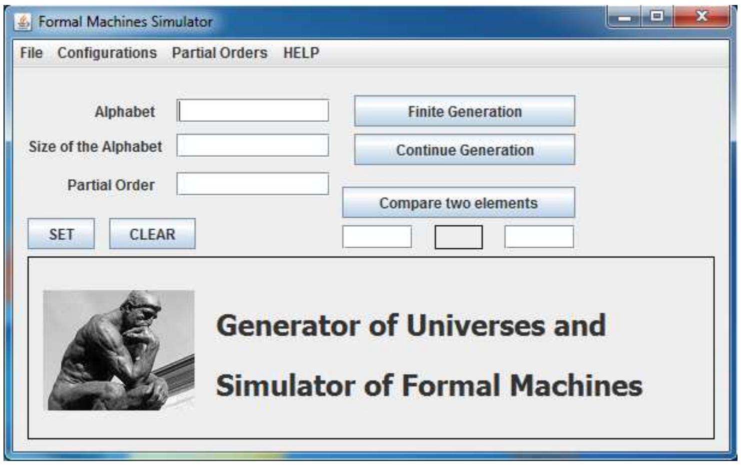

Formal Machines Simulator
File Configurations Partial Orders HELP
Alphabet
Size of the Alphabet
Partial Order
SET
CLEAR
Finite Generation
Continue Generation
Compare two elements
Generator of Universes and
Simulator of Formal Machines

Figure 10: The interface of the Generator of Universes and Simulator of Formal Machines (GU_SFM)

The file tab has two subtabs called Computational Models and Exit. In the computational system subtab is possible to choose one of the following several computational models, six of it, namely Turing Machines, Pushdown Automata, Transducer, Finite Automata, Unbounded Register Machine and Formal Machines (FMs). The Exit subtab is a subtab to close the software.

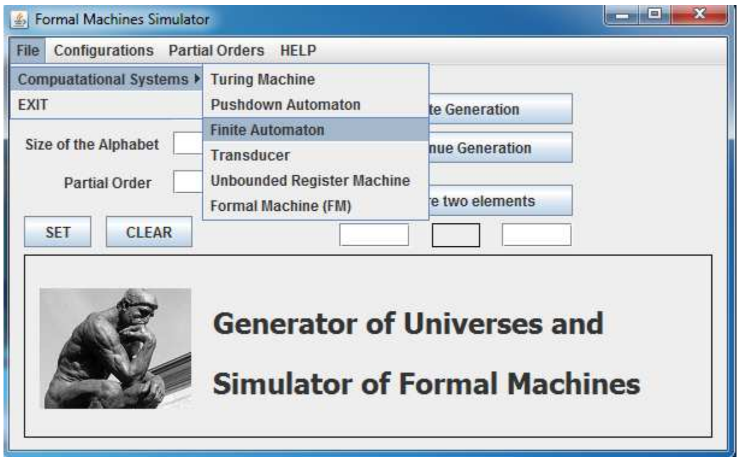


Figure 11: The tab of file of the GU_SFM

Now in the next four figures (figures 12, 13, 14, 15) can be seen the different uses of the configuration tab. In the figure 12 can be seen where is setting the letters of the alphabet, in figure 13 is where is set the size of the universe up to 20. The size of the universe is the maximum length possible that a word of the universe can have. In the subtab folder, figure 14 and figure 15, is for browse the path of the files with which the software works.

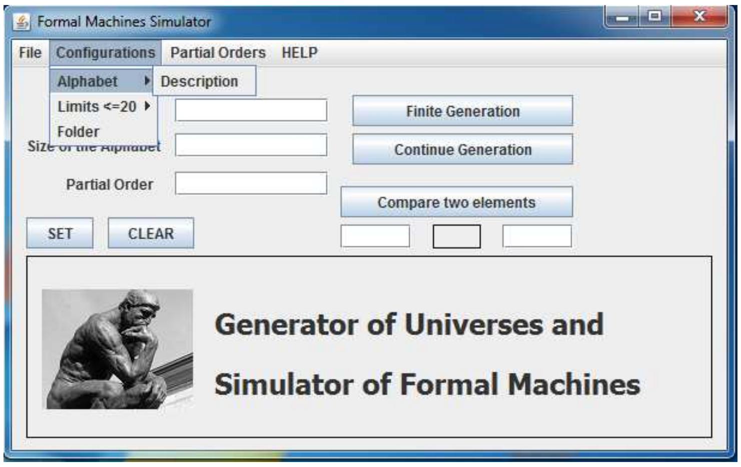


Figure 12: The Configurations tab of the GU_SFM, the alphabet

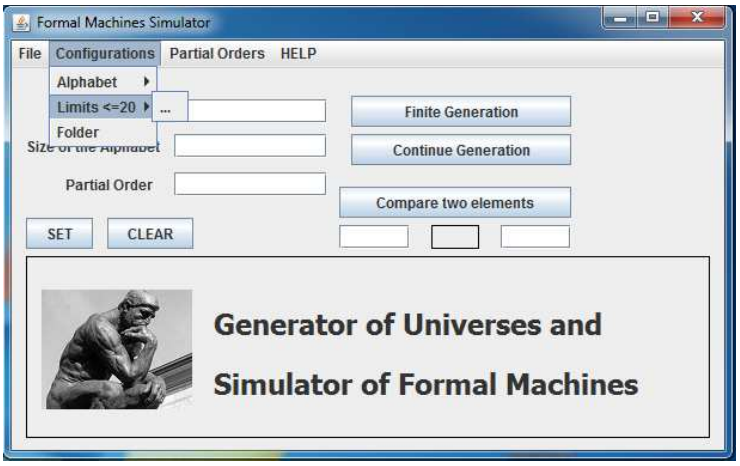


Figure 13: The Configurations tab of the GU_SFM, the Limit<21

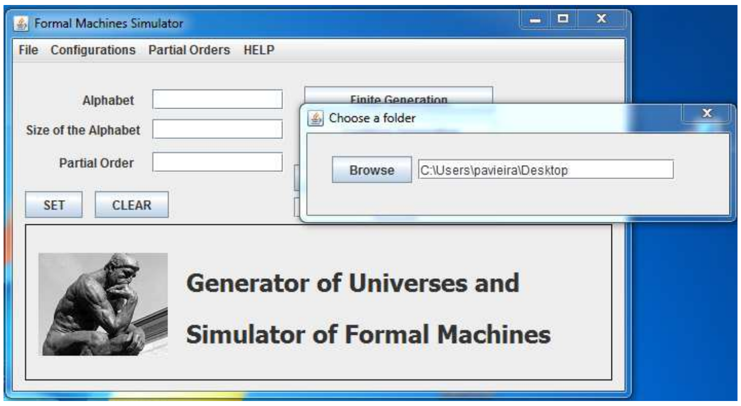


Figure 14: The Configurations tab of the GU_SFM, browse a path

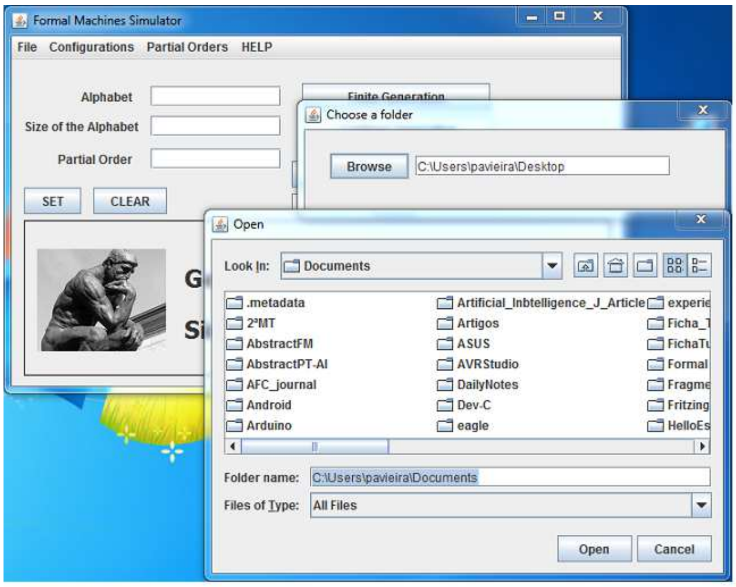


Figure 15: The Configurations tab of the GU_SFM, choosing a folder

Now I talk about the Partial Orders tab. In this tab is possible to choose several partial orders such that, a quasi lexicographic and co-lexicographic orders. For alphabets with two elements respectively the qlexicographicM2 and Co-lexicographicM2, and for alphabets with more two elements respectively qlexicographic and Co-qlexicographic. Another partial order that is possible to choose is one whose the words are propositions of a logic whose atomic terms are the letters of the alphabet. In this partial order called Propositional Logic is possible to choose between an universe where the propositions (the words) are in Formal Normal Conjunctive, FNC, or in Formal Normal Disjunctive, FND. This can be seen in figure 16.

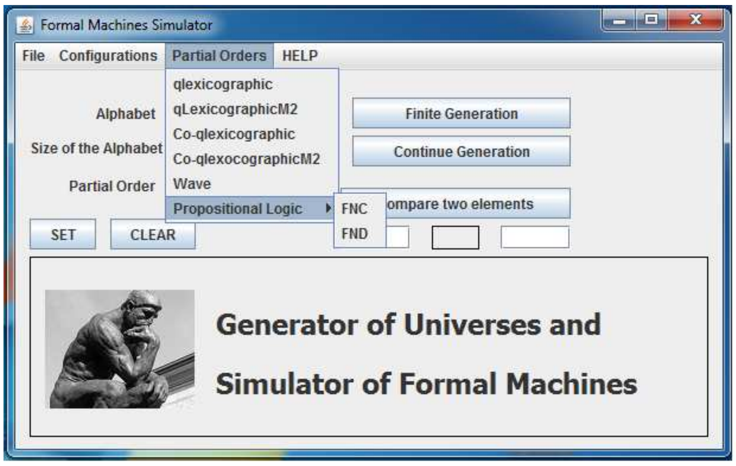


Figure 16: The Partial Orders tab of the GU_SFM, Propositional Logic

The next figure is the Help tab. This tab has 3 subtabs, the subtab About, License and How To . In subtab About can be seen the authors of the software, in subtab License can be seen what type of license is associated with the software and in the subtab HOW TO can be consulted an help for to know how to use the software

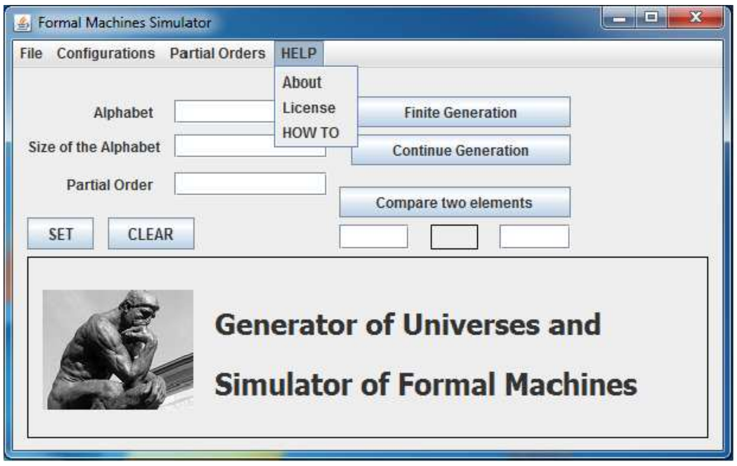


Figure 17: The HELP tab of the GU_SFM

In the following four next figures, the figures 18, 19, 20 and 21, are showed the different necessary steps to choose a Finite Automaton as computational model. The figure 18 shows the subtab where is possible to choose one of the computational models, the figure 19 is showed the computational model chosen, the Finite Automata. In figures 20 and 21 it can be seen respectively the explanation of the parameters of the Finite Automata and the interface of one of the computational systems.

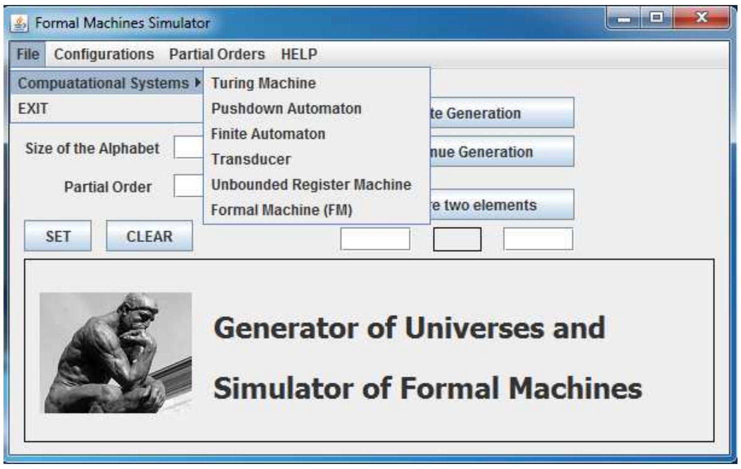


Figure 18: In the file tab the GU_SFM, choosing a computational model

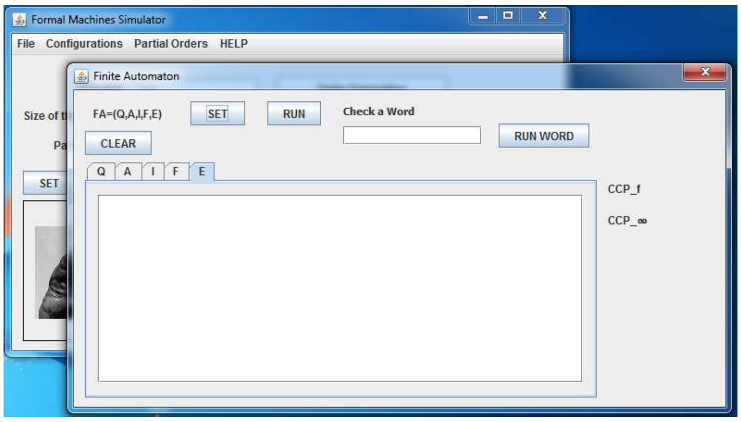
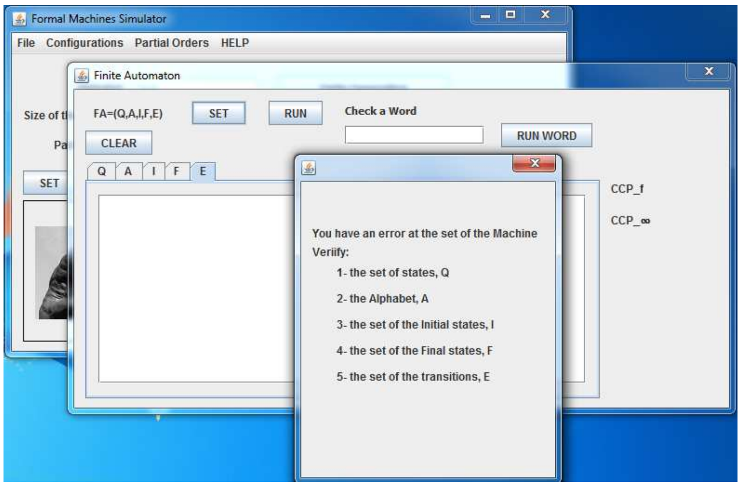


Figure 19: The computational model, of the GU_SFM, chosen a Finite Automata



You have an error at the set of the Machine
Verify:
1- the set of states, Q
2- the Alphabet, A
3- the set of the Initial states, I
4- the set of the Final states, F
5- the set of the transitions, E

Figure 20: The Finite Automata interface, in the GU_SFM, if you clik in button SET you obtain an explanation of the parameters

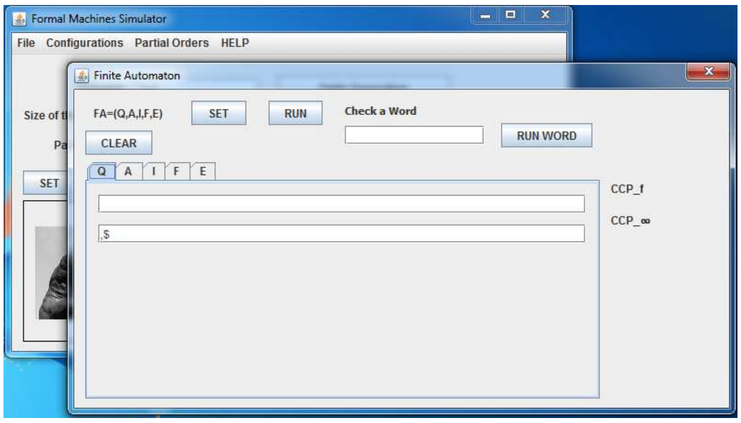


Figure 21: Introducing the parameter Q, in the GU_SFM, of the Finite Automata interface

In the figure 22 and 23 are showed the choices of the subtab called Formal Machine in the file tab, file>Formal Machine. This subtab allows to open the interface that allows to setting a FM, it is showed in figure 22. In the figure 23 is showed the interface of the FM.

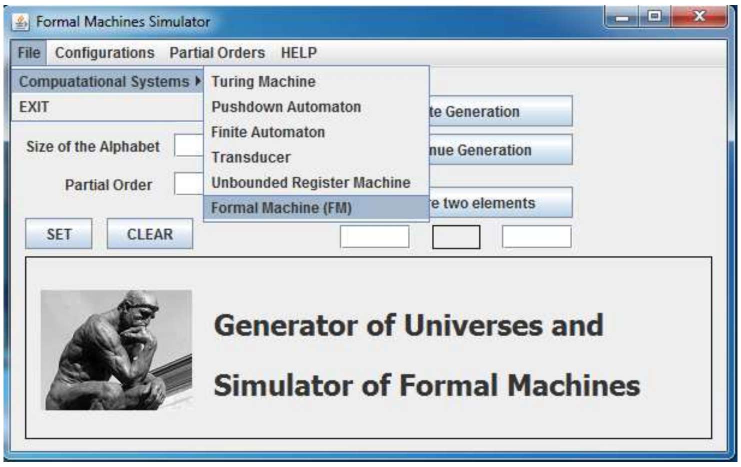
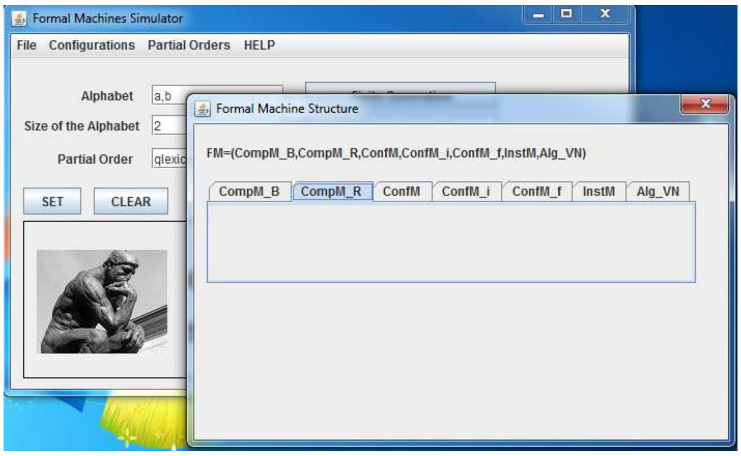


Figure 22: In the file tab of the GU_SFM, choosing a FM



Alphabet a,b
Size of the Alphabet 2
Partial Order qlexic

FM=(CompM_B,CompM_R,ConfM,ConfM_i,ConfM_f,InstM,Alg_VN)

Figure 23: In the file tab of the GU_SFM, the FM interface

4.4 Games and Formal Machines

4.4.1 Tic Tac Toe Game

Now I present the first game that I developed to implement a FM as a player of a game. I chosen the Tic Tac Toe game because it is a natural game for prepare an implementation of the Four In Line (FIL) game and the FIL game is a natural game where, for play it, is necessary to use concepts associated with the following idea, what is a behavior "being intelligent". Any player of the Four In Line game needs to use a lot of skills and this is important when it is being planned to use the machines in the context of Artificial Intelligence concepts.

Any medium player of the Four In Line game needs to have a considerable Intelligence.

In the firsts three figures 24, 25, 26, I show the folder of the Tic Tac Toe (TTT) game in 24, the folder of the TTT game and, in figure 25 I show the folder open. It can be seen that the folder has inside a folder lib. The folder contents is showed in the figure 26.

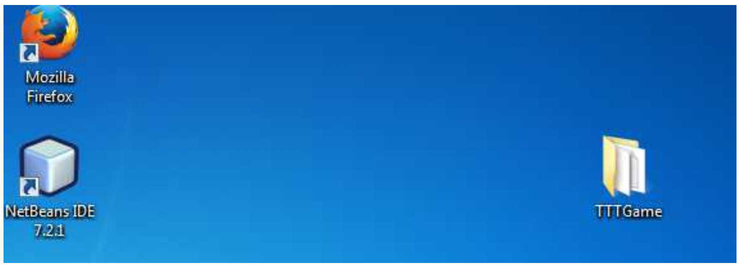
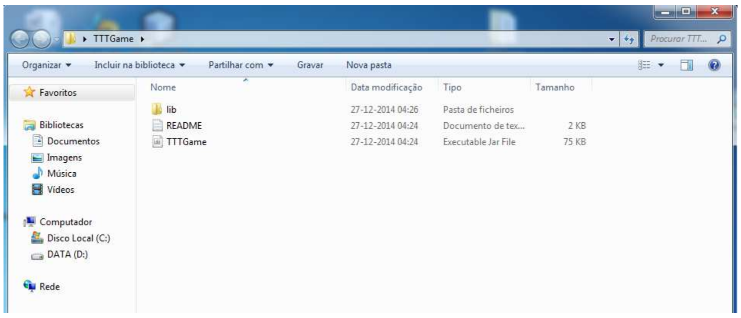
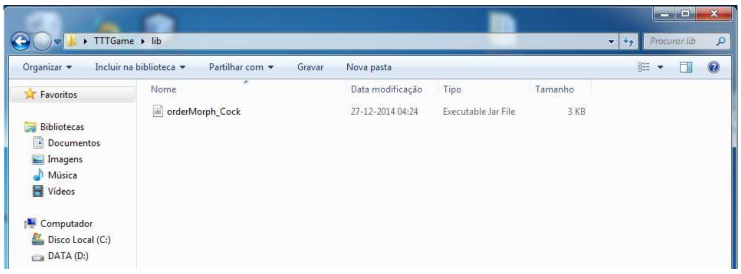


Figure 24: The folder of the Tic Tac Toe game (TTT game)



Nome	Data modificação	Tipo	Tamanho
lib	27-12-2014 04:26	Pasta de ficheiros	
README	27-12-2014 04:24	Documento de tex...	2 KB
TTTGame	27-12-2014 04:24	Executable Jar File	75 KB

Figure 25: Opening the TTTgame folder



Nome	Data modificação	Tipo	Tamanho
orderMorph_Cock	27-12-2014 04:24	Executable Jar File	3 KB

Figure 26: Opening the lib subfolder of the TTT game

In the next five figures I show the interface of the TTT game, figure 27, and the elements of the file tab, figures 28, 30, each one of the subtabs of the file tab, figures 29, 31. In the figure 29 is showed the About subtab of the About tab. In it I show the About subtab where it can be seen who are the authors of this implementation. In figure 30 is marked the FM Technology subtab. In figure 31 is showed the content of the FM Technology subtab.

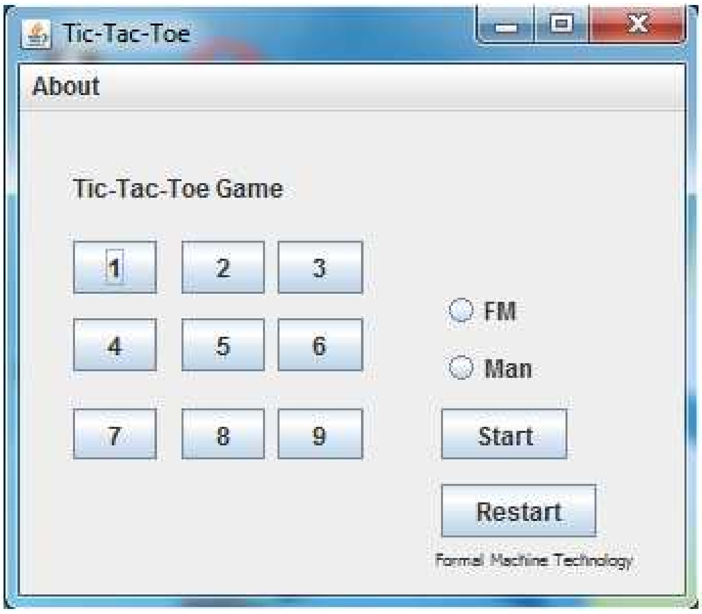


Figure 27: The interface of the TTT game

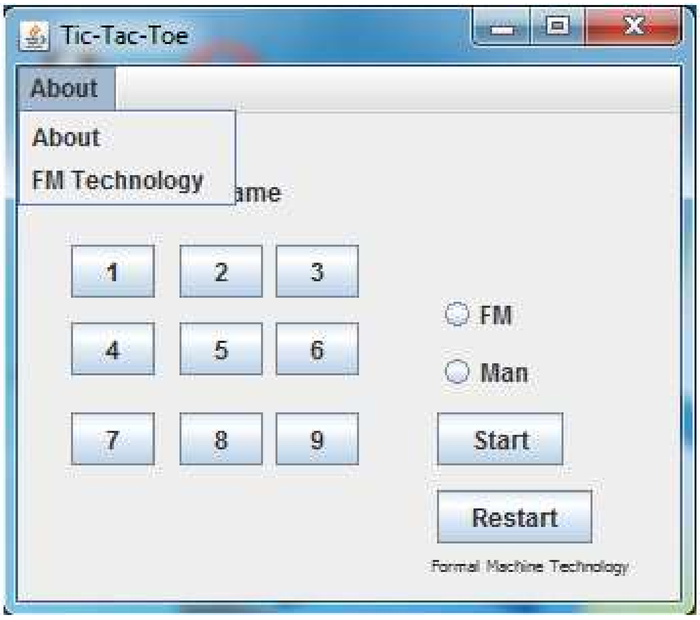
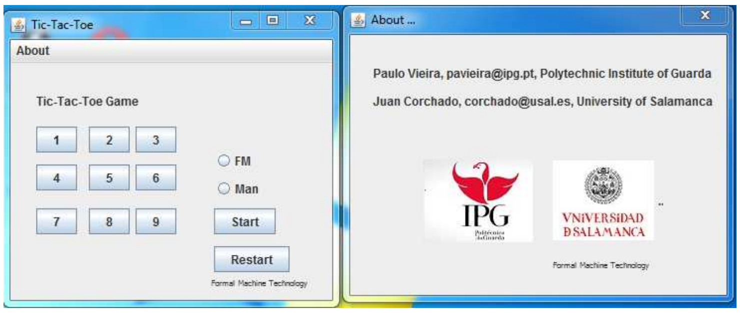


Figure 28: The About tab of the TTT game



Paulo Vieira, pavieira@ipg.pt, Polytechnic Institute of Guarda

Juan Corchado, corchado@usal.es, University of Salamanca

Figure 29: The subtab Authors of the About tab

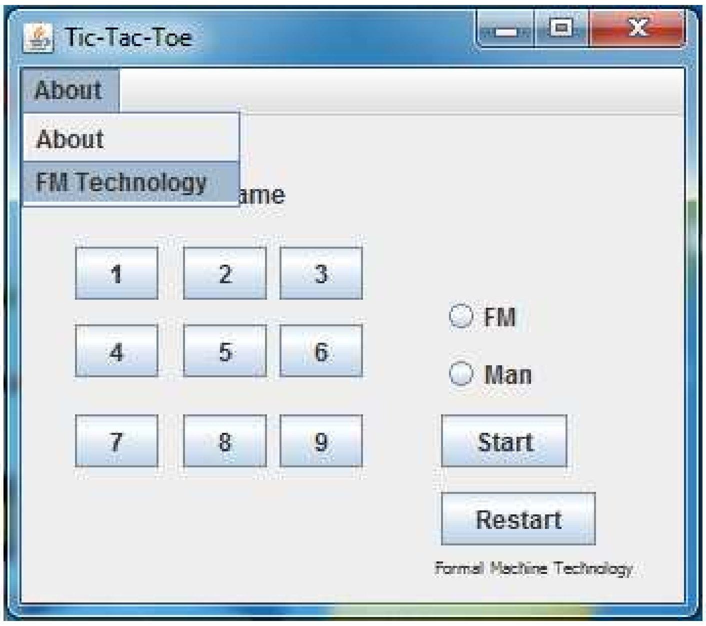
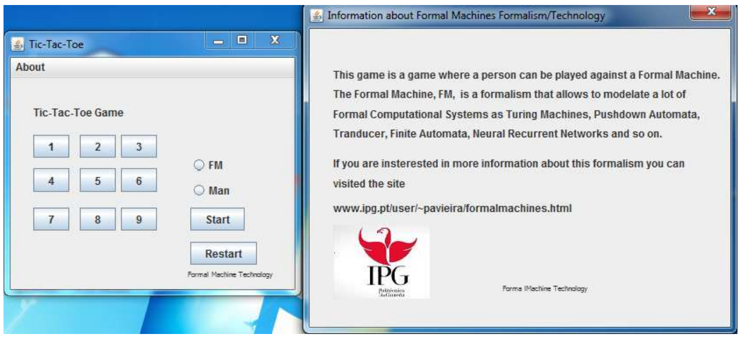


Figure 30: The subtab FM Technology of the About tab


Tic-Tac-Toe
About
Tic-Tac-Toe Game
1
2
3
4
5
6
7
8
9
FM
Man
Start
Restart
Formal Machine Technology
Information about Formal Machines Formalism/Technology

This game is a game where a person can be played against a Formal Machine. The Formal Machine, FM, is a formalism that allows to modelate a lot of Formal Computational Systems as Turing Machines, Pushdown Automata, Tranducer, Finite Automata, Neural Recurrent Networks and so on.

If you are insterested in more information about this formalism you can visited the site

www.ipg.pt/user/~pavieira/formalmachines.html

Figure 31: The dialog box of the subtab FM Technology of the About tab

The figures 32 and 33 show the two different ways of start the TTT game. The game can start with the FM doing the first move, figure 33, or with an Human Being preparing the first move, figure 32.

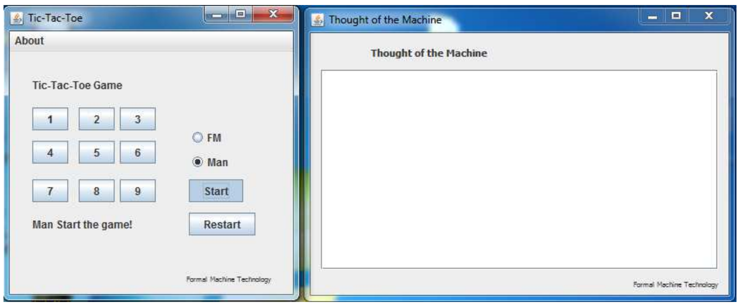
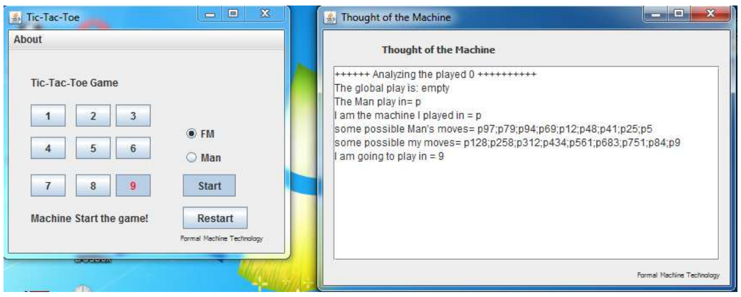


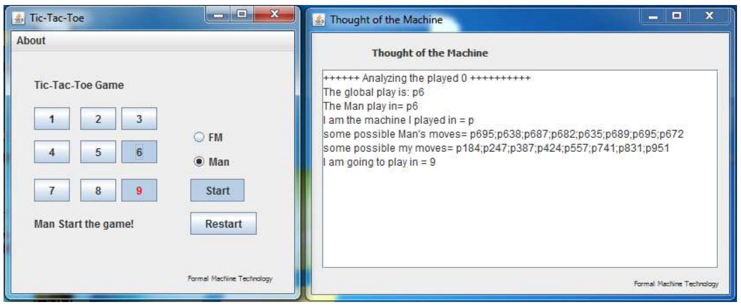
Figure 32: Choosing the first player, the Human Being, and start the game



```
++++++ Analyzing the played 0 ++++++++++
The global play is: empty
The Man play in= p
I am the machine I played in = p
some possible Man's moves= p97;p79;p94;p69;p12;p48;p41;p25;p5
some possible my moves= p128;p258;p312;p434;p561;p683;p751;p84;p9
I am going to play in = 9
```

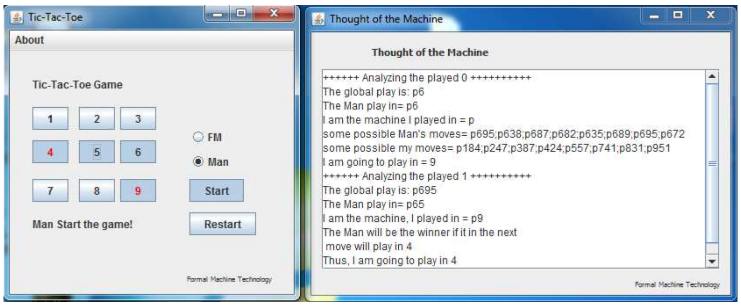
Figure 33: Choosing the first player, the Formal Machine, and start the game

In the following figures is presented all the moves of a complete game. Is possible to see the first move made by a Human Being player and the answer of the FM with its thoughts, figure 34. The sequence of moves alternating between the FM player and the Human Being player is in figures 35,36, 37, 38. For last, in figure 39 is showed, the resulted of the game.



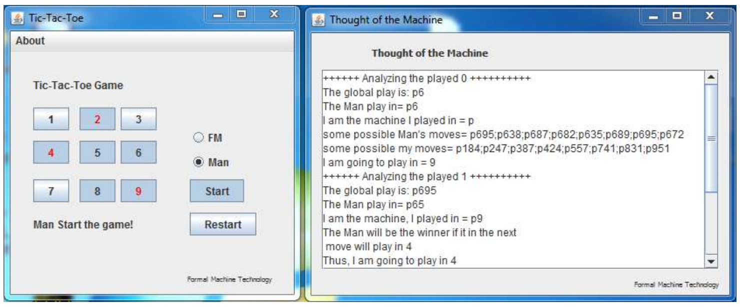
```
++++++ Analyzing the played 0 ++++++++++
The global play is: p6
The Man play in= p6
I am the machine I played in = p
some possible Man's moves= p695;p638;p687;p682;p635;p689;p695;p672
some possible my moves= p184;p247;p387;p424;p557;p741;p831;p951
I am going to play in = 9
```

Figure 34: The Human Being plays is first move and the FM makes its move in answer



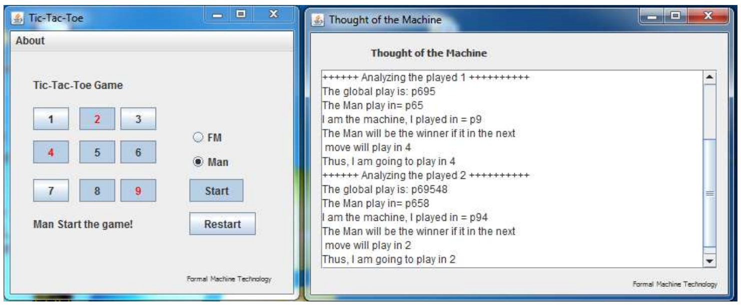
```
++++++ Analyzing the played 0 ++++++++++
The global play is: p6
The Man play in= p6
I am the machine I played in = p
some possible Man's moves= p695;p638;p687;p682;p635;p689;p695;p672
some possible my moves= p184;p247;p387;p424;p557;p741;p831;p951
I am going to play in = 9
++++++ Analyzing the played 1 ++++++++++
The global play is: p695
The Man play in= p65
I am the machine, I played in = p9
The Man will be the winner if it in the next
 move will play in 4
Thus, I am going to play in 4
```

Figure 35: A sequence of moves, the red is the FM, the gray is the Human Being moves



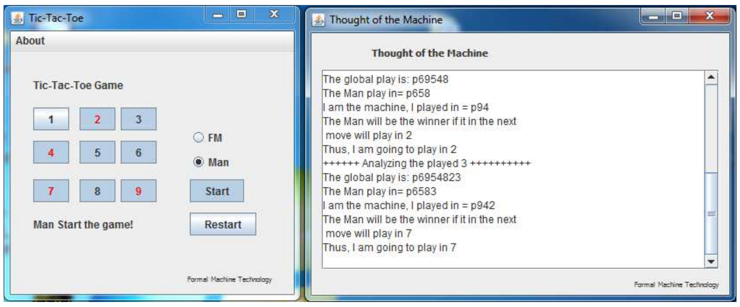
```
++++++ Analyzing the played 0 ++++++++++
The global play is: p6
The Man play in= p6
I am the machine I played in = p
some possible Man's moves= p695;p638;p687;p682;p635;p689;p695;p672
some possible my moves= p184;p247;p387;p424;p557;p741;p831;p951
I am going to play in = 9
++++++ Analyzing the played 1 ++++++++++
The global play is: p695
The Man play in= p65
I am the machine, I played in = p9
The Man will be the winner if it in the next
 move will play in 4
Thus, I am going to play in 4
```

Figure 36: The moves in the TTT game continue and the thoughts of the machine can be read in the right interface



```
++++++ Analyzing the played 1 ++++++++++
The global play is: p695
The Man play in= p65
I am the machine, I played in = p9
The Man will be the winner if it in the next
 move will play in 4
Thus, I am going to play in 4
++++++ Analyzing the played 2 ++++++++++
The global play is: p69548
The Man play in= p658
I am the machine, I played in = p94
The Man will be the winner if it in the next
 move will play in 2
Thus, I am going to play in 2
```

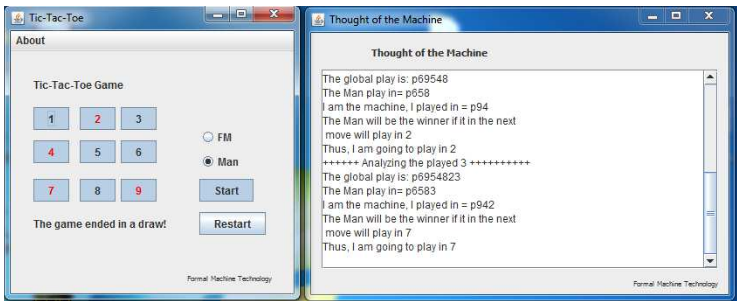
Figure 37: The moves in the TTT game continue



Man Start the game!

```
The global play is: p69548
The Man play in= p658
I am the machine, I played in = p94
The Man will be the winner if it in the next
 move will play in 2
Thus, I am going to play in 2
++++++ Analyzing the played 3 ++++++++++
The global play is: p6954823
The Man play in= p6583
I am the machine, I played in = p942
The Man will be the winner if it in the next
 move will play in 7
Thus, I am going to play in 7
```

Figure 38: More moves in the TTT game



The game ended in a draw!

```
The global play is: p69548
The Man play in= p658
I am the machine, I played in = p94
The Man will be the winner if it in the next
 move will play in 2
Thus, I am going to play in 2
++++++ Analyzing the played 3 ++++++++++
The global play is: p6954823
The Man play in= p6583
I am the machine, I played in = p942
The Man will be the winner if it in the next
 move will play in 7
Thus, I am going to play in 7
```

Figure 39: The game is over

4.4.2 Four In Line Game

The Four In Line game is a known game[37], the game[38] is constituted by a table that is a $n \times m$ matrix, with n rows and m columns. This $n \times m$ rectangle is as a matrix of $n \times m$ squares. I implemented a game whose table of the game can be setting among a matrix 4×4 and a matrix 10×10. This game is a game that is played between two players, in my implementation one of the players is a FM. The versatility of the FM allows to say that it is a good computational model to write and solve engineering problems with formal methods. I think in two types of algorithms to implement a FM. One of it is a serial FM implementation and the other is a parallel FM implementation. This sub subsection has the following structure. It is divided in four parts:
- The interface of the game, where is described the interface of the game;
- Playing the game, where is described how the game is run;
- Designing the FIL game as a FM, where is described the design of the FM player;

[37]A lot of references of this game can be found in internet

[38]`http://www.ipg.pt/user/~pavieira/private/SW/FILGame5x5.jar`, verify the working environment of the game, if you not did this the game can not be run `http://www.ipg.pt/user/~pavieira/private/SW/index.html`

- Results, where is presented a statistical study about the results of 100 completed games played between human beings and the FM player.

The Interface of the Game

In this section I describe the interface of the Four in Line implementation. For to do this I am putting nine figures of the interface, some of them with the game to be configured and others with it in running. I also comment each one of the figure and I say what it represent in the game.

The figure 40 and the figure 41 shows the interface of the game.

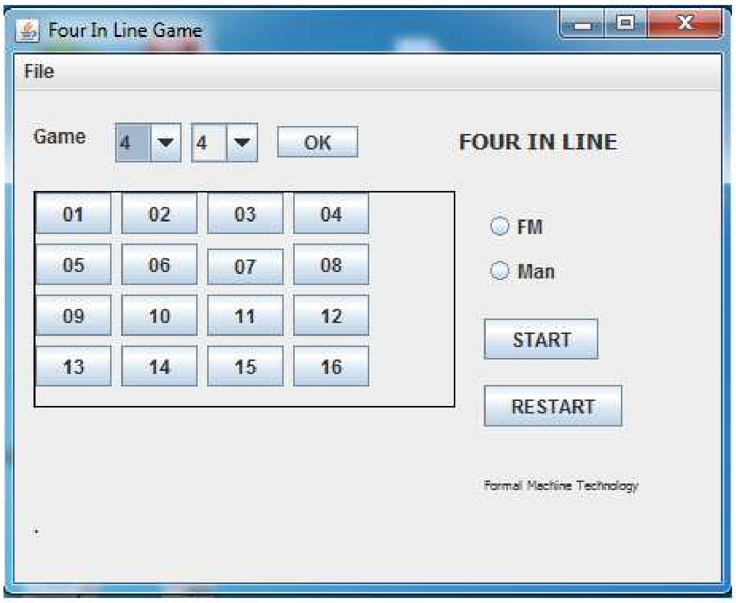


Figure 40: The interface of the Four In Line Game

For a better understanding I divided the layout of the interface of the figure 40 in a left part and a right part. In the left there is the table game, for default is a matrix 4×4. In the right part there are two blocks of buttons each one with two buttons. One of them has the buttons called FM and Man to choose who made the first move, who is the first player. The other block has the buttons called start (to start the game) and restore (to reset the game).

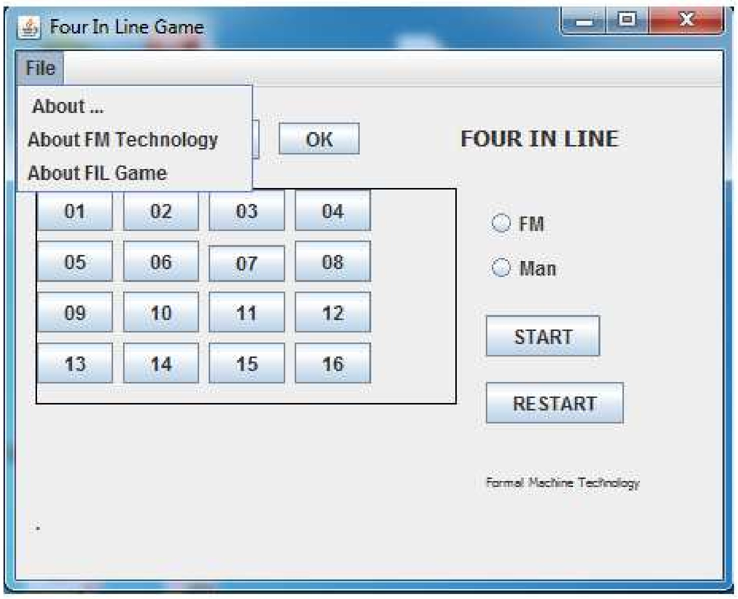


Figure 41: What are the elements presented in file tab

In the figure 41 is showed the tab file. In that tab is possible to obtain information about who developed the game, about the state of the art of the FM Technology and about this implementation of the FIL Game.

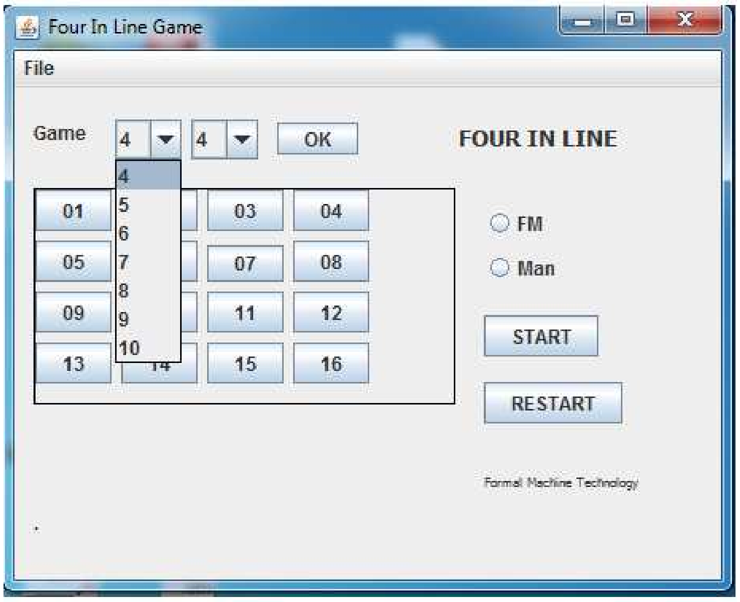


Figure 42: Choose the number of rows of the table, among 4 to 10

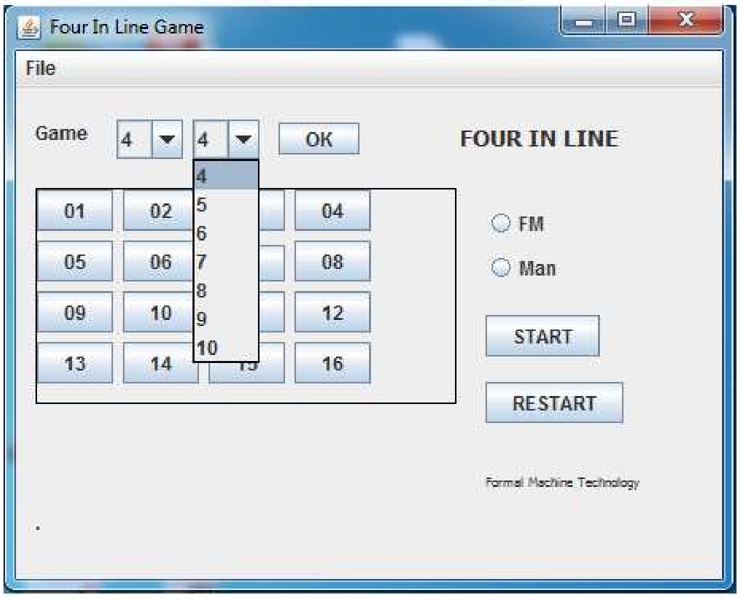


Figure 43: Choose the number of columns of the table, among 4 to 10

In figure 42 and figure 43 are represented the possible choices for the table of the game. Is possible to choose a table among a 4×4 matrix and a 10×10 matrix, some of them is not yet implemented. In the implementation already done, the table can be a matrix 4×4, 4×5, 5×4 or 5×5. The algorithm implemented is the FM serial procedure. In all of them the FM behaves as a serial player. In the tables of the games, whose matrices are a range from 5×6, 6×5 to 10×10, the FM behaves as a parallel player and yet is not implemented.

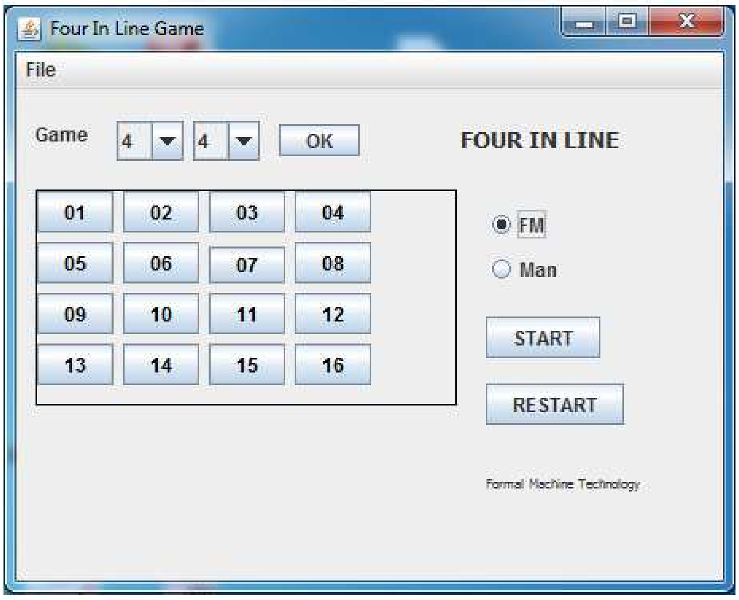
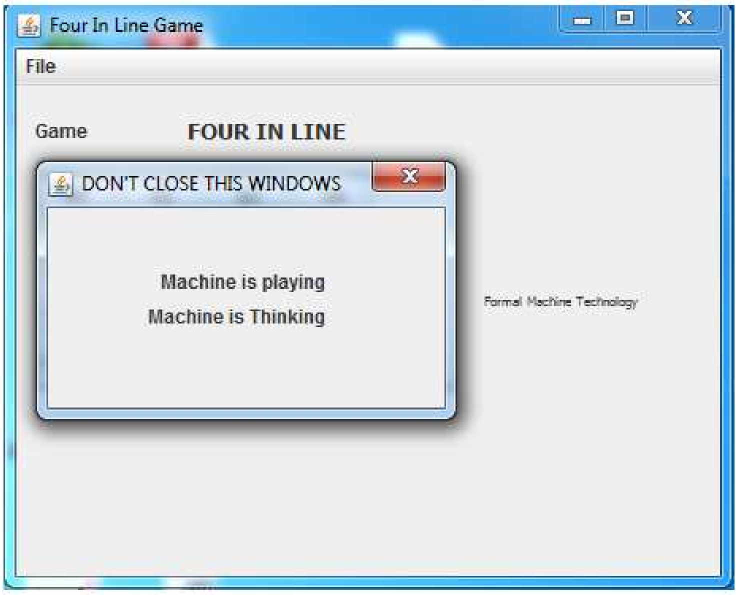


Figure 44: Choosing the first player; the FM



Machine is playing
Machine is Thinking

Figure 45: The FM is playing

In the figure 44 is chosen who does the first move. The option in the figure is the FM. The figure 45 shows a typical FM moves. The interface of the game announced that the FM is “thinking” in its move.

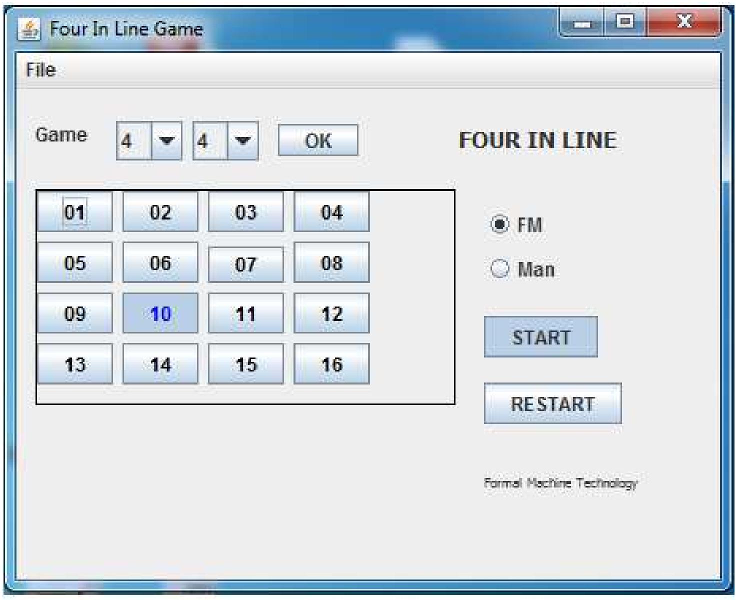


Figure 46: The FM did a move to the square 10

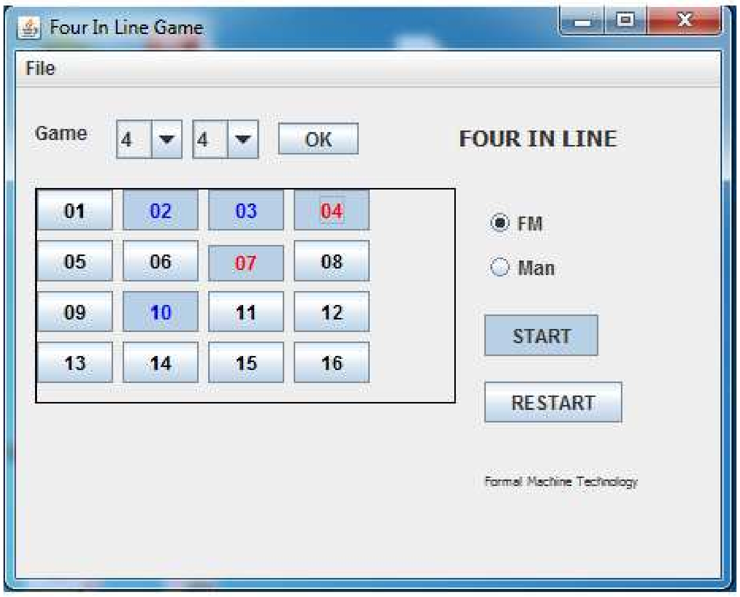
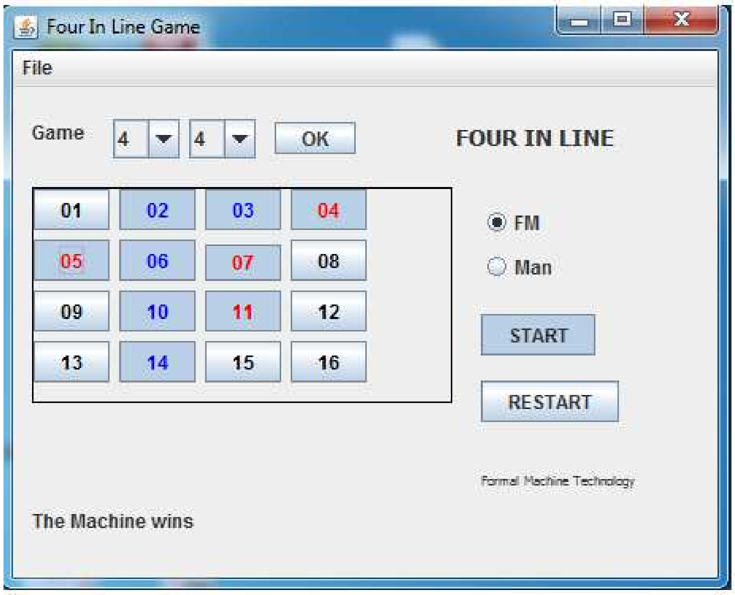


Figure 47: The game between the FM and the Human player in ongoing

The figure 46 shows a move of the FM player, its first move. The figure 47 shows the state of the table game after several moves. The red moves representing the moves done by the Human Player and the blue moves are the moves done by the FM.



The Machine wins

Figure 48: The FM made a move to the square 10

The figure 48 represents a result, with higher probability, if you play against this FM player, the FM wins.

Table 15: moves of the game illustrated, in figures 46, 47, 48, from the page 77

Moves	FM: FM	Human Being (HB)	comment about the FM move
First player, FM	10		random move of the FM
HB	10	4	
FM	10,3	4	obstruction of the HB's played
HB	10,3	4,7	
FM	10,3,2	4,7	tentative of the FM do a vertical alignment
HB	10,3,2	4,7,5	
FM	10,3,2,6	4,7,5	obstruction made by the FM to the HB's tentative to made an horizontal alignment
HB	10,3,2,6	4,7,5,11	bad move mabe by the FM
FM	10,3,2,6,14	4,7,5,11	The FM is the winner

Playing the Game

Now let's go to talk about the game. The way as the FM sees the game depends of the initial environment of the game. This environment is a table, that is set as matrix, where the game is run. If the table is a matrix 4×4, 4×5, 5×4 or 5×5 the FM implemented makes serial processing, and I say that the FM has a serial procedure. If the table implemented is from of the matrices referred until a matrix 10×10 the FM does parallel processing, figure 7, 8.

Designing a FIL game player as a FM

To design a FM as a player of a FIL game have in attention what is in the page 28 to know how to transform an engineering problem in a FM problem. Now, I transport the idea referred to the FIL game.
i) In the FIL game first is necessary to setting the table game.
ii) The set of the game consists in to choose the size of the table where the game is running. What is the size of the matrix which will run the game?,
iii) After the game is setting who starts the game?
iv) The moves consist in click in the buttons (the squares) of the matrix.

Thus, the constitution of the FM is generically:

Table 16: The generic constitution of the FM

- components of the FM	i) C_1 - the moves, in the game, made by the two players ii) C_2 - the moves, in the game, made by the Human Being (HB) iii) C_3 - the moves, in the game, made by the FM.
- instructions of the FM	The only action is to use the buttons, clicking in it, for move. Thus the instruction that the FM possesses is only one.
- $\vdash$	Defined through of the COA

Results

Here I present a statistical study about this FIL game implementation. I use inference statistic. For doing this I collect a set of data that are the results of play the game, one hundred times. In 50'% of the games played the first move is made by the FM and in the others 50% of the first moves are made by the human being that play the game against the FM. I gathered a sample of 100 results of plays, from ten different human players. The players are chosen in random way, that is a way to guarantee the representation of the sample.

Table 17: Set of data collected for a game matrix 4×4. Legend: D-draw, FM - wins the FM, HB-wins the Human Being

Matrix table 4×4									
First player									
FM					*HB*				
FM	FM	D	D	D	FM	D	D	FM	D
D	D	D	D	D	D	D	FM	D	FM
D	FM	D	FM	FM	D	D	D	D	D
D	FM	FM	FM	D	D	FM	D	D	FM
D	D	D	FM	D	D	FM	D	D	FM
FM	D	D	D	D	D	D	D	D	FM
FM	FN	FM	FM	FM	D	FM	D	FM	D
FM	D	D	FM	FM	D	FM	FM	FM	D
D	D	FM	FM	FM	D	FM	FM	FM	D
FM	D	FM	FM	D	D	D	D	FM	FM

I define a random variable $X_{P_{FM}}$ (respectively, $X_{P_{HB}}, X_{P_D}$) as "the proportion of victories that the FM obtains" (respectively, "the proportion of victories that the HB obtains", "the proportion of draws in the game"). I know, by the theory of statistic inference, that $X_{P_{FM}}$ has Bernoulli distribution with parameter p, $X_{P_{FM}} \sim B(p)$ (respectively, $X_{P_{HB}} \sim B(p)$, $X_{P_D} \sim B(p)$). This random variable possesses Bernoulli distribution and, as the sample has more than 30 elements the theory and practical say that, it can be approximated through of a normal distribution with average p_{FM} and variance $\frac{p_{FM}\, q_{FM}}{n}$. Thus, $\frac{X_{P_{FM}} - p_{FM}}{\sqrt{\frac{p_{FM}\, q_{FM}}{n}}} \sim N(0,1)$ (respectively, $\frac{X_{P_{HB}} - p_{HB}}{\sqrt{\frac{p_{HB}\, q_{HB}}{n}}} \sim N(0,1)$, $\frac{X_{P_D} - p_D}{\sqrt{\frac{p_D\, q_D}{n}}} \sim N(0,1)$) with $q_{FM} = 1 - p_{FM}$ (respectively, $q_{HB} = 1 - p_{HB}$, $q_D = 1 - p_D$). From this the results can be extrapolated for the population through of the use of the confidence interval with 95 percent of confidence.

$$p_{FM} \in [\hat{p_{FM}} - z_{\alpha/2}K, \hat{p_{FM}} + z_{\alpha/2}K] \text{ with confidence of } 100\ (1-\alpha)\ \%$$
$$\text{(respectively, } p_{HB} \in [\hat{p_{HB}} - z_{\alpha/2}K, \hat{p_{HB}} + z_{\alpha/2}K] \text{ with confidence of } 100\ (1-\alpha)\%,$$
$$p_D \in [\hat{p_D} - z_{\alpha/2}K, \hat{p_D} + z_{\alpha/2}K] \text{ with confidence of } 100\ (1-\alpha)\%)$$
$$\text{and } K = \sqrt{\frac{\hat{p_{FM}}\,\hat{q_{FM}}}{n}} \text{ (respectively, } K = \sqrt{\frac{\hat{p_{HB}}\,\hat{q_{HB}}}{n}},\ K = \sqrt{\frac{\hat{p_D}\,\hat{q_D}}{n}})$$

Table 18: values calculated from the sample

Matrix table 4×4	
symbols	values
α	0,05
$\frac{\alpha}{2}$	0,025
$z_{\frac{\alpha}{2}}$	1,959963985
$\hat{p}_{FM}$	0,43
$\hat{q}_{FM}$	0,57
$K = \sqrt{\frac{\hat{p}_{FM}\,\hat{q}_{FM}}{n}}$	0,049507575
$\hat{p}_{FM} - z_{\alpha/2}K, \hat{p}_{FM}$	0,332966936
$\hat{p}_{FM} + z_{\alpha/2}K, \hat{p}_{FM}$	0,527033064
$\hat{p}_{HB}$	0
$\hat{q}_{HB}$	1
$K = \sqrt{\frac{\hat{p}_{HB}\,\hat{q}_{HB}}{n}}$	0
$\hat{p}_{HB} - z_{\alpha/2}K, \hat{p}_{HB}$	0
$\hat{p}_{HB} + z_{\alpha/2}K, \hat{p}_{HB}$	0
$\hat{p}_D$	0,57
$\hat{q}_D$	0,43
$K = \sqrt{\frac{\hat{p}_D\,\hat{q}_D}{n}}$	0,049507575
$\hat{p}_D - z_{\alpha/2}K, \hat{p}_D$	0,472966936
$\hat{p}_D + z_{\alpha/2}K, \hat{p}_D$	0,667033064

Analyzing data I can say with a confidence of 95% that the FM wins between 33% to 52% of the games played, that they draw between 47% to 68% of the games played and that the wins of the Human Being are residual and haven't expression. This shows that the algorithm implemented, the FM player, is a good algorithm to implement the Artificial Intelligence necessary to play this game.

The FIL game was chosen to implement a FM, as a player of the game, for three reasons. One of it because the game is a game playing between two players, the second because one of the players is an human being and the third because for play it is necessary some admixture of properties that normally are associated with intelligence and presupposes the exercise of some skills. Thus, the FM player is in competition against a Human Being. At the moment that I write this thesis I have implemented the FM serial procedure not the parallel FM procedure (this is ongoing). Thus, now is possible to play the game in tables whose matrix are 4×4, 4×5, 5×4 and 5×5. The implementation of tables that are matrices from 5×6, 6×5 until 10×10 are ongoing and follows the algorithm illustrated in figure 8, the parallel FM procedure. For measure the skills of the FM player I did a statistical study when the table game is a matrix of 4×4. The results of the measures presented show that the FM player has a considerable skill to play this game. In the future I am going to do similar analysis for the others environments possibles.

5 Mathematics and FMs

This section is divided in three subsections called: Mathematical results in computational models, Mathematical results in Formal Machines and Formal Machines and Category Theory. In the subsection 5.1 are defined; arithmetic, algebraic and logic operations in the CSFMs; are constructed some mathematical structures for CSFMs; and are presented theorems that allow to construct new CSFMs from the use of these operations. In second subsection, subsection 5.2, are defined operations in languages recognized by FMs and from these are constructed new languages, and are presented theorems that allow to construct the FMs that recognize the new languages constructed. In the third subsections, subsection 5.3, is showed how to write several FCSs and FMs in categories and how to embed the FCSs, written as categories, in FMs.

5.1 Mathematical Results in the Computational Model

In this subsection I define several properties in CSFM, the computational model of a FM as already was referred, the CSFM, is a 4-tuple CSFM=(VN_{Alg},psm,$\mathcal{A}$,$\vdash$). The operations here defined for the CSFM are all about the psm. I define arithmetic, algebraic and logic properties of the psm and I establish theorems that from that properties allow to create CSFMs.

5.1.1 Formal Machines and Dynamical Systems

$b(t) = psm(A, \sigma_P(t))$ denotes the active configurations on the iteration or time t. This allows to study the evolution of a FM through of a sequence of iterations or through of the time.

5.1.2 Formal Machines and Algebra

Arithmetic operations at the *psm*
Let A and B be matrices and $psm(A, \sigma_{P_A})$, $psm(B, \sigma_{P_B})$ are the psm's of the FMs respectively f_1M and f_2M.
i) *addiction*:
$psm(A, \sigma_{P_A}) + psm(B, \sigma_{P_B}) = psm(A + B, \times_{i=1}^{m} min\{\sigma_{P_A}(i), \sigma_{P_B}(i)\})$, where:
InstM=Inst$_1$M $\cup$ Inst$_2$M, $A, B \in Matrix(n, m)$
ii) *subtraction*:
$psm(A, \sigma_{P_A}) - psm(B, \sigma_{P_B}) = psm(A - B, \times_{i=1}^{m} min\{\sigma_{P_A}(i), \sigma_{P_B}(i)\})$, where:
InstM=Inst$_1$M $\cup$ Inst$_2$M, $A, B \in Matrix(n, m)(\mathbb{R})$
iii) *multiplication*:
$psm(A, \sigma_{P_A}) psm(B, \sigma_{P_B}) = psm(AB, \sigma_{P_B})$, where:
InstM=Inst$_1$M $\cup$ Inst$_2$M, $A \in Matrix(n, m)(\mathbb{R})$ and $B \in Matrix(m, k)(\mathbb{R})$
iv) *inverse*:
$psm(A, \sigma_{P_A})^{-1} = psm(A^{-1}, \sigma_{P_A})$ where:
A is an invertible matrix, InstM=Inst$_1$M, $A \in Matrix(n, n)(\mathbb{R})$

v) Let $\alpha \in \mathbb{R}$ be such that $\alpha < 0$, $\alpha psm(A, \sigma_{P_A}) = psm(\alpha A, 1 - \sigma_{P_A})$
vi) Let $\alpha \in \mathbb{R}$ be such that $\alpha \geq 0$, $\alpha psm(A, \sigma_{P_A}) = psm(\alpha A, \sigma_{P_A})$

Theorem 5.1 *Let $\mathcal{F}$ be the following set*
$\mathcal{F} = \{psm(A, \sigma_{P_A}): A \in M(m,n)(\mathbb{R}), \sigma_{P_A} \in \{0,1\}^{|ConfM|}\}$,
$psm(0, \sigma_{P_0})$ where 0 is the null matrix, $\sigma_{P_0} = \{0\}^{|ConfM|}$ and $psm(I_n, \sigma_{P_{I_n}})$ where $\sigma_{P_{I_n}} = \{1\}^{|ConfM|}$
i) For each $psm(A, \sigma_{P_0})$ exist only one $psm(-A, 1 - \sigma_{P_0})$ such that
$psm(A, \sigma_{P_0}) + psm(-A, 1 - \sigma_{P_0}) = psm(0, \sigma_{P_0})$
ii) The set $(\mathcal{F}, +)$ is a commutative group with neutral element $psm(0, \sigma_{P_0})$
iii) The set $(\mathcal{F}, +)$ is a monoid with element one $psm(I_n, \sigma_{P_{I_n}})$
iv) $(\mathcal{F}, +, \mathbb{R}, .)$ is a vector space.

Proof 5.4: The proof of the different results can be obtained through of algebraic manipulations. The proof is left to the readers □

Operators configurations-components:
Concatenation (·), Transpose (T), Left Shift (LS), Right Shift (RS)
i) *Concatenation*:

$$psm(A, \sigma_{P_1}) \cdot psm(B, \sigma_{P_2}) = psm(\overbrace{A \cdot B}^{\in Matrix(n,m+k)}, \sigma_{P_1} \cdot \sigma_{P_2}),$$

InstM=∪ (Inst$_1$ · Inst$_2$)$(\sigma_{P_1}, \sigma_{P_2}) = inst_1(\sigma_{P_1}) inst_2(\sigma_{P_2})$ and $\sigma_P = \sigma_{P_1} \cdot \sigma_{P_2}$, with $A \in Matrix(n,m)$, $B \in Matrix(n,k)$
ii) *Transpose*:
$\text{T}(psm(A, \sigma_P)) = psm(\text{transpose}(A), \times_{i=1}^{n} 0)$
InstM$_{\text{T}}$=$\emptyset$, $0 \in Matrix(m,n)$ and $A \in Matrix(n,m)$
iii) *Right Shift*, RS:

$$\text{RS}(psm(A, \sigma_{P_1}), psm(B, \sigma_{P_2})) = (psm(\overbrace{A(I_m|B)}^{\in Matrix(n,k)}, \sigma_{P_{RS}})$$

with $\sigma_{P_{RS}} = \sigma_{P_1} \cdot \sigma_{P_2}$, $A \in Matrix(n,m)$, $B \in Matrix(m,k)$, $I \in Matrix(m,m)$ (I_m is the identity matrix of order m). $A(I_m|B) = (A|AB)$
iv) *Left Shift*, LS:

$$\text{LS}(psm(A, \sigma_{P_1}), psm(B, \sigma_{P_2})) = psm(\underbrace{(A|I)B}_{\in Matrix(n,k)}, \sigma_{P_2})$$

with $A \in Matrix(n,k)$, $I \in Matrix(m,m)$ (I is the identity matrix of order m), $B \in Matrix(k+m,k)$

Theorem 5.2 *i) $\{(psm(A, \sigma_{P_1}) psm(B, \sigma_{P_2})\}(psm(C, \sigma_{P_3})$*
$= (psm(A, \sigma_{P_1})\{psm(B, \sigma_{P_2})(psm(C, \sigma_{P_3})\}$
with $A \in Matrix(n,m)$, $B \in Matrix(m,k)$ and $C \in Matrix(k,r)$
ii) $(psm(A, \sigma_{P_1})\{psm(B, \sigma_{P_2}) + psm(C, \sigma_{P_3})\}$

$= psm(A,\sigma_{P_1})psm(B,\sigma_{P_2}) + psm(A,\sigma_{P_1})psm(C,\sigma_{P_3})$
with $A \in Matrix(n,m)$, $B \in Matrix(n,m)$ *and* $C \in Matrix(m,k)$
iii) $\{psm(A,\sigma_{P_1}) + psm(B,\sigma_{P_2})\}(psm(C,\sigma_{P_3})\}$
$= psm(A,\sigma_{P_1})psm(C,\sigma_{P_3}) + (psm(B,\sigma_{P_2})psm(C,\sigma_{P_3})$
with $A \in Matrix(n,m)$, $B \in Matrix(n,m)$ *and* $C \in Matrix(m,k)$
iv) $T(psm(A,\sigma_{P_1})psm(B,\sigma_{P_2}))$
$= T(psm(A,\sigma_{P_1}))T(psm(B,\sigma_{P_2}))$
with $A \in Matrix(n,m)$ *and* $B \in Matrix(m,k)$
v) $T(psm(A,\sigma_{P_1}) + psm(B,\sigma_{P_2}))$
$= T(psm(A,\sigma_{P_1})) + T(psm(B,\sigma_{P_2}))$
with $A \in Matrix(n,m)$ *and* $B \in Matrix(n,m)$
vi) $T(\alpha(psm(A,\sigma_{P_1})psm(B,\sigma_{P_2})))$
$= (\alpha T(psm(A,\sigma_{P_1})))T(psm(B,\sigma_{P_2}))$
with $\alpha \in \mathbb{R}$, $A \in Matrix(n,m)$ *and* $B \in Matrix(m,k)$
vii) $RS(psm(A,\sigma_{P_1}), psm(A^{-1}B,\sigma_{P_2}))$
$= psm((A|B),\sigma_{P_1} \cdot \sigma_{P_2})$
with $A \in Matrix(n,n)$ *and* $B \in Matrix(n,k)$

Proof 5.6: The proof of the different results can be obtained through algebraic manipulations. The proof is left to the reader. □

Propositional Logic operators: $\neg, \wedge, \vee, \rightarrow$
Let A be a matrix, $A \in Matrix(n,m)$
i) *negation*:
$\neg psm(A,\sigma_P) = psm(A,\neg\sigma_P) = psm(A, \times_{i=1}^{m}(1-\sigma_P(i)))$
ii) *conjunction*:
$\wedge$: $psm(A,\sigma_{P_1}) \wedge psm(A,\sigma_{P_2})$
$= psm(A, \times_{i=1}^{m}(\sigma_{P_1}(i)\sigma_{P_2}(i)))$
iii) *disjunction*:
$\vee$: $psm(A,\sigma_{P_1}) \vee psm(A,\sigma_{P_2})$
$= psm(A, \times_{i=1}^{m} max\{\sigma_{P_1}(i), \sigma_{P_1}(i)\})$
iv) *implication*, $\rightarrow$: $psm(A,\sigma_{P_1}) \rightarrow psm(A,\sigma_{P_2})$
$= psm(A, \times_{i=1}^{m} max\{1-\sigma_{P_1}(i), \sigma_{P_2}(i)\})$

Theorem 5.3 *Let* $\{psm(A,\sigma_{P_1})$, $psm(A,\sigma_{P_2})$, $psm(A,\sigma_{P_3})$ *be psm's of certain FMs.*
i) $\neg\neg psm(A,\sigma_{P_1}) = psm(A,\sigma_{P_1})$
ii) $psm(A,\sigma_{P_1}) \wedge psm(A,\sigma_{P_2})$
$= psm(A,\sigma_{P_2}) \wedge psm(A,\sigma_{P_1})$
iii) $psm(A,\sigma_{P_1}) \vee psm(A,\sigma_{P_2})$
$= psm(A,\sigma_{P_2}) \vee psm(A,\sigma_{P_1})$
iv) $psm(A,\sigma_P) \wedge psm(A,\sigma_P) = psm(A,\sigma_P)$
v) $psm(A,\sigma_P) \vee psm(A,\sigma_P) = psm(A,\sigma_P)$
vi) $\{psm(A,\sigma_{P_1}) \wedge psm(A,\sigma_{P_2})\} \wedge psm(A,\sigma_{P_3})$
$= psm(A,\sigma_{P_2}) \wedge \{psm(A,\sigma_{P_1}) \wedge psm(A,\sigma_{P_3})\}$
vii) $\{psm(A,\sigma_{P_1}) \vee psm(A,\sigma_{P_2})\} \vee psm(A,\sigma_{P_3})$

$= psm(A,\sigma_{P_2}) \vee \{psm(A,\sigma_{P_1}) \vee psm(A,\sigma_{P_3})\}$
viii) $\{psm(A,\sigma_{P_1}) \wedge psm(A,\sigma_{P_2})\} \vee psm(A,\sigma_{P_3})$
$= \{psm(A,\sigma_{P_1}) \vee \{psm(A,\sigma_{P_3})\} \wedge \{psm(A,\sigma_{P_2}) \vee psm(A,\sigma_{P_3})\}$
ix) $\{psm(A,\sigma_{P_1}) \vee psm(A,\sigma_{P_2})\} \wedge psm(A,\sigma_{P_3})$
$= \{psm(A,\sigma_{P_1}) \wedge \{psm(A,\sigma_{P_3})\} \vee \{psm(A,\sigma_{P_2}) \wedge psm(A,\sigma_{P_3})\}$
x) $psm(A,\sigma_{P_1}) \wedge \{psm(A,\sigma_{P_1}) \vee psm(A,\sigma_{P_3})\} = psm(A,\sigma_{P_1})$
xi) $psm(A,\sigma_{P_1}) \vee \{psm(A,\sigma_{P_1}) \wedge psm(A,\sigma_{P_3})\} = psm(A,\sigma_{P_1})$
xii) Let A be an element of $Matrix(n,m)$. The set $\{psm(A,\sigma_P) : \sigma_P \in \{0,1\}^{|ConfM|}\}$ is a reticulated with $\wedge$ and $\vee$.

Proof 5.3: The proof of the different results can be obtained through of algebraic manipulations. The proof is left to the reader □

5.2 Mathematical Results in Formal Machines

In this sections is demonstrated generic proprieties for FMs, all the proofs are constructive and allow to build new FMs from other(s) that perform new languages obtained from some known operators. How to build FMs from performed languages that result of the intersection $\cap$, union $\cup$, concatenation $\cdot$, difference $\setminus$ of two languages each one performed by a FM. For last is built the FM that performs the star language (results of apply in a language the operator *) and the iteration language (results of apply in a language the operator $^+$) of a language performed by a FM.

Theorem 5.4 *Let fM be a FM. Then:*
i) $A(C_1) \times A(C_2) \times .... \times A(C_n)$ *is an alphabet of the $ConfM$ and is a code.*
ii) $\forall C_i \in CompM_B : |C_i| \leq |\mathbb{Q}|$
iii) $\forall C_i \in CompM_B : |\times_{i=1}^{n} C_i| \leq |\mathbb{Q}|$
iv) $\forall C_i \in CompM_B$: *$(C_i,\cdot)$ is a semigroup with $\cdot$ the concatenation operator. If $\epsilon \in C_i$, then $(C_i,\cdot)$ is a monoide*
v) $(CompM_B,\cdot)$ is a semigroup wherein $\cdot$ is the following operation,
$c \cdot c' = (c_1,...,c_n)(c'_1,...,c'_n) = (c_1 \cdot c_1,...,c_n \cdot c'_n)$.
If $\forall i \in \{1,...,n\} : \epsilon \in C_i$, then $(ConfM,\cdot)$ is a monoide.

Proof 5.4
The proof can be made through algebraic manipulations. The proof is left to the reader. □

Theorem 5.5 *Let f_1M, f_2M be a FMs such that:*
f_kM=$(Comp_kM_B, Comp_kM_R, Conf_kM, Conf_kM_i, Conf_kM_f, Inst_kM, VN_{Alg})$ *with* $k = 1,2$.
i) Exists a $f_\cap M$,
$f_\cap M$=$(Comp_\cap M_B, Comp_\cap M_R, Conf_\cap M, Conf_\cap M_i, Conf_\cap M_f, Inst_\cap M, VN_{Alg})$, *such that:*
$L(f_\cap M) = L(f_1M) \cap L(f_2M)$

ii) Exists a projection function Φ *and a* $f_{\cup}M$,
$f_{\cup}M$=$(Comp_{\cup}M_B, Comp_{\cup}M_R, Conf_{\cup}M, Conf_{\cup}M_i, Conf_{\cup}M_f, Inst_{\cup}M, VN_{Alg})$, *such that:*
$\Phi(L(f_{\cup}M)) = L(f_1M) \cup L(f_2M)$
iii) Exists a $f_{\cdot}M$,
$f_{\cdot}M$=$(Comp_{\cdot}M_B, Comp_{\cdot}M_R, Conf_{\cdot}M, Conf_{\cdot}M_i, Conf_{\cdot}M_f, Inst_{\cdot}M, VN_{Alg})$, *such that:*
$L(f_{\cdot}M) = L(f_1M) \cdot L(f_2M)$
iv) Exists a $f_{\setminus}M$,
$f_{\setminus}M$=$(Comp_{\setminus}M_B, Comp_{\setminus}M_R, Conf_{\setminus}M, Conf_{\setminus}M_i, Conf_{\setminus}M_f, Inst_{\setminus}M, VN_{Alg})$, *such that:*
$L(f_{\setminus}M) = L(f_1M) \setminus L(f_2M)$

Proof 5.6

i) Just take,
$Comp_{\cap}M_B = \{C_1^1 \cap C_1^2, ..., C_n^1 \cap C_n^2\}$,
$Comp_{\cap}M_R = \{R_i^1 \times R_j^2 : R_i^1 \in Comp_1M_R, R_j^2 \in Comp_2M_R\}$,
$Conf_{\cap}M = \times_{i=1}^{n}(C_i^1 \times C_i^2)$
$Conf_{\cap}M_i = Conf_1M_i \cap Conf_2M_i$
$Conf_{\cap}M_f = Conf_1M_f \cap Conf_2M_f$
For each $I_1 : (Conf_1M)^{k_1} \longrightarrow \mathcal{P}(Conf_1M)$ and $I_2 : (Conf_2M)^{k_2} \longrightarrow \mathcal{P}(Conf_2M)$ I build the instruction $I_{\cap} : (ConfM)^{k_1+k_2} \longrightarrow \mathcal{P}(ConfM)$ such that for each $\vec{c} = (\vec{c_1}, \vec{c_2})$ with $\vec{c_1} \in Conf_1M$ and $\vec{c_2} \in Conf_2M$, then $I_{\cap}(\vec{c}) = I_1(\vec{c_1}) \cap I_2(\vec{c_2})$.

ii) In this case is supposed that the FMs f$_1$M and f$_2$M have the following properties: $Comp_1M_B \cap Comp_2M_B = \emptyset$, $Comp_1M_R \cap Comp_2M_R = \emptyset$ and $Inst_1M \cap Inst_2M = \emptyset$. Thus, in this situation, Just take,
$Comp_{\cup}M_B = \{C_{\cup}\} \cup Comp_1M_B \cup Comp_2M_B$ with $C_{\cup} = \{1, 2, 3, 4, 5, 6\}$,
$Comp_{\cup}M_R = C_{\cup} \times Comp_1M_R \times Comp_2M_R\}$,
$Conf_{\cup}M = C_{\cup} \times Conf_1M \times Conf_2$
$Conf_{\cup}M_i = \{1\} \times Conf_1M_i \times Conf_2M \cup \{2\} \times (Conf_1M \times Conf_2M_i)$
$Conf_{\cup}M_f = \{5\} \times Conf_1M_f \times Conf_2M \cup \{6\} \times (Conf_1M \times Conf_2M_f)$
For each $I_1 : (Conf_1M)^{k_1} \longrightarrow \mathcal{P}(Conf_1M)$ and $I_2 : (Conf_2M)^{k_2} \longrightarrow \mathcal{P}(Conf_2M)$ I build the instruction $I_{\cap} : (ConfM)^{k_1+k_2} \longrightarrow \mathcal{P}(ConfM)$ such that for each $\vec{c} = (\vec{c_1}, \vec{c_2})$ with $\vec{c_1} \in Conf_1M$ and $\vec{c_2} \in Conf_2M$, then $I_{\cap}(\vec{c}) = I_1(\vec{c_1}) \cap I_2(\vec{c_2})$.
The language performed by the f$_{\cup}$M is:
$L(\mathrm{f}_{\cup}\mathrm{M}) = \{\{1\} \times \pi_1(L(\mathrm{f}_1\mathrm{M})) \times Conf_2M\} \times \{\{5\} \times \pi_2(L(\mathrm{f}_1\mathrm{M})) \times Conf_2M)\} \bigcup$
$\{\{2\} \times Conf_1M \times \pi_1(L(\mathrm{f}_2\mathrm{M}))\} \times \{\{6\} \times Conf_1M \times \pi_2(L(\mathrm{f}_2\mathrm{M}))\}$. If in the language performed you apply a partial function, Φ, $\Phi(L(\mathrm{f}_{\cup}\mathrm{M}))$, where $\Phi : (Conf_{\cup}M_i \times Conf_{\cup}M_f) \longrightarrow Conf_1M \bigcup Conf_2M$ such that for each $\vec{c} = (\vec{c_1}, \vec{c_2}) \in (Conf_{\cup}M_i \times Conf_{\cup}M_f) \bigcap L(\mathrm{f}_{\cup}\mathrm{M})$ there are two exclusive situations:
1st) $\vec{c_1} = (1, \vec{c}_{11}, \vec{c}_{12})$ and $\vec{c_2} = (5, \vec{c}_{21}, \vec{c}_{22})$, or
2nd) $\vec{c_1} = (2, \vec{c}_{11}, \vec{c}_{12})$ and $\vec{c_2} = (6, \vec{c}_{21}, \vec{c}_{22})$,

for the 1st situation $\Phi(\vec{c}) = (\vec{c}_{11}, \vec{c}_{12})$ and for the 2nd $\Phi(\vec{c}) = (\vec{c}_{21}, \vec{c}_{22})$. Thus, $\Phi(L(\mathrm{f}_{\cup}\mathrm{M})) = L(\mathrm{f}_1\mathrm{M}) \cup L(\mathrm{f}_2\mathrm{M})$.

iii) Just take,
$CompM_B = \{C_1^1 \times C_1^2, ..., C_n^1 \times C_n^2\}$,
$CompM_R = \{R^1 \times R^2 : R^i \in Comp_k M_R, k = 1, 2\}$,
$Conf M = \times_{k=1}^{n}(C_k^1 \times C_k^2)$,
$Conf M_i = \{\times_{k=1}{}^n(c_k^1 X c_k^2) : \times_{i=1}^{n} c_k^j \in Conf_j M_i$, with $j = 1, 2\}$,
$Conf M_f = \{\times_{k=1}{}^n(c_k^1 X c_k^2) : \times_{i=1}^{n} c_k^j \in Conf_j M_f$, with $j = 1, 2\}$,
for each $I_1 \in InstM_1$, $I_2 \in InstM_2$, I build an instruction I that is denoted by $I_1 \cdot I_2$, $I = I_1 \cdot I_2$. Thus, for each $\vec{c}_j = ((\vec{c}_j^1, \vec{c}_j^2)_{j=1,2,...,n}) = (\vec{c}_1, \vec{c}_2, ..., \vec{c}_n) \in Conf M^k$ such that $Domain(I_1) \subseteq Conf M^{k_1}$, $Domain(I_2) \subseteq Conf M^{k_2}$ and $k = max\{k_1, k_2\}$,
$$\vec{c}_j = ((c_{1j}^1, c_{1j}^2), ..., (c_{nj}^1, c_{nj}^2)) = \underbrace{(c_{1j}^1, ..., c_{nj}^1)}_{\vec{c}_1} \cdots \underbrace{(c_{1j}^2, ..., c_{nj}^2)}_{\vec{c}_2}.$$
Then $I(\vec{c}) = I_1(\vec{c}_1) \cdot I_2(\vec{c}_2)$

iv) just take,
$CompM_{\setminus B} = Comp_1 M_B$
$CompM_{\setminus R} = Comp_1 M_R$
$Conf_{\setminus} M = Conf_1 M$
$Conf_{\setminus} M_i = Conf_1 M_i$
$Conf_{\setminus} M_f = Conf_1 M \; Conf_1 M_f$
$InstM = Inst_1 M$
$VN_{Alg} = VN_{Alg_1}$
The elements, instructions, of the set $Inst_1 M$ are all total functions. Thus, from each instruction $I \in Inst_1 M$ I construct an instruction $I_{\setminus}(\vec{c}) = \emptyset$ if $\vec{c} \notin domain(I)$ and $I_{\setminus}(\vec{c}) = I(\vec{c})$ when $\vec{c} \in domain(I)$ $\square$

Lemma 5.1 *Let $f_1 M$ be a FM such that:*
$f_1 M$=$(Comp_1 M_B, Comp_1 M_R, Conf_1 M, Conf_1 M_i, Conf_1 M_f, Inst_1 M, VN_{Alg})$.
Exist a FM, fM, such that $L(fM) = L(f_1 M) \cup \{\epsilon\}$.

Proof 5.1 Just take,
$\bar{C}_i = C_i \dot{\cup} \{\lambda\}$ with $i = 1, ..., n$ $C_i \in CompM_B$
$CompM_B = \{\bar{C}_i : \bar{C}_i = C_i \dot{\cup} \{\lambda\}, C_i \in CompM_B\}$ $CompM_{+R} = Comp_1 M_R$
$Conf_+ M = Conf_1 M$
$Conf_+ M_i = Conf_1 M_i \dot{\cup} \{(\lambda, \lambda, ..., \lambda)\}$
$Conf_+ M_f = Conf_1 M_f \dot{\cup} \{(\lambda, \lambda, ..., \lambda)\}$
$InstM = InstM$
$VN_{Alg} = VN_{Alg_1}$
$L(\mathrm{fM}) = L(\mathrm{f}_1\mathrm{M}) \cup \{\epsilon\}$.
$\square$.

Theorem 5.6 *i) Exists a f_*M,*
f_*M=($Comp_*M_B, Comp_*M_R, Conf_*M, Conf_*M_i, Conf_*M_f, Inst_*M, VN_{Alg}$), *such that:*
$L(f_*M) = L(f_1M)^*$
ii) Exists a f_+M,
f_+M=($Comp_+M_B, Comp_+M_R, Conf_+M, Conf_+M_i, Conf_+M_f, Inst_+M, VN_{Alg}$), *such that:*
$L(f_+M) = (L(f_1M))^+$

Proof 5.6

i) just take,
$CompM_{+B} = Comp_1M_B$
$CompM_{+R} = Comp_1M_R$
$Conf_+M = Conf_1M$
$Conf_+M_i = Conf_1M_i$
$Conf_+M_f = Conf_1M_f$
$VN_{Alg} = VN_{Alg_1}$
I construct the instructions, I_+, of the FM$_+$ from each one of $I \in Inst_1M$. $I_+(\vec{c}) = I(\vec{c})$ if $I(\vec{c}) \nsubseteq ConfM_f$ and $I_+(\vec{c}) = I(\vec{c}) \cup ConfM_i$ if $I(\vec{c}) \subseteq ConfM_f$.

ii) In i) I show that is possible to build a FM, called f$_+$M, that recognize the language L(f$_+$M) = L(f$_1$M)$^+$. Thus, using the Lemma 5.1 is possible to build a FM, f$_*$M, such that L(f$_*$M) = L(f$_1$M)$^+ \cup \{\epsilon\} = L$(f$_1$M)* $\square$.

5.3 Formal Machines and Category Theory

In this subsection I define, with several examples, the meaning of the notion to preserve a structure in a FM. In the tradition of Eilenberg ([Eil74]) I define Automata as a category. Thus a Finite Automaton (FA) is a 4-tuple

$$\mathcal{A} = (Q, A, E, I, F)$$

with $E \subseteq (Q \times A) \times Q$, and it can be described as a category $\mathcal{C}_{\mathcal{A}}$ where $Obj_{\mathcal{A}} = Q$ and the morphisms between q and q' is the set $Morf(q, q') = \{(q, a, q') : (q, a, q') \in E\}$.
A Pushdown Automaton (PA) is a 5-tuple

$$\mathcal{PA} = (Q, A, E, I, F, Z_0, \Gamma)$$

with $E \subseteq (Q \times A \times \Gamma) \times (Q \times \Gamma^*)$, can be described as a category $\mathcal{C}_{\mathcal{PA}}$ where $Obj_{\mathcal{PA}} = Q \times \Gamma$ and the morphisms between (q, γ_1) and (q', γ_1') is the set $Morf_{\mathcal{PA}}((q, \gamma_1), ((q', \gamma_1'))) = \{(q, a, \gamma_1, q', \gamma) \in E : \gamma \in \Gamma^*\gamma_1'\}$.

Following this tradition I construct a FM, fM, as a category, $\mathcal{C}_{\text{fM}}$, wherein $Obj_{\text{fM}} = \mathcal{P}(ConfM) = \{\sigma_P : \sigma_P \subseteq ConfM\}$. The morphisms between $\sigma_P, \sigma_P' \in$

$\mathcal{P}(ConfM)$ is the set $FM_{\mathrm{fM}}(\sigma_P,\sigma'_P)$=$\{(\sigma_P,\vec{c},I,\sigma'_P) : \sigma_P,\sigma'_P \in \mathcal{P}(ConfM)$, is a k - partial instruction, $\vec{c} = (c_1,...,c_k) \in (ConfM)^k$, with $c_1,...,c_k \in \sigma_P$ and $I(\vec{c}) = \sigma'_P\}$. if $\sigma'_P \nsubseteq \sigma_P$, then $Morf_{\mathrm{fM}}(\sigma_P,\sigma'_P) = FM_{\mathrm{fM}}(\sigma_P,\sigma'_P)$. When $\sigma'_P \subseteq \sigma_P$, then $Morf_{\mathrm{fM}}(\sigma_P,\sigma'_P) = \{(\sigma_P,NOP,\sigma'_P)\} \cup FM_{\mathrm{fM}}(\sigma_P,\sigma'_P)$.

Let $\mathcal{C}$ and $\mathcal{D}$ be categories and $\mathcal{F}$ a functor between $\mathcal{C}$ and $\mathcal{D}$, $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{D}$. Is said that a category, $\mathcal{C}$, *preserves a structure* in a FM if exist a FM, fM, as category $\mathcal{C}_{\mathrm{fM}}$, $\mathcal{F}$ is an embed functor and $\mathcal{F}(\mathcal{C}) = \mathcal{C}_{\mathrm{fM}}$. Now I give an example of a functor that preserve the structure of a FA in a FM.

Let $\mathcal{F}$ be an example of a functor that preserve the structure of a Finite Automaton in a FM. $\mathcal{F} : \mathcal{C}_{\mathcal{A}} \longrightarrow \mathcal{C}_{\mathrm{fM}}$ $\mathcal{F}_{Obj} : Obj_{\mathcal{A}} \longrightarrow Obj_{\mathrm{fM}}$, where $\mathcal{F}_{Obj}(q) = \{q\} \times A^*\mathrm{b}^\omega \times \mathbb{N}$ $\mathcal{F}_{Morf} : Morf(q,q') \longrightarrow Morf(\mathcal{F}(q),\mathcal{F}(q'))$, where $\mathcal{F}((q,a,q'))(c) = c'$, with $c = (q,u_1au_2\mathrm{b}^\omega,|u_1|+1)$ and $c' = (q',u_1au_2\mathrm{b}^\omega,|u_1|+2)$ and $u_1,u_2 \in A^*$, $a \in A$. It is easy to prove that this functor is a faithful functor.

The FCSs can be represented, in Category Theory, in terms of categories. In page 91 you can see how to define a Finite Automata, a Pushdown Automata, a Turing Machines and a FMs as a category.

$Machines$	$Obj_{\mathcal{C}}$	$Identity$	$Morf_{\mathcal{C}}$	Composition
$\mathcal{A}=(Q,I,F,A,\delta_{\mathcal{A}})$	Q	$1_q=(q,\epsilon,q)$	$Morf_{\mathcal{C}}(q,q')=\{(q,a,q'):q'\in\delta_{\mathcal{A}}(q,a)\}$	$(q_1,a,q_2)(q_2,b,q_3)$ $=(q_1,ab,q_3)$
$\mathcal{AS}=(Q,I,F,A,\Gamma,$ $Z_0,\delta_{\mathcal{AS}})$	$Q\times\Gamma$	$1_{(q,\gamma)}$ $=((q,\gamma),\epsilon,(q,\gamma))$	$Morf_{\mathcal{C}}((q,\gamma),(q',\gamma'))=$ $=\{((q,\gamma),a,(q',\gamma')):\exists\alpha\in\Gamma^*:$ $(q',\alpha\gamma)\in\delta_{\mathcal{AS}}(q,\gamma)\}$	$((q_1,\gamma_1),a,(q_2,\gamma_2))$ $.((q_2,\gamma_2),b,(q_3,\gamma_3))$ $=((q_1,\gamma_1),ab,(q_3,\gamma_3))$
$\mathcal{TM}=(Q,A,\Gamma,O,$ $\delta_{\mathcal{T}},\mu_{\mathcal{T}},I,F)$	$Q\times(\Gamma\cup\{\flat\})^k$	$1_{(q,\overline{\gamma},o,\overline{r})}$ $=((q,\overline{\gamma},o,\overline{r}),\epsilon,(q,\overline{\gamma},o,\overline{r}))$	$Morf_{\mathcal{C}}((q,\overline{\gamma}),(q',\overline{\gamma'})$ $=\{((q,\overline{\gamma}),a,(q,\overline{\gamma'})):$ $\exists\overline{r}\in\{R,L,S\}^{(k+1)},o\in O:$ $(q',\overline{\gamma'},o,\overline{r})\in\delta(q,a,\overline{\gamma'})\}$	$((q_1,\overline{\gamma_1}),a,(q_2,\overline{\gamma_2}))\cdot$ $((q_2,\overline{\gamma_2}),b,(q_3,\overline{\gamma_3}))$ $=((q_1,\overline{\gamma_1}),ab,(q_3,\overline{\gamma_3}))$
MF	$\mathcal{P}(ConfM)$	$1_{\sigma_P}=(\sigma_P,NOP,\sigma_P)$	$FM(\sigma_P,\sigma'_P)=$ $=\{(\sigma_P,\vec{c},I,\sigma'_P):\sigma'_P\in I(\vec{c})\}$ I is a k-partial function, $\vec{c}=(c_1,...,c_k)\in ConfM^k$, $c_1,...,c_k\in\sigma_P$ and $I(\vec{c})=\sigma'_P$ If $\sigma_P\nsubseteq\sigma'_P$ then $Morf(\sigma_P,\sigma'_P)=FM(\sigma_P,\sigma'_P)$ If $\sigma_P\subseteq\sigma'_P$ then $Morf(\sigma_P,\sigma'_P)=$ $\{(\sigma_P,NOP,\sigma'_P)\}\cup FM(\sigma_P,\sigma'_P)$	$f\in Morf(\sigma_P,\sigma'_P),g\in Morf(\sigma'_P,\sigma''_P)$ $f:\sigma_P\longrightarrow\sigma'P=(\sigma_P,\vec{c}_f,I_f,\sigma'_P)$ $g:\sigma'_P\longrightarrow\sigma''_P=(\sigma'_P,\vec{c}_g,I_g,\sigma''_P)$ $fg=_{def}$ $=_{def}(\sigma_P,\vec{c}_f,I_f,\sigma'_P)(\sigma'_P,\vec{c}_g,I_g,\sigma''_P)$ $=_{def}(\sigma_P,\vec{c}_f\circ\vec{c}_g,I_g\circ I_f,\sigma''_P)$ $\vec{c}_f\circ\vec{c}_g=_{def}\vec{c}_g\subseteq I_f(\vec{c}_f)$

6 Measures of the Intelligence of a FM

This section is divided in six subsections called respectively:
- *System of units of a Formal Machine.* In this subsection is established a systems of unities that allows to measure, in FMs, several of the concepts associated to intelligence,
- *PCC and ECC,* are measures created for measure respectively the potential and effective (in execution) computational capacity of the machine
- *PCC and ECC in current Formal Computational Systems.* In this subsection are established the PCC and ECC expressions for several FCSs
- *Concrete Automata.* Here is presented a theorem with dedicated expressions under a relation ρ for Finite Automata and are presented concrete Finite Automata, $\mathcal{A}_4$, ..., $\mathcal{A}_{10}$.
- *Calculation of the PCC and ECC for concrete Automata.* Here is calculated the PCC and ECC for some tasks of the Finite Automata whose design is in the previous section and
- *Intelligent measures for Formal Machines.* In this subsection are defined for FMs several concepts that are associated with the idea of intelligence.

Now, I am going to define the function $-$. Let c be a configuration of a FM, $c \in ConfM$. The set of configurations wherein is possible to put the machine from a computation c is called the *set of the computations from* c and is denoted by c_{pos}, $c_{pos} = \cup\{c_P - \{c\} \in \mathcal{P}(ConfM) : c \vdash c_P\}$. The set of the configurations from which the machine can be put in the configuration c is called the *set of computations to* c and denoted by c_{prior}, $c_{prior} = \cup\{c_P \in \mathcal{P}(ConfM) : \exists c'_P \in \mathcal{P}(ConfM) : c \in c'_P, c \notin c_P$ and $c_P \vdash c'_P\}$.

A step of computation $c_P \vdash c'_P$ is called a *self-contained computing step* if $c'_P \subseteq c_P$.

Let $C_i \in CompM_B$, $u, v \in C_i, \alpha_1, a, b, \alpha_2, \beta \in A(C_i)$ with $a \neq b$. I define a function, "-",

$$- : C_i \times C_i \longrightarrow ((A(C_i) \cup \{\epsilon\}) - (A(C_i) \cup \{\epsilon\})) \cup \{0\}$$

of the following manner :
i) $u = \alpha_1 a \alpha_2$, $v = \alpha_1 b \beta$; $u - v = a - b$,
ii) $u = \alpha_1$, $v = \alpha_1 b \beta$; $u - v = \epsilon - b$,
iii) $u = \alpha_1 a \alpha_2$, $v = \alpha_1$; $u - v = a - \epsilon$,
iv) $u = v$; $u - v = \epsilon - \epsilon = 0$.

Let $c_1, c_2 \in ConfM$ be, two configurations, with $c_1 = (c_{11}, ..., c_{1n})$ and $c_2 = (c_{21}, ..., c_{2n})$. The $-(c_1, c_2)$ (by abuse of notation is denoted by $c_1 - c_2$) is defined by

$$c_1 - c_2 = (c_{11} - c_{21}, ..., c_{1n} - c_{2n})$$

6.1 System of Units of a Formal Machine

6.1.1 Fundamental Constitution Units of a Machine

In this section is presented the system of units of a FM. With the aim of obtain an intuitive introduction for the system of units, here present, I treat the formal machines in an anthropomorphic way. So, I am going to start by presenting the fundamental units that serve to measure concepts that are created from, in anthropomorphic context, from the anatomy and physiology of the machine. Here I define the fundamental units that are considered related with the constitution, architecture of the machine and with the relations that the components, of the machine, have with each others. The Fundamental Units are related with the Anatomy and Physiology (FUAP) of the machine.

Table 19: Fundamental Constitution Units of a Machine (FUAP)

Elements	FM notation	Components	
Components	$CompM_B$	$C_1, ..., C_n$	$A(C_i)$
Relations	$CompM_R$	$R_1, ..., R_m$	$\times_{i \in R_j} C'_i$
Configurations	$ConfM$	$c_i = (c_{i1}, ..., c_{in})$	
Instructions	$InstM$	$I \in InstM - \{NOP\}, NOP$	$I(c_P)$
Von Neumann's Algorithm	algorithm operation	$\mathcal{P}_i$	

Of the FUAP makes part, the anatomic units of the machine, $CompM_B$, $CompM_R$, $ConfM$ and the physiologic units of the machine, $InstM$ and the algorithm machine operation, Alg_{VN}. The unities referred as anatomic are associated with the structure and constitution of the machine, the unities referred as physiologic are related with the basic operation of the machine.

6.1.2 Behavior Fundamental Units (BFU)

Now I am going to seek the fundamental units to measure the behavior of a FM. Continuing in the anthropomorphic perspective for FMs, I describe what are the Fundamental Behavioral Units (FBU). In the following I show in a table the unities that characterize the behavior of a formal machine. The FBU are:

Table 20: Behavior Fundamental Units

FBU	Distinct	Total
Occupied space	$Space$	$SpaceT$
Spent time	$Time =$	$TimeT$
Symbols used	$Symbols$	$SymbolsT$
Written carried	$Write$	$WriteT$
Readings taken	$Read$	$ReadT$
Used instructions	$Inst$	$InstT$
Changes in settings	$Movements$	$MovementsT$
Language recognize	L(fM)	

Through of this two types of fundamental units is possible to build a great variety of derived measures. From these fundamental units I build two measures of computational capacity, for formal machines, denoted PCC and ECC. Many other measures can be built based in these fundamental units, depending always about what is intended to measure in a formal machine.

6.2 PCC and ECC

In this section are referred two types of computational measures in FMs, the *Potential Computational Capacity*, shortening PCC, and the *Effective Computational Capacity* shortening ECC. The PCC is a measure that intends measure, quantifying, the computational resources which belong to the machine and is because of this which it depends of the structure of the machine. The ECC intends to measure the type and the quantity of the tasks executed, the resources involved and the efficiency of the executions. The ECC consists in the measurement of the computational process related with the execution of the tasks.

6.2.1 PCC, Potential Computational Capacity

The PCC has into account the structure of the machine, their components and the connections among them, and consists in the counts of those elements. The structure of the machine and the configurations of the machine are divided, for effect of the calculus of the PCC, in part finite and no finite:

1) The set of the components, $CompM_B$, is a finite set of sets, $CompM_B = \{C_1, C_2, ..., C_n\}$[39]. For effect of PCC calculation the set $CompM_B$ is divided in two disjoint parts, $CompM_{Af} = \{C_j \in CompM_B : |C_j| < \infty\}$ and $CompM_{A\infty} = \{C_j \in CompM_B : C_j \notin CompM_{Af}\}$. I build the sets A_f (the Cartesian product of the finite components) and A_∞ (the Cartesian of the infinite components) as follows.

$$A_f = \times\{A \in CompM_B : A \in CompM_{Af}\}, \text{ and}$$

[39]The sets $C_j \in CompM_B$ can or not to be finite

$$A_\infty = \times\{A \in CompM_B : A \notin A_f\}.$$

The division, for effect of construction of the measure, of the components of the machine between the components that are finite sets and the components that are not, is taken in attention because the existence, in a machine, of components that have an infinite quantity of elements has significant influence in their potential computational capacities.

2) Now I analyze the connection among the components. The connection among the components is established by the elements, R_i, of the set $CompM_R$. The existence of connections determines the possibilities to build different instructions. The processing $c_P \vdash c'_P$ depends of the instructions that the machine possesses. The relation ρ that is defined below is a relation involved in the processing of computation and it allows to group the configurations taking into account its participation in a computation process.

The relation, ρ, is a relation of the set of configurations, $ConfM$, and it is defined in the following way. The set of configurations of the machine, $ConfM$, is for definition a subset of Cartesian product, finite, of the components of the machine, $ConfM \subseteq C_1 \times C_2 \times ... \times C_n$. With the purpose of know the cardinal of each one of the delta's sets, Δ, I am going to begin for define a relation, ρ, in the set of the configurations, of the machine, $ConfM$.

Let c, c' be such that $c, c' \in ConfM$, $c\rho c'$ if and only if:
i) ($c_{pos} \neq \emptyset$ and $c'_{pos} \neq \emptyset$) or ($c_{pos} = c'_{pos} = \emptyset$ and $c_{prior} \neq \emptyset$ and $c'_{prior} \neq \emptyset$) or ($c_{pos} = c_{prior} = c'_{pos} = c'_{prior} = \emptyset$), and
ii) for $c_{pos} \neq \emptyset$ and $c'_{pos} \neq \emptyset$, then $c_{pos} - c = c'_{pos} - c'$, and
iii) for $c_{pos} = c'_{pos} = \emptyset$ and $c_{prior} \neq \emptyset$ and $c'_{prior} \neq \emptyset$, then $c - c_{prior} = c' - c'_{prior}$, and
iv) for $c_{pos} = c_{prior} = c'_{pos} = c'_{prior} = \emptyset$, then $c = c'$.

Theorem 6.1 [40]
ρ is an equivalence relation

By the previous theorem and the definition of the relation, ρ, in slight language, exists a partition of the set $ConfM$ related with the execution process of the tasks by the machine.

The set $\Delta_1 = \{[c]_\rho : c \in ConfM_i\}$, $\Delta_2 = \{[c]_\rho : c \in ConfM_f\}$ and $\Delta_3 = \{[c]_\rho : \exists I \in InstM, \pi_{(\vec{u},i)}, c' \in ConfM : c \neq c', c \in ConfM \text{ e } c' \in I \circ \pi_{(\vec{u},i)}(c)\}$, wherein

$$\pi_{(\vec{u},i)} : ConfM \longrightarrow (ConfM)^k, c \mapsto \pi_{(\vec{u},i)}(c), \text{ with}$$
$$\pi_{(\vec{u},i)}(c) = (u_1, ..., u_{i-1}, c, u_{i+1}, ..., u_n)$$

and $\vec{u} = u_1, ..., u_{i-1}, u_{i+1}, ..., u_n$.

Each one of the sets Δ, Δ_i with $i = 1, 2, 3$, is decomposed in finite parts, Δ_{i_f} with $i = 1, 2, 3$, and infinite part, Δ_{i_∞} with $i = 1, 2, 3$. When $|\Delta_i| < \aleph_0$, then $\Delta_{i_f} = \Delta_i$ and $\Delta_{i_\infty} = 0.5$ (for convention). When $|\Delta_i| \geq \aleph_0$, then $\Delta_{i_f} = 0.5$ (for

[40]The proof of the theorem consists of algebraic manipulations and is left to the reader

convention) and $\Delta_{i_\infty} = \Delta_i$[41].

The delta's sets are doing a classification of the components based in the behavior that the configurations have in the computational process. That delta's sets are built through of the relation ρ that is an equivalent relation. In that sense, the elements of Δ_1, Δ_2 and Δ_3 are sets in $ConfM\ mod\rho$.

Thus, the counting of the elements of distinct initial configurations, $|\Delta_1|$, of the distinct final configurations, $|\Delta_2|$, and of the distinct configurations from which is allowed to effect changes of configuration, $|\Delta_3|$ are made $mod\rho$.

The Δ_1, Δ_2 are important because communication with the outer is important. In Δ_1 there are the initial configurations $mod\rho$ and in Δ_2 there are the final configurations $mod\rho$. Δ_1 is related with the capacity to listen the outer, Δ_2 is related with the capacity to action over the outer. Δ_3 is important because it is related with the capacity of make processing. In Δ_3 there are the configurations that allow to give a step of computation. An execution is a process that is carried out step by step.

The importance of the delta's sets are the following. In the case of Δ_1 and Δ_2, they are important because communication is important, the channels of communication, which the machine possesses for communicate with the outer are important. The importance of the distinct configurations $mod\rho$ is that Δ_3 allows to continue and end a computation. Thus, Δ_3 allows to process tasks.

The PCC is defined as,

$$PCC_f = log_6(2^{|A_f|} \times 3^{\nabla_f}), \text{ with } \nabla_f = |\Delta_{1_f}| \times |\Delta_{2_f}| \times |\Delta_{3_f}|$$
$$PCC_\infty = |A_\infty| + \nabla_\infty, \text{ with } \nabla_\infty = |\Delta_{1_\infty}| \times |\Delta_{2_\infty}| \times |\Delta_{3_\infty}|$$ [42]
$$PCC = (PCC_f, PCC_\infty)$$

Note 6.1 *The PCC is a measure constituted by two parts, a finite part, PCC_f, and an infinite part PCC_∞.*

The PCC_f allows to measure the potential capacity and it is obtained through of the finite components of the machine. A_f consists in the entirety of configurations, with the finite components of the machine, which the machine can withstand. ∇_f consists entirely of distinct configurations modρ and is taken into account the type of configuration which the machine has. The PCC_f is an algorithmic measure, the use of log_6 has the only aim to lowering the values of A_f and ∇_f in magnitude. To $|A_f|$ and $|A_\infty|$ are assigned different weights in the measure.

[41]The chooses for convention of the value $0,5$ is made because is necessary to penalize the fact of certain Δ have not elements and this penalization should be done without which that provoke the annulment of ($\nabla_\bullet = \Delta_{1\bullet} \times \Delta_{2\bullet} \times \Delta_{3\bullet}$).

In a care observation of the sets Δ can be seen which the delta's sets have elements in common. That fact can originate duplicated counts for a certain ρ-class. Can occur, for example, that a configuration is classified at the same time in Δ_1 and in Δ_2. That occur because it is intended to give importance to that type of configurations

[42]Can occur that the value of PCC_∞ let be a transfinite cardinal number, $\aleph_0$,$\aleph_1$,$\aleph_2$, and so on ... The sum that occur in PCC_∞ is calculated in transfinite arithmetic. For a more comfortable manipulation of the measures, the exposed theory includes the Set Theory of Zermelo-Fraenkel, ZF, with the countinuum hypothesis as true, CH, ZFC=ZF+CH.

The PCC_∞ measures the potential capacity obtained through of the infinite components. The infinite part is the part more important of the potential capacity and that is taken into attention in the definition of the partial order $\leq_{PCC}$ presented later.

Definition 6.1 *The constant $p = \frac{log_6(3)}{log_6(2)} \approx 1,340574$ is called the potential constant and is denoted by p.*

The potential constant allows to determine and to quantify, what means the relative importance between $|A_f|$ and ∇_f. $|A_f|$ has an importance of $1,3$ higher than the ∇_f.

Theorem 6.2 [43]
i) If $|A_f| = \nabla_f$, then $PCC_f = (\nabla_f)^2$
ii) $PCC_f = |A_f|\ log_6(2) + \nabla_f\ log_6(3)$
iii) $\nabla_f = 0,63093\ |A_f|$ (approximate)
iv) If $\Delta_{1_\infty} = \Delta_{2_\infty} = \Delta_{3_\infty} = \emptyset$, then $PCC_\infty = |A_\infty| + 0,125$
v) $PCC_\infty \geq \aleph_0$ or $PCC_\infty = 0,125$
vi) If $A_\infty = \emptyset$, then $PCC_\infty = \nabla_\infty$
vii) If $A_\infty = \Delta_{1_\infty} = \Delta_{2_\infty} = \Delta_{3_\infty} = \emptyset$, then $PCC_\infty = 0,125$

In the set or class of all FMs, which is denoted by $\mathbb{Z}$eus, I define an order relation, $\leq_{PCC}$ that easily can be proved to be a partial order. Let fM$_1$ and fM$_2$ be two FMs.

$$\begin{gathered}\text{fM}_1 \leq_{PCC} \text{fM}_2 \text{ if} \\ (PCC_\infty(\text{fM}_1) \leq PCC_\infty(\text{fM}_2)) \text{ or} \\ (PCC_\infty(\text{fM}_1) = PCC_\infty(\text{fM}_2) \text{ and } PCC_f(\text{fM}_1) \leq PCC_f(\text{fM}_2))\end{gathered}$$

I define

$$\begin{gathered}\text{fM}_1 <_{PCC} \text{fM}_2 \text{ if} \\ PCC(\text{fM}_1) \leq PCC(\text{fM}_2) \text{ and } PCC(\text{fM}_1) \neq PCC(\text{fM}_2)\end{gathered}$$

It is said that the machine fM$_2$ has more (respectively equal) potential computation capacity than fM$_1$ if fM$_1$ $<_{PCC}$ fM$_2$ (respectively $PCC_\infty(\text{fM}_1) = PCC_\infty(\text{fM}_2)$ and $PCC_f(\text{fM}_1) = PCC_f(\text{fM}_2)$).

The measure PCC in its infinite component, PCC_∞, serves to define *Machine with Super Resources* (MSR)[44]. Thus, the definition of MSR is a definition in potential sense. A MSR is a FM wherein $PCC_\infty \geq \aleph_1$. An example of a type of formal machine that is a Supermachine is the recurrent neural networks such as defined by Siegelmann and Sontag [SiS94]. Sielgelmann and Sontage proved which those computational models are more potent than Turing Machines.

[43]The prove of the theorem results of simple algebraic manipulations

[44]It is used, sometimes, the expression Supermachine as designated Machine with Super Resources

Theorem 6.3 *The recurrent neural networks (RNN) are Supermachines*
Proof: It is proved in Theorem 4.2, page 42, that the RNNs are FMs. Can be seen in page 54 that the RNN as a FM has the set of components $CompM_B = \{\vec{U}, Neu, T\}$ *where* $Neu = \overbrace{\mathbb{R} \times ... \times \mathbb{R}}_{N \text{ times}}$. *Thus,* $|Neu| = \aleph_1$. *Then* $PCC_\infty \geq \aleph_1$.

6.2.2 ECC, Effective Computational Capacity

For the construction of the ECC as a measure of computational performance, it is necessary to know what kind of resources are involved in the computing process. To do this I start to define a function $\chi : \mathbb{N}_0 \longrightarrow \{0,1\}$, $\chi(0) = 0$ and for $n \geq 1, \chi(n) = 1$.

The ECC is a measure of computational power of the machine in the amount and complexity (in the length of the words, that are tasks, $|\tau\rfloor$) of tasks which the machine carries out and also it allows to know how the machine carries out them. The ECC is measured globally and locally, respectively ECC_{global} and ECC_{local}. The ECC global measurement consists on to measure, for each task that is carried out, the total amount of different resources, of the machine, that are used. The ECC local measurement consists on to measure, for each task that is carried out, the total amount of machine resources used per machine cycle. Without to lose generality, I take the execution, $e = c_{(0,P)} \vdash c_{(1,P)} \vdash ... \vdash c_{(r,P)}$, with $c_{i,P} \in ConfM$, of a task, $\tau = (c_{(0,P)}, c_{(r,P)})$. The resource involved are:

i)- the space occupied, $Space(\tau, e)$, while is running, e, the task τ

$$Space(\tau, e) = \sum_{i=1}^{n} max\{|u\rfloor | \exists c \in \cup ConfM(e) : u = c(C_i) \text{ and } (u \in (A(C_i))^+ \text{ or } u = \epsilon)\}$$ [45]

ii)- the time spent, $Time(\tau, e)$, while is running, e, the task τ.

$Time(\tau, e)$ = quantity of the Von Neumann's Cycle, VNC, in the execution e, of the task τ

iii)- the total quantity of distinct symbols used, $Symbols(\tau, e)$, while is running, e, the task τ.

$$Symbols(\tau, e) = |\{a : (\exists 1 \leq i \leq n)(\exists c_P \in ConfM(e))(\exists \vec{c} = (c_1, ..., c_n) \in ConfM) | a \in A(C_i), c \in c_P \text{ and } \chi(|c_i\rfloor_a) = 1\}|$$

iv)- the quantity of distinct changes in the configurations, $Movements(\tau, e)$, while is running, e, the task τ.

$$Ind(\tau, e) = \{(c, c') \in ConfM \times ConfM | \exists (c_P, c'_P) \in ConfM(e) \times ConfM(e) : c \in c_P, c' \in c'_P, c \notin c'_P, ^{\prime}_{,} \notin c_P, \text{ and } c_P \vdash c'_P \text{ in } e\}$$
$$Movements(\tau, e) = \sum_{(c,c') \in Ind(\tau,e)} \sum_{i=1}^{n} \chi(|c_i - c'_i|)$$

[45]The alphabet of the component C_i is the code $A(C_i)$, the counts of $|u\rfloor$ is carried out in $A(C_i)$

v)- the quantity of distinct reads of the state of the machine, $Read(\tau,e)$, while is running, e, the task τ.

$$Read(\tau,e) = |c_{(0,P)} \cup (\cup_{i=0}^{r-1}(c_{(i+1,P)} - c_{(i,P)}))|,$$

is considered to exist a read in a step of computation if $c_P \vdash c'_P$ and $c'_P - c_P \neq \emptyset$

vi)- the quantity of distinct instructions and distinct applications of the instructions of the program, $Inst(\tau,e)$, while is running, e, the task τ.

$$Inst(\tau,e) = |\{(\vec{c},I,c'_P) \in (ConfM)^k \times InstM \times ConfM(e) | \exists c_P \in ConfM(e) : \vec{c} = (c_1,...,c_n), c_i \in c_P(\text{for } i = 1,...,n), c'_P = I(\vec{c}) \text{ and } c_P \vdash c'_P\}|$$

vii)- the quantity of distinct written of the state of the machine, $Write(\tau,e)$, while is running, e, the task τ.

$$Write(\tau,e) = | \cup_{i=1}^{r} (c_{(i,P)})|,$$

From the $Time(\tau,e)$ (respectively, $Space(\tau,e)$, $Symbols(\tau,e)$, $Movements(\tau,e)$, $Read(\tau,e)$, $Inst(\tau,e)$), $Write(\tau,e)$. I construct $T(\tau,e)$ (respectively and in analog form, $Sp(\tau,e)$, $S(\tau,e)$, $M(\tau,e)$, $R(\tau,e)$, $Inst(\tau,e)$, $Write(\tau,e)$) of the following way

$$T(\tau,e) = \begin{cases} \frac{1}{Time(\tau,e)} & \text{if} \quad 0 < Time(\tau,e) < \aleph_0, \\ 2 & \text{if} \quad Time(\tau,e) = 0, \\ 0 & \text{if} \quad Time(\tau,e) \geq \aleph_0 \end{cases}$$

In slight language the $ECC_{global}(\tau,e)$ is the sum of the inverse quantity of [46] each one of the different resources used during the execution process, e, of the task τ, multiplied by the inverse of the factorial of the length of τ. Thus, in $ECC_{global}(\tau,e)$ are measured the resources involved during the execution process, e, of the task τ.

$$ECC_{global}(\tau,e) = \frac{1}{|\tau]!}(T(\tau,e)+Sp(\tau,e)+S(\tau,e)+M(\tau,e)+R(\tau,e)+I(\tau,e)+W(\tau,e)).$$

Then I am going to define what is $ECC_{global}(\tau)$. The way as it will be defined, allows for assign to $ECC_{global}(\tau)$ the value of $ECC_{global}(\tau,e)$ wherein the execution, e, that carries out the task τ is the most efficient. Thus the value of $ECC_{global}(\tau)$ is the value of the most efficiency task, τ, carried out. This efficiency is measure through of the use of the least machine resources. In comparison, between two formal machines, the largest $ECC(\tau)$ indicates which one is the machine that carries out the task with more efficiency, and it is one that uses fewer resources. The $ECC_{global}(\tau)$ has into attention how the machine carries out τ.

[46] note which in general $T(\tau,e) = \frac{1}{Time(\tau,e)}$ and analogously for the other measures

$$ECC_{global}(\tau) = max\{ECC_{global}(\tau,e) : e \text{ is an execution of } \tau\},$$

the way as $ECC_{global}(\tau)$ is defined allows to choose for value of the measure, among different executions of the task τ, one that uses fewer resources.

$$ECC_{global} = \sum_{\tau \in L} ECC_{global}(\tau)$$

This allows which for the characteristics already referred of the ECC_{global} it can be added the amount and complexity (in terms of length of task, $|\tau\rfloor$) of tasks that the machine is able of carry out. This measure assigns more importance to the execution of simple tasks, and the machine that has more high value of the ECC_{global} is that one which carries out the more simple task and that carries out that in the most efficient way. Is because of this which must be done a factor analysis of this measure (section 6.2.3)[47].

The ECC_{global} for some formal machine is designated as the *global Effective Computational Capacity* of the machine and is a measure for all tasks that the machine is able to carry out, in their construction are chosen the executions that use fewer resources.

Now, I am going to define the local measure of ECC, the ECC_{local}. In the ECC_{local} only are taken in consideration the process of execution, e, of tasks τ wherein $0 < Time(\tau) < \aleph_0$.

In the execution, e, of a task, τ, the resources of local nature that should be consider are:

i)- the total quantity of the symbols, $SymbolsT(\tau,e)$, used while is running, e, the task τ. $Symbols_{VN}(\tau,e) = \frac{SymbolsT(\tau,e)}{Time(\tau,e)}$

$$SymbolsT(\tau,e) = \sum_{c \in \cup ConfM(e)} \sum_{i=1}^{n} |c_i\rfloor$$

ii)- the total quantity of shifts in the configurations, $MovementsT(\tau,e)$, used while is running, e, the task τ. $Movements_{VN}(\tau,e) = \frac{MovementsT(\tau,e)}{Time(\tau,e)}$

$$MovementsT(\tau,e) = \sum_{j=1}^{n} \sum_{(c,c') \in c_{(j-1,P)} \times c_{(j,P)}} \sum_{i=1}^{n} \chi(|c'_i - c_i|)$$

iii)- the total quantity of reads of the state of the machine, $ReadT(\tau,e)$, used while is running, e, the task τ. $Read_{VN}(\tau,e) = \frac{ReadT(\tau,e)}{Time(\tau,e)}$

$$ReadT(\tau,e) = |c_{(0,P)}| + \sum_{i=0}^{n-1} |c_{(i+1,P)} - c_{(i,P)}|$$

iv)- the total quantity of program instructions, $InstT(\tau,e)$, used while is running, e, the task τ. $Inst_{VN}(\tau,e) = \frac{InstT(\tau,e)}{Time(\tau,e)}$

$$InstT(\tau,e) = |e|_{\vdash} = \text{ quantity of occurrences on } e \text{ of the symbol } \vdash$$

[47]The ECC value the realization of simply tasks and is necessary an evaluation for the capacity to perform complex tasks

The set $ConfM \cup \{\vdash\}$ can be taken as the alphabet of the execution processes, e, of the tasks. Thus, $|e|_{\vdash}$ is measured on $ConfM \cup \{\vdash\}$.

From the $Symbols_{VN}(\tau,e)$ (respectively, $Movements_{VN}(\tau,e)$, $Read_{VN}(\tau,e)$, $Inst_{VN}(\tau,e)$) is constructed $S_{VN}(\tau,e)$ (respectively and in analog form, $M_{VN}(\tau,e)$, $R_{VN}(\tau,e)$, $I_{VN}(\tau,e)$) of the following way

$$S_{VN}(\tau,e) = \begin{cases} \frac{1}{Symbols_{VN}(\tau,e)} & \text{if} \quad 0 < Symbols_{VN}(\tau,e) < \aleph_0, \\ 0,5 * T(\tau,e) & \text{if} \quad Symbols_{VN}(\tau,e) = 0, \\ 0 & \text{if} \quad Symbols_{VN}(\tau,e) \geq \aleph_0 \end{cases}$$

Thus, $ECC_{local}(\tau,e)$, is

$$ECC_{local}(\tau,e) = \frac{1}{|\tau]!}(S_{VN}(\tau,e) + M_{VN}(\tau,e) + R_{VN}(\tau,e) + I_{VN}(\tau,e))$$

The ECC_{local} measures the total amount of resources used by the Von Neumann's Cycle multiplied by the inverse of the factorial of τ. ECC_{local} is defined as:

$$ECC_{local}(\tau) = max\{ECC_{local}(\tau,e) : e \text{ is an execution of } \tau\}.$$

The $CEE_{local}(\tau)$ as is defined allows to take for value of the measure the value obtained in the running, e, of the task τ that use less resources in the Von Neumann's Cycle.

Now, the ECC_{local} is defined as:

$$ECC_{local} = \sum_{\tau \in L} ECC_{local}(\tau)$$

As the ECC_{local} is the sum of $ECC_{local}(\tau)$ for all τ carried out by the machine it has in consideration the quantity of the tasks that the machine carries out. The ECC_{local} of a FM is designated as *local Effective Computational Capacity* of the machine.

Theorem 6.4 [48]
i) $ECC_{global}(\tau) \leq \frac{12}{|\tau]!}$
ii) $ECC_{local}(\tau) \leq \frac{8}{|\tau]!}Time(\tau,e)$, *for an execution process e of the task* τ *such that* $0 < Time(\tau,e) < \aleph_0$
iii) If $A_1, A_2, ..., A_n$ *are finite and* $r = |A_1 \times A_2 \times ... \times A_n|$ *com* $A_i = A(C_i)$, *then*
- $ECC_{global} \leq \sum_{n=0}^{\infty} 12\frac{r^n}{n!} \leq 12e^r$
- $ECC_{local} \leq \sum_{n=0}^{\infty} 8\frac{Time(\tau,e)r^n}{n!}$, *for an execution process e of the task* τ *such that* $0 < Time(\tau,e) < \aleph_0$.

[48] The prove of the theorem results of simple algebraic manipulations

In $\mathbb{Z}eus$, I define two order partial relations related with the ECC, $\leq_{ECC_{local}}$ and $\leq_{ECC_{global}}$. Let fM_1, fM_2 be FMs.

I define $\leq_{ECC_{global}}$ as,

$$\mathrm{fM}_1 \leq_{ECC_{global}} \mathrm{fM}_2 \text{ if}$$
$$ECC_{global}(\mathrm{fM}_1) \leq_{ECC} ECC_{global}(\mathrm{fM}_2)$$

and I define $\leq_{ECC_{local}}$ as,

$$\mathrm{fM}_1 \leq_{ECC_{local}} \mathrm{fM}_2 \text{ if}$$
$$ECC_{local}(\mathrm{fM}_1) \leq_{ECC} ECC_{local}(\mathrm{fM}_2)$$

In this situation is said that fM_2 has *global Effective Computational Capacity* higher than fM_1. fM_2 (respectively fM_1) is more (respectively less) *globally enable* that fM_1 (respectively fM_2).

I define $<_{ECC_{local}}$ as,

$$\mathrm{fM}_1 <_{ECC_{local}} \mathrm{fM}_2 \text{ if}$$
$$\mathrm{fM}_1 \leq_{ECC_{local}} \mathrm{fM}_2 \text{ and } ECC_{local}(\mathrm{fM}_1) \neq ECC_{local}(\mathrm{fM}_2)$$

In this situation is said that fM_2 has *local Effective Computational Capacity* higher than fM_1. fM_2 (respectively fM_1) is more (respectively less) *locally enable* than fM_1 (respectively fM_2).

Thus, in $\mathbb{Z}eus$ was defined an order partial relation, $\leq_{PCC}$, $\leq_{ECC_{local}}$ and $\leq_{ECC_{global}}$. Those relations allow to a classification through of the ordination of the formal formal machines by, respectively, Potential Computational Capacity, local and global Effective Computational Capacity.

6.2.3 Factorial analysis of the measures

To define the notions that allows to effect a factorial analysis I am going to use, in ECC, the known operators **"big O", "big Ω" and "little o"**. This way of proceed, is similar that one used in the Theory of Complexity. These operators which can be generalized to measures which there are or will be created in FMs. Let fM be a FM.

Let $G : \mathbb{N} \longrightarrow \mathbb{R}$ be, with $G(\mathbb{N}) \subseteq [0, +\infty[$, a total function and

$$ECC_{global}(n) = \{EEC_{global}(\tau) : |\tau\rfloor = n \text{ and } \tau \in ConfM_i \times ConfM_f \text{ and } \tau \in \mathcal{L}(\mathrm{fM})\}$$
(respectively,
$$ECC_{local}(n) = \{EEC_{local}(\tau) : |\tau\rfloor = n \text{ and } \tau \in ConfM_i \times ConfM_f \text{ and } \tau \in \mathcal{L}(\mathrm{fM})\}).$$

It is said that fM has a ECC_{global} (respectively ECC_{local}) *with magnitude order* of G, denoted $ECC_{global}(n) = o(G(n))$ (respectively, $ECC_{local}(n) = o(G(n))$), if

$$lim_{n\to\infty} \frac{ECC_{global}(n)}{G(n)} = k \text{ with } k \in]0, +\infty[.$$
$$\text{(respectively, } lim_{n\to\infty} \frac{ECC_{local}(n)}{G(n)} = k \text{ with } k \in]0, +\infty[.)$$

It is said that fM has ECC_{global} (respectively, ECC_{local}) with *inferior magnitude* to G, denoted $ECC_{global}(n) = O(G(n))$ (respectively, $ECC_{local}(n) = O(G(n))$, if

$$lim_{n\to\infty}\frac{ECC_{global}(n)}{G(n)} = 0 \text{ (respectively, } lim_{n\to\infty}\frac{ECC_{local}(n)}{G(n)} = 0\text{)}$$

It is said that fM has ECC_{global} (respectively, ECC_{local}) with *higher magnitude* to G, denoted $ECC_{global}(n) = \Omega(G(n))$ (respectively, $ECC_{local}(n) = \Omega(G(n))$, if

$$lim_{n\to\infty}\frac{ECC_{global}(n)}{\Omega(G(n))} = +\infty \text{ (respectively, } lim_{n\to\infty}\frac{ECC_{local}(n)}{\Omega(G(n))} = +\infty).$$

The FMs can sorted through of the operators O and Ω. Let fM_1, fM_2 be FMs. It is said that fM_1 has, in complexity, inferior ECC_{global} (respectively, ECC_{local}) magnitude than fM_2, denoted $fM_1 \leq^{O}_{ECC_{global}} fM_2$ (respectively, $fM_1 \leq^{O}_{ECC_{local}} fM_2$) if $ECC_{global}(fM_1)(n) = O(ECC_{global}(fM_2)(n))$ (respectively, $ECC_{local}(fM_1)(n) = O(ECC_{local}(fM_2(n))))$.

fM_1 has, in complexity, higher ECC_{global} (respectively, ECC_{local}) magnitude than fM_2, denoted $fM_2 \leq^{\Omega}_{ECC_{global}} fM_1$ (resp, $fM_2 \leq^{\Omega}_{ECC_{global}} fM_1$) if ECC_{global} $(fM_1)(n)$ $=\Omega(ECC_{global}(fM_2)(n))$ (respectively, $ECC_{local}(fM_1)(n)=\Omega(\ ECC_{local}(\ fM_2(n))))$.

Theorem 6.5 *The relations $\leq^{O}_{ECC_{global}}$, $\leq^{O}_{ECC_{local}}$, $\leq^{\Omega}_{ECC_{global}}$ and $\leq^{\Omega}_{ECC_{local}}$ are anti-symmetry and transitive in $\mathbb{Z}eus$*[49].

A factorial analysis of the measures ECC can be done through of a factorial scale. To do this is sufficient to take $G(n) = \frac{1}{n!}$ and this allows to have a way to classify the machines in the factorial scale $\{1, \frac{1}{2!}, \frac{1}{3!}, ..., \frac{1}{n!}, \frac{1}{(n+1)!}, ...\}$.

This classification, between the machines, allows to sort the machines by its efficiency in complex tasks. The classification of the FMs through of the ECC favors an analysis of the efficiency of the machine as the capacity to perform simple tasks. Therefore, is necessary to do a factorial analysis of the ECC to obtain a classification of the machines by its efficiency in to execute complex tasks.

6.3 ECC and PCC in Current Formal Computational Systems

In this subsection are presented the measures PCC and ECC for the k-Turing Machines, Pushdown Automata, Transducers, Finite Automata and k- Unbounded Registers Machine. In this subsection for a more easy approach, each one of the FMs are called by the name of the machines from which they are obtained. Thus, for example, the FM obtained from a k-Turing Machine is called, for abuse of language, a k-Turing Machine.

[49]The prove of the theorem consists of algebraic manipulations and is left to the reader

Except the k-Unbounded Registers Machine, in all others machines, that have been referred, happen that $\Delta_{1_\infty} = \Delta_{2_\infty} = \Delta_{3_\infty} = \emptyset$, and $|A_\infty| = \aleph_0$. Then of each one of that machines $PCC_\infty = \aleph_0$. Thus, the PCC_∞ is $\aleph_0$ and the ECC measure is obtained for each one of the FCSs values for $Read_{VN}$ and $InstM_{VN}$. In Turing Machines and Transducers in each VNC they are 3 uses of the operator $\vdash$[50].

i) PCC an ECC of a k-Turing Machine

$$A_f = Q, A_\infty = T_I \times (T_1 \times \ldots \times T_k) \times T_O \times p_I \times (p_1 \times \ldots \times p_k) \times p_O$$

A configuration of the set of a k-Turing Machine is an element:

$$Q \times A\mathrm{b}^\omega \times (\Gamma \mathrm{b}^\omega)^k \times O\mathrm{b}^\omega \times \mathbb{N} \times \mathbb{N}^k \times \mathbb{N}$$
$$(q, Z_0 u \mathrm{b}^w, (Z_0 \gamma_1 \mathrm{b}^w, \ldots, Z_0 \gamma_k \mathrm{b}^w), Z_0 o \mathrm{b}^w, n_0, (n_1, \ldots, n_k), n_O)$$

The PCC_f is:
$PCC_f = |Q| log_6(2) + \nabla_f log_6(3)$, and $PCC_\infty = \aleph_0$

To local and global ECC is possible to get, in general, the possible components for a machine. All others must be obtained in each concrete machine. Let $\tau = (\tau_I, \tau_O)$ be a task and e an execution of τ. $Read_{VN}(\tau, e) = 1$ and $Inst_{VN}(\tau, e) = 3$.

ii) PCC and ECC of a Pushdown Automata

$$A_f = Q, A_\infty = T_I \times p_I \times P$$

The set of configurations of a Pushdown Automata is:

$$Q \times A\mathrm{b}^\omega \times \Gamma \times \mathbb{N} \times \mathbb{N}$$

A configuration of the machine is an element

$$(q, Z_0 u \mathrm{b}^w, Z_0 \gamma, n_0, n_P)$$

$PCC_f = |Q| log_6(2) + \nabla_f log_6(3)$, and $PCC_\infty = \aleph_0$

To local and global ECC is possible to get, in general, the possible components of a machine. All others must be obtained in each concrete machine. Let $\tau = \tau_I$ be a task and e an execution of τ. $Time(\tau, e) = |\tau_I(T_I)\rfloor$, $Read_{VN}(\tau, e) = 1$ and $Inst_{VN}(\tau, e) = 1$

iii) PCC and ECC of a Transducer

$$A_f = Q, A_\infty = T_I \times p_I \times T_O \times p_O$$

The set of configurations of the machine is a transducer,

$$Q \times A\mathrm{b}^\omega \times O\mathrm{b}^\omega \times \mathbb{N} \times \mathbb{N}.$$

[50]See the definition of a Turing Machines and a Transducer, in [Hop08], and observe that they have two functions for work with the steps of computation. That is the reason for $InstM_{VN} = 3$. You can see in the same book the others FCSs and to repair that each one of them has only one function to operate with the steps of computation that. This is the reason for $InstM_{VN} = 1$.

A configuration of the machine is an element

$$(q, Z_0 u\text{ƀ}^w, Z_0 o\text{ƀ}^w, n_0, n_o)$$

$PCC_f = |Q|log_6(2) + \nabla_f log_6(3)$, e $PCC_\infty = \aleph_0$

To local and global ECC is possible to get, in general, the possible components of a machine. All others must be obtained in each concrete machine. Let $\tau = (\tau_I, \tau_O)$ be a task and e an execution τ. $Time(\tau, e) = |\tau_I(T_I)\rfloor$, $Read_{VN}(\tau, e) = 1$ e $Inst_{VN}(\tau, e) = 3$.

iv) PCC and ECC of Finite Automata

$$A_f = Q, A_\infty = T_I \times p_I$$

The set of configurations of the machine of a Finite Automata ,

$$Q \times A\text{ƀ}^\omega \times \mathbb{N}.$$

A configuration of the machine is an element

$$(q, Z_0 a\text{ƀ}^w, n)$$

$PCC_f = |Q|log_6(2) + \nabla_f log_6(3)$, e $PCC_\infty = \aleph_0$

To local and global ECC is possible to get, in general, the possible components of a machine. All others must be obtained in each concrete machine. Let $\tau = (\tau_I, \tau_O)$ a task and e an execution of τ. $Time(\tau, e) = |\tau_I(T_I)\rfloor$, $Read_{VN}(\tau, e) = 1$ e $Inst_{VN}(\tau, e) = 1$

v) PCC and ECC of a k-Unbounded Register Machine

$$A_f = \emptyset, A_\infty = I \times p_I \times R_0 \times (R_1 \times \ldots \times R_k)$$

The set of configurations of the machine of a k-Unbounded Register Machine, $I \times \mathbb{N} \times \mathbb{N}^3 \times \times_{i=1}^{k} \mathbb{N}$. A configuration of the machine is an element

$$(I_1 \ldots I_n, (n_{00}, n_{10}, n_{20}), r_1, \ldots, r_k)$$

$PCC_f = \nabla_f log_6(3)$, e $PCC_\infty = \aleph_0$

To local and global ECC is possible to get, in general, the possible components of a machine. All others must be obtained in each concrete machine. Let $\tau = (\tau_I, \tau_O)$ a task and e an execution process of τ. $Read_{VN}(\tau, e) = 1$ e $Inst_{VN}(\tau, e) = 1$.

6.4 Concrete Automata

In this subsection I present several finite automata represented in directed graphs. From the definition of a finite automaton follows that there is always a transition $(q, \text{ƀ}, q)$ for any state q. In the representations of Automata, which follow, it is understood, to be common practice, not put in every state, q, a loop representing the configuration $(q, \flat, q)$. The existence of these configurations are seen implicitly. Let e be an execution of a task τ in a finite automaton. Let $c \in ConfM(e)$ be a configuration of e such that $c = (q, Z_0 u \flat^w, n)$. As $c_{pos} = \{c' : c \vdash c'\}$ (respectively, $c_{prior} = \{c' : c' \vdash c\}$), then $c_{pos} = \{(q', Z_0 u \flat^w, n+1)\}$ or $c_{pos} = \emptyset$ (respectively, $c_{prior} = \{(q', Z_0 u \flat^w, n-1)\}$ or $c_{prior} = \emptyset$). Thus the relation ρ in finite automata can be rewritten by the following theorem.

Theorem 6.6 [51]
$c\rho c'$ if only if
i) if $|c_{pos}| = |c'_{pos}| = 1$, then $c_{pos}(Q) - c(Q) = c'_{pos}(Q) - c'(Q)$ or
ii) if $|c_{post}| = |c'_{post}| = 0$ and $|c_{prior}| = |c'_{prior}| = 1$, then $c(Q) - c_{prior}(Q) = c'(Q) - c'_{prior}(Q)$ or
iii) if $|c_{post}| = |c'_{post}| = |c_{prior}| = |c'_{prior}| = 0$, then $c = c'$.

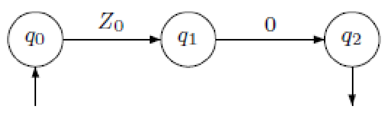


Figure 49: $\mathcal{A}_4$

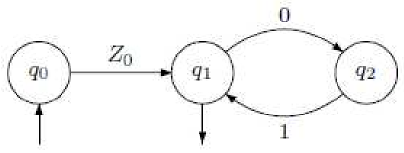


Figure 50: $\mathcal{A}_5$

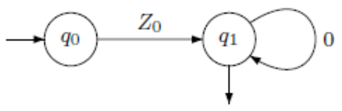


Figure 51: $\mathcal{A}_6$

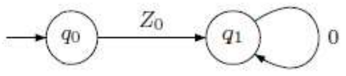


Figure 52: $\mathcal{A}_7$

[51] The prove of the theorem consists of algebraic manipulations and is left to the reader

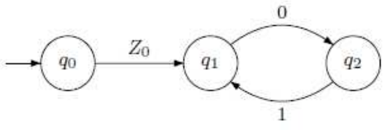


Figure 53: $\mathcal{A}_8$

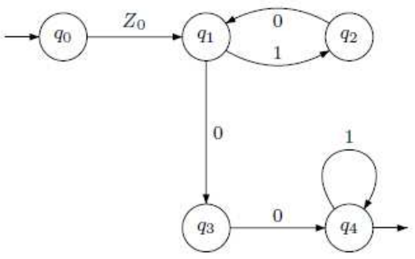


Figure 54: $\mathcal{A}_9$

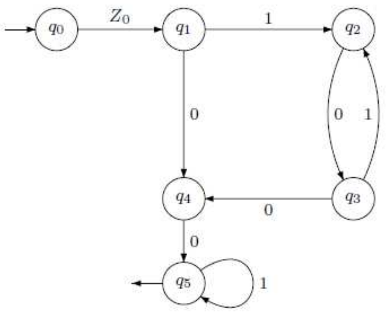


Figure 55: $\mathcal{A}_{10}$

6.5 Calculation of the PCC and ECC for concrete Automata

The relation ρ in automata is characterized by the set Q. I Take the automaton $\mathcal{A}_{10}$, $\Delta_1 = \{q_1 - q_0\}$, $\Delta_2 = \{q_5 - q_5 = 0, q_5 - q_4\}$ and $\Delta_3 = \{q_1 - q_0, q_2 - q_1, q_3 - q_2, q_2 - q_3, q_4 - q_3, q_4 - q_1, q_5 - q_4, q_5 - q_5 = 0\}$. Thus, $|\Delta_1| = 1$, $|\Delta_2| = 1$ and $|\Delta_3| = 8$.

Table 21: Calculation the $PCC(\mathcal{A})$

Automata, $\mathcal{A}$	$\lvert A_f\rvert$	$\lvert\Delta_{1_f}\rvert$	$\lvert\Delta_{2_f}\rvert$	$\lvert\Delta_{3_f}\rvert$	∇_f	$PCC_f(\mathcal{A})$	$PCC_\infty(\mathcal{A})$
$\mathcal{A}_4$	3	1	1	2	2	2,357525045	$\aleph_0$
$\mathcal{A}_5$	3	1	2	3	6	4,582735049	$\aleph_0$
$\mathcal{A}_6$	2	1	2	2	4	3,055156699	$\aleph_0$
$\mathcal{A}_7$	2	1	0,5	2	1	1,386249197	$\aleph_0$
$\mathcal{A}_8$	3	1	0,5	3	1,5	2,079373795	$\aleph_0$
$\mathcal{A}_9$	5	1	2	6	12	8,750496749	$\aleph_0$
$\mathcal{A}_{10}$	6	1	2	8	16	11,3906801	$\aleph_0$

In the following table are presented the values of the components of the $ECC_{global}(\tau, e)$ of some words τ, $\tau = 001^n$ with $n = 0, 1, 2, 3, 12, 13$, carried out by the automata $\mathcal{A}_9$ and $\mathcal{A}_{10}$. It is exemplified the counts of the components in ECC through of the description of the computation of the task Z_0001.

$e = (q_0, Z_0001b^w, 1) \vdash (q_1, Z_0001b^w, 2) \vdash (q_4, Z_0001b^w, 3) \vdash (q_5, Z_0001b^w, 4) \vdash (q_5, Z_0001b^w, 5)$ (computation in $\mathcal{A}_{10}$)

Table 22: Calculation the $ECCC_{global}(\tau)$

$\mathcal{A}_9, \mathcal{A}_{10}, \tau$	$Read$	$Movements$	$Time$	$Space$	$Symbols$	$InstM$	$ECC_{global}(\tau)$
Z_000	3	3	3	6	12	3	0,065972222
Z_0001	4	4	4	7	13	4	0,010164835
Z_00011	5	4	5	8	14	5	0,001453373
Z_000111	6	4	6	9	15	6	0,000184083
$Z_000\underbrace{1...1}_{12}$	16	4	15	19	18	15	4,23669E-13
$Z_000\underbrace{1...1}_{13}$	16	4	16	20	18	16	2,59552E-14

For the same tasks and the same automata I am building the table which is following for obtain the measure $ECC_{local}(\tau, e)$

Table 23: Calculation the $ECC_{local}(\tau)$

(1)	(2)	(3)	(4)	(5)	$Read_{VN}$	$Movements_{VN}$	$Symbols_{VN}$	$InstM_{VN}$	$ECC_{local}(\tau)$
Z_000	3	3	3	15	1	1	5	1	0,533333333
Z_0001	4	4	4	24	1	1	6	1	0,131944444
Z_00011	5	5	4	35	1	0,8	7	1	0,02827381
Z_000111	6	6	4	63	1	0,666666667	10,5	1	0,004993386
$Z_000\underbrace{1...1}_{12}$	15	15	4	255	1	0,266666667	17	1	4,4421E-12
$Z_000\underbrace{1...1}_{13}$	16	16	4	288	1	0,25	18	1	2,89424E-13

(1)-$\mathcal{A}_9, \mathcal{A}_{10}$, τ; (2)-lenght of the word; (3)-$Time$, (4)-$MovementsT$, (5)-$SymbolsT$

6.6 Intelligent measures for Formal Machines

I begin this subsection for assume that the environment where machines and people move is described by the theory of words [Lothaire, 2005],[Lothaire, 2002] and therefore can be described through of the use of an alphabet, a non-empty finite set. I will denote this alphabet by Env. The set of all possibles descriptions of the environment contains the set Env^*. Based on what I said I describe in FMs some concepts that are associated with the idea of intelligence. That descriptions are in the following table:

Table 24: Concepts Associated with the Idea of Intelligence

The concept	Measure
Adaptation (A)	$MIQ_A = \frac{1}{\lvert Env\rvert - \lvert W_{IO}\rvert}$ $W_{IO} = \{\vec{a} = (a_1,...,a_n) \in \times_{i=1}^{n} A(C_i) \vert \exists \vec{c} = (c_1,...,c_n) \in ConfM_i \cup ConfM_f : \lvert c_i\rvert_{a_i} \neq 0$ for all $i = 1,...,n\}$
Creativity (C)	$MIQ_C = \sum_{I \in InstM, \vec{c} \in dom(I)} \sum_{k=1}^{\infty} \frac{\lvert I(\vec{c}) \cap (\times_{i=1}^{n} A(C_i))^k)\rvert}{\lvert Env\rvert^k}$
Deep of Knowledge ($DeepK$)	$MIQ_{DeepK} = \max\{\sum_{\vec{c} \in ConfM} \lvert c_i \rfloor : \vec{c} = (c_1,...,c_n)\}$
Language ($Lang$)	$MIQ_{Lang} = \times_{i=1}^{n} (A(C_i) \cap C_i)$
Level of Knowledge (LK)	$MIQ_{LK} = \sum_{k=1}^{+\infty} \frac{\lvert\{\tau = (\tau_I, \tau_O) \in ConfM : \lvert\tau_I\rvert + \lvert\tau_O\rvert = k\}\rvert}{\lvert Env^k\rvert}$
Learning (L)	$MIQ_L(t) = lim_{\Delta \longrightarrow 0} \frac{\{ECC(t+\Delta) - ECC(t)\}}{\Delta} * LK$
Memory (Mem)	$MIQ_{Mem} = \sum_{i=1}^{n} \lvert C_i\rvert$
Motivation (Mot)	$MIQ_{Mot}(t) = MIQ_A * MIQ_L(t) * PCC$
Perception (Pep)	$MIQ_{Pep} = \sum_{\tau \in L(fM)} \frac{Read(\tau)}{ReadT(\tau)}$
Reasoning (R)	$MIQ_R = ECC * MIQ_C$

The Intelligent measures which I define are the Adaptation (A), Creativity (C), Language ($Lang$), Level of Knowledge (LK), Learning (L), Memory (Mem), Motivation (Mot), Perception (Pep) and Reasoning (R). To measure the intelligence of a machine I define a quantitative measure, the Machine Intelligent Quotient (MIQ), of each one of the concepts associated to intelligence. Thus, for measure the adaptation (A), the creativity (C) and so on ... I use respectively the MIQ_A, MIQ_C and so on.

Now I explain each one of the concepts through of the measures defined:

i) Adaptation (A). This measure measures the adaption of the FM to the environment. This is made through of the capacity that the machine shows to have for rewrite the external environment in words of FM's alphabets.

$$MIQ_A(\text{fM}) = \frac{1}{\lvert Env\rvert - \lvert W_{IO}\rvert} \text{ with } W_{IO} = \{\vec{a} = (a_1,...,a_n) \in \times_{i=1}^{n} A(C_i) \vert \exists \vec{c} = (c_1,...,c_n) \in ConfM_i \cup ConfM_f : \lvert c_i\rvert_{a_i} \neq 0 \text{ for all } i = 1,...,n\}$$

W_{IO} is the FMs alphabet. The Adaptation is measured by the inverse of the difference between the quantity of the letters necessary for describe all the situations in the environment and the quantity of letters that exist in the tasks carried out by the machine

ii) Creativity (C). This measure intends to measure the creativity of the machine. The creativity is measured through of the measuring of the capacity that the machine have to diverge.

$$MIQ_C(\text{fM}) = \sum_{I \in InstM, \vec{c} \in dom(I)} \sum_{n=1}^{\infty} \frac{\lvert I(\vec{c}) \cap Env^n\rvert}{\lvert Env^n\rvert}$$

The creativity is measured through of the capacity that the FM has for diverge. This measure is a sum of proportions. Each proportion is the quantity of the configurations that are possible to obtain for an instruction divided by all words of the same lengths of the environment alphabet.

This measure measures the capacity of the machine to do computation from a set of configurations c_P. Indeed, the machine diverge capacity is high if the sets of configurations $\{c'_p : c_p \vdash c'_p\}$ has very much elements. The capacity to diverge is expressed in the steps of computation $c_P \vdash c'_P$. The creativity is measure from vectors of configurations $\vec{c} = (c_1, ..., c_n)$ wherein all c_i belong to c_P and the instruction $I \in InstM$ which are applied on $\vec{c}$

iii) Language ($Lang$). This measured intends to measure the capacity that the machine presents to communicate on words.

$$MIQ_{Lang}(\text{fM}) = | \times_{i=1}^{n} (A(C_i) \cap C_i)|$$

This measure intends to measure the capacity that the machine has to communicate on words. This communications is made through of the alphabets. Hence, it has in count the alphabets of the set of configurations $ConfM$ and the quantity of letters that they possesses.

iv) Level of Knowledge (LK). This measure intends to measure the level of the knowledge of the machine. This is made through of the a ratio between the quantity of tasks that the machine carries out and the quantity of words that the Environment possesses.

$$MIQ_{LK}(\text{fM}) = \sum_{i=1}^{+\infty} \frac{|\{\tau=(\tau_I,\tau_O)\in\mathcal{L}(fM):|\tau_I|+|\tau_O|=n\}|}{|Env^n|}$$

This is a sum of ratios. Each one of the ratios is a ratio between the length of the words, that the machine carries out, and the quantity of the words, of the same length, possibles in the environment

v) Learning (L). This measure measures the capacity of the machine to learn. This is done by calculating the instantaneous Effective Computational Capacity, in certain iteration or moment, multiplied by the Level of the Knowledge of the machine.

$$MIQ_L(\text{fM})(t) = lim_{\Delta\longrightarrow 0} \frac{ECC(t+\Delta)-ECC(t)}{\Delta} * LK$$

Looking to the formula is possible to see that to be able to learn is necessary that the machine is involved in the process of acquisition of knowledge. In this process the values of the FBU measures are changed over the time or along the iterations. The learning is a process that happen over the time or along the iterations.

vi) Memory (Mem). This measure measures the capacity that the machine has to store. This is done by the sum of the cardinality of the machine components.

$$MIQ_M = \sum_{i=1}^{n} |C_i|$$

vii) Motivation (Mot). This measures intends to measure the motivation which the machine possesses. The Adaptation of the machine to the environment, their capacity to learn, and their potential computational capacity, are considered important measures related with the motivation and, because of that, they are part of the measure.

$$MIQ_{Mot}(t) = MIQ_A * MIQ_L(t) * PCC$$

viii) Perception (Pep). This measure measures the capacity which the machine presents to precept the outer environment. This is done through of a ratio between the distinct reads and the total reads that the machines carries out to run a certain task.

$$MIQ_{Pep} = \sum_{\tau \in L(\text{fM})} \frac{Read(\tau)}{ReadT(\tau)}$$

Is supposed that all the phenomenons, each one of its occurrence, happen always with something different. So don't exist phenomenons exactly equals. Thus, several attributions of the same symbol to several perceptions of the environment must have a penalisation.

ix) Reasoning (R). This measures intends to measure the capacity which the machine has to reasoning. This capacity is related with the effective computational capacity of the machine and with its creativity. Thus are part of the measure the ECC and the MIQ_C.

$$MIQ_R = ECC * MIQ_C$$

7 Validating some Formal Machines Measures using Machine Learning

In this section I describe a study that I did. The study had as aim to validate some of the measures that was referred in sub section 6.6. The work consisted in to study Machine Lerning (ML) and FMs measures and to relate heuristically that measures. This was done through of the analysis that was done in a set of data about the health condition of a person[52]. These data are collected in four countries Spain, Italy, Japan and Chine[53]. The data consisted in a set of Excel files[54]. In each Excel file the data are divided in 5 columns (one for each sensor) and for each one the data are one hundred until two hundreds values obtained of five sensors. For each person that was analyzed, the five sensors are used to obtain data about its health condition. The decision about the health condition, good or not, was evaluated by the monitor that performed the measurements. In following can be seen the steps that was given to perform the work. This work was accomplished in six phases:
1- Analyzing graphically the data. First I built a java program with a library imported to work with Excel files and in each one of the files I draw the graphical of the sensors. From the graphical built I saw that the graphics have all the same structure and based on that I found several features (See table 24)
2- Building a dataset. I used the features that I found in the data and I built a dataset. The dataset is a set of rows, 4230 rows, each row is composed by values of the features referred[55]. After this I prepared the dataset for working with MLs[56]
3- Generating and training 1000 Neural Networks. After to have the dataset I used a Software to generate and to train Neural Networks with three layers[57]. For to do this was necessary to divided the dataset in three new excel files; one for the train, one to validate the train and more one to test the train of the networks[58]. As the result of the training of one thousand of that machines the software give us a file about each one of the machines. In this file is described the structure of each one of the machines and for each one appear the respective ML measurements[59].

[52] https://drive.google.com/folderview?id=0B-VWbS8s90IlZElRdXU1TkNjaUk&usp=sharing

[53] The data were sent to me by profesor Omatu and already originated a published work "Multi-agent systems for classification of e-nose data"[Iketal15]

[54] https://drive.google.com/folderview?id=0B-VWbS8s90IlZElRdXU1TkNjaUk&usp=sharing

[55] https://docs.google.com/spreadsheets/d/1RGDB5w0DFMNW8iVPXW9QKVxouBbJswsH_05xZZnezZ4/edit?usp=sharing

[56] https://docs.google.com/spreadsheets/d/1nK5Y57GUdF0pT_yFMm96hY1MJT6H3CFPkb8ImPUwiDk/edit?usp=sharing

[57] mbp.sourceforge.net/

[58] https://drive.google.com/folderview?id=0B-VWbS8s90IlfndqTUN2cHJPRndEaXdJWHJZb3NEbG9PNlFjYklPR1puNjlaZER4c0V5V0E&usp=sharing

[59]

4- Developing a drive to the Back-Propagation Neural Networks Machines (BPNNs). I developed an algorithm to transform the generated ML machines in FMs and I calculated for each one of the FMs several FMs measurements[60].
5- Relating MLs and FMs measurements. I compared the usual measures of the ML with the measures built in FMs and I established relations between the two types of measures.
6- A mathematical conjecture as result. As the conclusions obtained were heuristic conclusions I made a mathematical conjecture based in statistical observations. The population are all BPNNs or the FMs that are obtained from BPNNs without they losing their structure. The mathematical result is conjectured for all BPNNs.

7.1 An information system

The original data were data without treatment. These data were collected in four countries, in Italy in Spain, Japan and in Chine and were about the heath condition of a person. There were two possible results, the person was in a good health state or not. The data were collected from measures of five sensors, and the analysis about the health state, of the person submitted to the sensors, is doing by the person who collected the data. The sensors used to measure the health state of a person were following.

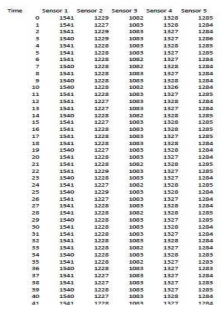
Table 25: Table of sensors used

Sensor number	Model number	Application
1	SB· AQ5	Volatile Organic Coumpound)
2	SB· 15 · 00	Flammable Gas(Propane, Butatne)
3	SB· 30 · 04	Alchol Detection
4	SB· 42A · 00	Refrigerant Gas (Freon)
5	SB· 31 · 02	Solvent Detection

The data collected were around 1000 thousand Excel files each file with five columns of data one for each sensor. Each column referred had a quantity of data rows between one hundred until two hundred.

After the collection of the data I drew the graphical of the sensor values in each one of the Excel files. I saw that all graphics are similar. The graphics are similar to the following graphic.

[60] https://drive.google.com/file/d/0B-VWbS8s9OIlUjY4VXRyNFVMaFk/view?usp=sharing



Time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
0	1541	1229	1082	1328	1285
1	1541	1227	1083	1328	1284
2	1541	1229	1083	1327	1284
3	1540	1229	1083	1327	1286
4	1541	1228	1083	1328	1285
5	1541	1228	1083	1327	1285
6	1541	1228	1082	1327	1284
7	1540	1228	1082	1328	1284
8	1541	1228	1083	1327	1284
9	1540	1228	1083	1328	1284
10	1540	1228	1082	1326	1284
11	1541	1228	1083	1327	1285
12	1541	1227	1083	1328	1284
13	1541	1227	1083	1327	1284
14	1540	1228	1082	1328	1285
15	1541	1227	1083	1328	1285
16	1541	1228	1083	1328	1285
17	1541	1228	1083	1327	1285
18	1541	1228	1083	1328	1284
19	1540	1227	1083	1328	1284
20	1541	1228	1083	1327	1284
21	1541	1228	1082	1328	1285
22	1541	1229	1083	1327	1285
23	1540	1228	1083	1327	1284
24	1541	1227	1082	1328	1285
25	1540	1229	1083	1328	1284
26	1541	1227	1083	1327	1284
27	1541	1228	1083	1328	1284
28	1541	1228	1082	1328	1285
29	1540	1228	1083	1327	1285
30	1541	1228	1083	1328	1284
31	1541	1228	1083	1327	1284
32	1541	1228	1083	1328	1284
33	1541	1228	1082	1327	1284
34	1540	1228	1083	1328	1283
35	1541	1228	1082	1327	1283
36	1540	1228	1083	1327	1283
37	1541	1227	1083	1327	1284
38	1541	1227	1083	1327	1283
39	1540	1228	1083	1327	1285
40	1540	1227	1083	1328	1284
41	1541	1228	1083	1327	1284

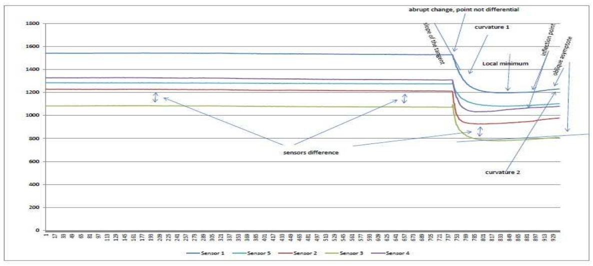
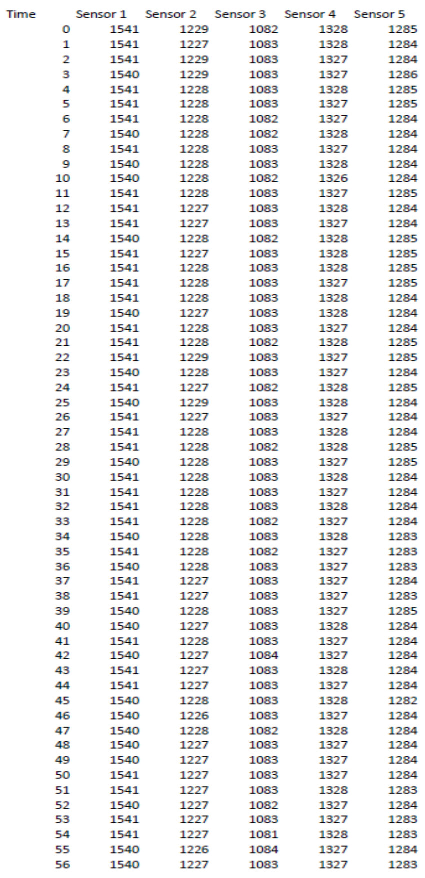


Figure 56: (left) Excel file about the data collection of the five sensors. (Right) features of the data)



Time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
0	1541	1229	1082	1328	1285
1	1541	1227	1083	1328	1284
2	1541	1229	1083	1327	1284
3	1540	1229	1083	1327	1286
4	1541	1228	1083	1328	1285
5	1541	1228	1083	1327	1285
6	1541	1228	1082	1327	1284
7	1540	1228	1082	1328	1284
8	1541	1228	1083	1327	1284
9	1540	1228	1083	1328	1284
10	1540	1228	1082	1326	1284
11	1541	1228	1083	1327	1285
12	1541	1227	1083	1328	1284
13	1541	1227	1083	1327	1284
14	1540	1228	1082	1328	1285
15	1541	1227	1083	1328	1285
16	1541	1228	1083	1328	1285
17	1541	1228	1083	1327	1285
18	1541	1228	1083	1328	1284
19	1540	1227	1083	1328	1284
20	1541	1228	1083	1327	1284
21	1541	1228	1082	1328	1285
22	1541	1229	1083	1327	1285
23	1540	1228	1083	1327	1284
24	1541	1227	1082	1328	1285
25	1540	1229	1083	1328	1284
26	1541	1227	1083	1327	1284
27	1541	1228	1083	1328	1284
28	1541	1228	1082	1328	1285
29	1540	1228	1083	1327	1285
30	1541	1228	1083	1328	1284
31	1541	1228	1083	1327	1284
32	1541	1228	1083	1328	1284
33	1541	1228	1082	1327	1284
34	1540	1228	1083	1328	1283
35	1541	1228	1082	1327	1283
36	1540	1228	1083	1327	1283
37	1541	1227	1083	1327	1284
38	1541	1227	1083	1327	1283
39	1540	1228	1083	1327	1285
40	1540	1227	1083	1328	1284
41	1541	1228	1083	1327	1284
42	1540	1227	1084	1327	1284
43	1541	1227	1083	1328	1284
44	1541	1227	1083	1327	1284
45	1540	1228	1083	1328	1282
46	1540	1226	1083	1327	1284
47	1540	1228	1082	1328	1284
48	1540	1227	1083	1327	1284
49	1540	1227	1083	1327	1284
50	1541	1227	1083	1327	1284
51	1541	1227	1083	1328	1283
52	1540	1227	1082	1327	1284
53	1541	1227	1083	1327	1283
54	1541	1227	1081	1328	1283
55	1540	1226	1084	1327	1284
56	1540	1227	1083	1327	1283

Figure 57: Excel file about the data collection of the five sensors

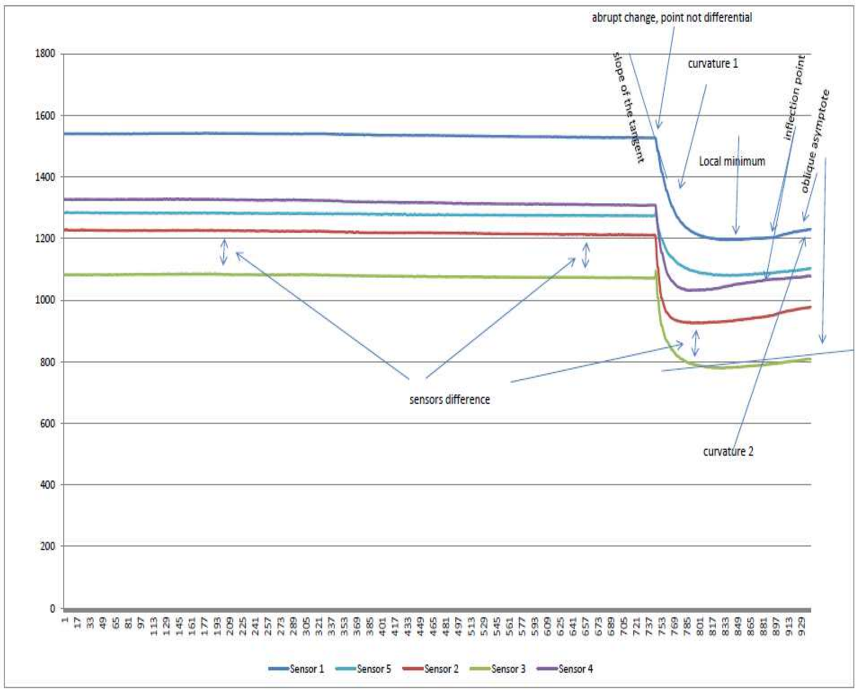


Figure 58: features of the data

From the study of this graphics I designed the features and I discovered a lot of them. The features found are about twelve different types[61] and with them I built a dataset[62].

I wrote the dataset in only one Excel sheet, that is a set of raws of the features and the Result is the health condition of the person submitted to the measurements. Each previous file generate 5 different rows in the dataset one for each sensor and were found new algebraic features as for example the difference between the values of the sensors. These new features were increased for around 30. Thus, the dataset is a file with around 4235 rows and 30 columns. Each column was a feature or the health condition of the person.

Following I used a software to train Neural Networks Machines[63] and I generated 1000 machines and for each one I calculated 4 measures [64]; the F-measure, the Accuracy, the RMS and the time of the train.

[61] https://drive.google.com/file/d/0B-VWbS8s9OIlcC1seHMyQ3RVUlU/view?usp=sharing
[62] https://drive.google.com/drive/folders/0B-VWbS8s9OIlZGlyWjZfc1A5ZkU
[63] http://sourceforge.net/projects/mbp/
[64] https://drive.google.com/file/d/0B-VWbS8s9OIlVGpGZU42MnUwZ0k/view?usp=sharing

Table 27: Measures of the MLs

$precision = \frac{tp}{tp+fp}$	$recall = \frac{tp}{tp+fn}$
$Accuracy = \frac{tp+tn}{tp+tn+fp+fn}$	$F - measure = \frac{precision\dot{r}ecall1}{precision+recall}$
$RMS = \sqrt{\sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)}$	

Table 26: Evaluation of All possible results of a ML

Total population	Condition positive	Condition negative
Test outcome positive	*tp*= true positive	*fp*= false positive
Test outcome negative	*fn*= false negative	*tn*= true negative

For doing this from the dataset I built three new Excel data files extracted from the dataset. These new files were sets of rows of the dataset file. Thus I obtained a file to train the machines[65], a file to validate the train[66] and a file to test the decisions of the machine[67]. After this step I developed an algorithm to transform Neural Networks in Formal Machines and then I calculated several measures of the FMs[68].

7.2 Obtaining a FMs from a Back-Propagation of Neural Network Machine

fM = $(CompM_B, CompM_R, ConfM, ConfM_i, ConfM_f, InstM, Alg_{VN})$.

Let a ML be a Back-Propagation Neural Network (BPNN) that is a network with three layers one of them is a hidden layer, a set of inputs and an output layer. The network is composed in each layer by neurons, that neurons make processing. The neurons within of the same layer have not connection among them. Each neuron of a layer is connected to all neurons of the next layer. Each connections is an edges between neurons. The inputs introduced in the networks are in same quantity of the neurons in the first layer. Each input is introduced in only one neuron of the first layer. These networks, these machines should be trained to answer to certain question, to take some decisions.

[65] https://docs.google.com/spreadsheets/d/1kM1C6MAT_Yi-cxWpp4PVxPcn3RxuAX5by2pE3tU18Xk/edit?usp=sharing

[66] https://docs.google.com/spreadsheets/d/1uxRle6RpAN5PE01pGsWBsejpkxVdNKrdfB8X6fs2X3o/edit?usp=sharing

[67] https://docs.google.com/spreadsheets/d/1YWkepSHLp-KtHZfpxhTkaFcQwxQG-B49H9mKmS8wB-8/edit?usp=sharing

[68] https://docs.google.com/spreadsheets/d/1jiRlInuGYLwyecjX47gXZrKQkPUzAgLSeDPLlTiI1io/edit?usp=sharing

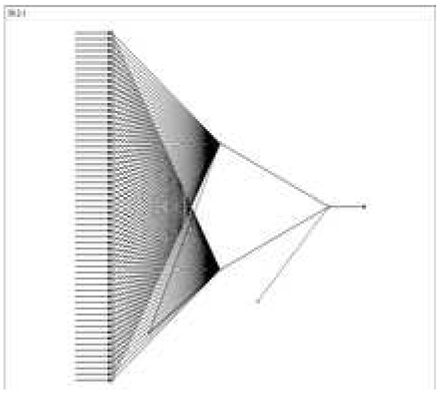


Figure 59: A Back-Propagation Neural Network Machine Architecture, 59-3-1

Now, I am going to describe the algorithm that allows to transform the BPNN to a FM. The implementation of this algorithm is what I call a drive of the BPNN_FMs.

$CompM_B = \{C_1, C_2, C_3, C_4\}$ wherein
$C_1 = \bigtimes_{i=1}^{n_2}[0,1]$, $C_2 = \bigtimes_{i=1}^{n_1}(\bigtimes_{j=1}^{n_2}[0,1])$, $C_3 = \bigtimes_{i=1}^{n_2}[0,1]$ and $C_4 = \{0,1\}$,
$CompM_R = \emptyset$,
$ConfM = C_1 \times C_2 \times C_3 \times C_4$
$ConfM_i = ConfM_f = ConfM$
$InstM = \{I_1, I_2, I_{mv}\}$ wherein $I_1(x_1^1, x_2^1, ..., x_{n_1}^1) = (x_1^2, x_2^2, ..., x_{n_2}^2)$, and
$I_2(x_1^2, x_2^2, ..., x_{n_2}^2) = z$.

$(x_1^2, x_2^2, ..., x_{n_2}^2) = I_1(x_1^1, x_2^1, ..., x_{n_1}^1) = f_1(M(n_2, n_1)(x_1^1, x_2^1, ..., x_{n_1}^1)^T)$ wherein

$$M(n_2, n_1) = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 & \cdots & w_{1n_1}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 & \cdots & w_{2n_1}^1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n_21}^1 & w_{n_22}^1 & w_{n_23}^1 & \cdots & w_{n_2n_1}^1 \end{bmatrix}$$

$z = I_2(x_1^2, x_2^2, ..., x_{n_2}^2) = f_2(M(1, n_2)(x_1^2, x_2^2, ..., x_{n_1}^2)^T)$ wherein

$$M(1, n_2) = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 & \cdots & w_{1n_2}^2 \end{bmatrix}$$

Von Neumann Algorithm, VN_{Alg}
1- Get an input from the Environment, c_{input}
2- Translate c_{input} for a n_2-tuple of the machine alphabet, u
3- Take the initial configuration $c = (\bigtimes_{i=1}^{n_1} u_i, \bigtimes_{i=1}^{n_1}(\bigtimes_{j=1}^{n_2} w_{ij}^1), \bigtimes_{j=1}^{n_2} w_{1j}^2, 0)$
4- Take $u = (u_1, ..., u_n)$ and applies the instruction I_{mv}, $I_{mv}(u) = u'$ wherein $u' = (u_1', ..., u_n')$ such that if the i-th row is not a missing value row then $u_i' = u_i$ otherwise $u_i' = u_i + \delta r_i$.

5- Obtain c_1 from c, $c_1 = (\bigtimes_{i=1}^{n_1} u'_i, \bigtimes_{i=1}^{n_1}(\bigtimes_{j=1}^{n_2} w_{ij}^1), \bigtimes_{j=1}^{n_2} w_{1j}^2, 0)$
6- Obtain c_2 from c_1, $c_2 = (\bigtimes_{i=1}^{n_1} u'_i, \bigtimes_{i=1}^{n_1}(\bigtimes_{j=1}^{n_2} w_{ij}^1), I_1(u'), I_2(I_1(u')))$
7- $z \longleftarrow I_2(I_1(u'))$;
8- Translate z to c_{output} in the Environment Language.

7.3 BPNNs and FMs measures

In following I created a new dataset to analyze and compare the measures used on BPNNs and the measures used and created for FMs. This new dataset[69] is a sheet of an Excel file with around 1000 rows and 37 columns. The columns is composed by measures of the BPNN and by measures of FMs. This new dataset is structure of the following way:
i) dataset row number
ii) data about the Neural Network Machines, BPNN
ii.1) The architecture of the machine
ii.1.1) Network
ii.2) measures of the machine
ii.2.1) Epoch, ii.2.2) Time (s), ii.2.3) Train RMS (%), ii.2.3) F-Measure (%), ii.2.4) Accuracy (%), ii.2.5) Saved (filename),
iii) data about FMs
iii.1) The architecture of the FM
iii.1.1) C_1, iii.1.2) C_2, iii.1.3) C_3, iii.1.4) Inst, iii.1.5) COA
iii.2) measures unities of FMs
iii.2.1) Space, iii.2.2) Time,iii.2.3) Symbols, iii.2.4) Movements, iii.2.5) Read, iii.2.6) Inst,iii.2.7) Write,
iii.3) derived measures of FMs
iii.3.1) ECC_{global}, iii.3.2) MIQ_{DK},iii.2.2) MIQ_{LK}, iii.3.4) $MIQ_{L_{fact}}$, iii.3.5) MIQ_L
iv) Normalization of the BPNNs and FMs measures and the construction of the respective graphics
iv.1) N_Train RMS (%), iv.2) N_F-Measure (%), iv.3) N_Accuracy (%),
iv.4) N_Space,iv.5) N_Movements,iv.6) N_ECC, iv.7) MIQ_{DK}, iv.8) MIQ_L

Analyzing the graphics, referred above in iv), below and reading the data is possible to observe the behavior of certain measures compare them and conclude somethings.

[69]https://docs.google.com/spreadsheets/d/1WCxVPAhHA0-jLKg_AMysj9vkpvkukYwUMutv2k_A_Ck/edit#gid=1794133931

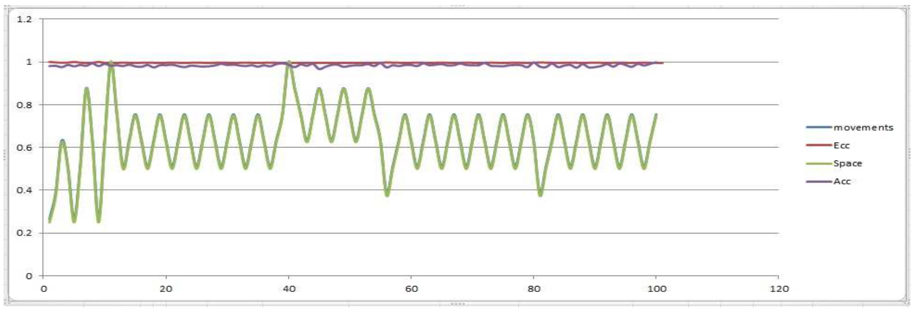


Figure 60: Normalized FMs measures, Space, Movements, ECC. Normalized ML measure: A_cc

Observing the figure 60 in BPNNs the $ECC \approx A_{cc}$ and $Movements \approx Space$ assume approximately the same values, heuristically can be said that they are equal. This make sense because in slight language the ECC_{global} is a measure that intended to measure the computational capacity, of the FM, to process simple tasks. Considering that the population are the tasks which can occur in the environment. That tasks should be divided in tasks that a BPNN is able to perform, condition positive, and tasks that they are not, condition negative. (See Table 26) The A_{cc} of a BPNN is a measure that corresponding to a proportion between the tasks which one specifies BPNN performed tp under the condition positive and the tasks that the machine no performed fp under the condition negative. Thus, in slight language the ECC_{global} and A_{cc}, in BPNNs, intended to measure the same computational capacity. When both measures are normalized the values obtained by ECC_{global} and A_{cc} are very close. The measures $Movements$ and $Space$ are both measures of FMs and their proximity is a consequence of the properties of the BPNNs machines. The $Movements$ intended to measure the quantity of movements that is made by the machine and $Space$ intends to measure the space occupied in the machine during the execution of a task. In the BPNN machines always that is done a movement, within the machine, is generated a new value that is stored in a place.

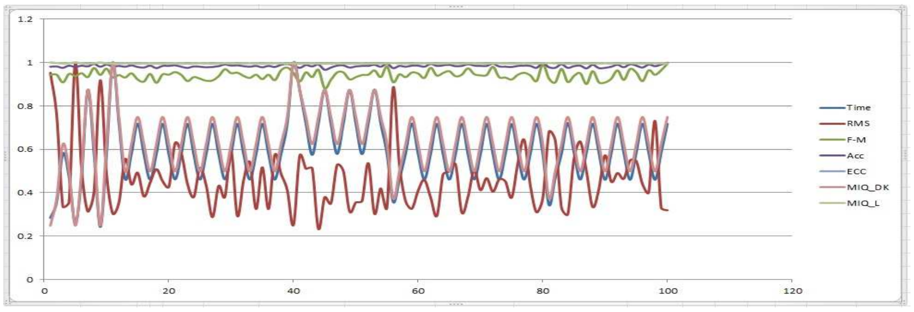


Figure 61: Back-Propagation Neural Network Machine Architecture

Analyzing the graphical in the figure 61, in BPNNs, the $A_{cc} \approx MIQ_L$, $Time \approx MIQ_{DK}$ and $reflexion(RMs) \approx MIQ_{DK}$. The A_{cc}, the ECC_{global} and MIQ_L with the values normalized are measures have values very close. Thus, you can measure the learning capacity of a BPNN, the MIQ_L, by its A_{cc} and the deep knowledge of the machine through of a reflection of the RMS made under a specific line reflection previews built.

From the analyze of the dataset[70] is possible to stablish the relations that are presented in the figure 61. The read of the tables should be done in following way. In the table above can be seen, in the columns Row, *Movements* and *Space* respectively the value of their quartile 1, quartile 3 and quartile 3. The read should be done in the following way in the dataset, in a row where you can see the value of the RMS corresponding to the quartile 1, in the same row you can find for *Movements* a value around their quartile 3 and for the measure *Space* also a value around their quartile 3. Now at the shadow of this way of interpreting the table I found a lot of relations among the Measures.

Table 28: Relations among RMS, *Movements* and *Spaces*

RMS	*Movements*	*Space*
Minimum	Maximum	Maximum
quartile 1	quartile 3	quartile 3
quartile 2	quartile 2	quartile 2
quartile 3	quartile 1	quartile 1
Maximum	Minimum	Minimum

The values, of the measures RMS of the BPNNs and *Movements* and *Space* of the FMs, that give position in the BPNNs as a population have inverted position values. If you classify the machines, the BPNNs, through of their RMS values, by the quartile:
i) you have that the machines that have RMS values between the minimum and the quartile 1 are approximately the same machines that would be between the quartile 3 and the maximum if you now classify FMs machines, obtained from the BPNN_FM drive, by the measures *Movements* and *Space*.

ii) you have that the machines that have RMS values between the quartile 1 and the quartile 2 are approximately the same machines that would be between the quartile 2 and the quartile 3 if you now classify FMs machines, obtained from the BPNN_FM drive, by the measures *Movements* and *Space*.

iii) you have that the machines that have RMS values between the quartile 2 and the quartile 3 are approximately the same machines that would be between the quartile 1 and the quartile 2 if you now classify FMs machines, obtained from the BPNN_FM drive, by the measures *Movements* and *Space*.

[70] https://docs.google.com/spreadsheets/d/1jiRlInuGYLwyecjX47gXZrKQkPUzAgLSeDPLlTiIlio/edit?usp=sharing

iv) you have that the machines that have RMS values between the quartile 3 and the Maximum are approximately the same machines that would be between the Minimum and the quartile 1 if you now classify FMs machines, obtained from the BPNN_FM drive, by the measures $Movements$ and $Space$.

Table 29: Relations among Time, MIQ_{DK}

Time	MIQ_{DK}
Minimum	Minimum
quartile 1	quartile 1
quartile 2	quartile 2
quartile 3	quartile 3
Maximun	Maximun

Following the same way of read the data you have that the values of the BPNN measure time(s) (the time that is necessary to train the BPNN) and the values of the FM measure MIQ_{DK} , which give position in the BPNNs as a population, have the same position values. If you classify the machines, the BPNNs, through of their time(s) value by the quartile:
i) you have that the BPNN machines that have values between the minimum and the quartile 1 are approximately the same FMs machines, obtained from the drive BPNN_FM, that have values between the minimum and the quartile 1 measured from MIQ_{DK},
ii) you have that the BPNN machines that have values between the quartile 1 and the quartile 2 are approximately the same FMs machines, obtained from the drive BPNN_FM, that have values between the quartile 1 and the quartile 2 measured from MIQ_{DK},
iii) you have that the BPNN machines that have values between the quartile 2 and the quartile 3 are approximately the same FMs machines, obtained from the drive BPNN_FM, that have values between the quartile 2 and the quartile 3 measured from MIQ_{DK},
iv) you have that the BPNN machines that have values between the quartile 3 and the Maximum are approximately the same FMs machines, obtained from the drive BPNN_FM, that have values between the quartile 3 and the Maximum measured from MIQ_{DK},

The results that I have are about samples, samples of 1000 MLs more specifically 1000 BPNNs. Now based in these samples and in others indications presented by the data and joining my experience as mathematician I conjecture with a considerable grade of belief the following results.

Conjecture 7.1 *In Back Propagation Neural Networks and the FMs obtained from them there are the following results about measurements:*
i) The row of the dataset that contains in the column of the Time, measures on

BPNNs, their Minimun (respectively, quartile 1, quartile 2, quartile 3, Maximun) contains also asymptotically in the column of the MIQ_{DK} their Minimun (respectively, quartile 1, quartile 2, quartile 3, Maximun).
ii) The row of the dataset that contains in the column of the RMS their Minimun (respectively, quartile 1, quartile 2, quartile 3, Maximun) contains also asymptotically in the column of the $Movements$ their Maximun (respectively, quartile 3, quartile 2, quartile 2, Minimun).
iii) $Movements_i \sim_a Space_i$, *with* $i \in \{Minimun, quartile\ 1, quartile\ 2, quartile\ 3, Maximun\}$
iv) $Accuracy \sim_a ECC_{global}$

To transform this conjecture in a mathematical theorem is part of a future work in FMs technology theory. I am thinking to do the proof in a soon future.

8 Analyzing a Microcontroller

Next I am going to sketch an instantiation of a ATMEL MCU Hardware in the formalism that was presented here, the FMs. The MCU is a chip. The $CompM_B$ are the components of the chip; transistors, resistors, capacitors, and so on ... All the discrete electrical elements that constitute the chip. The communication channels, which allows to connect the different elements of the chip and from which runs the electric flux are $CompM_R$ elements. The machine configurations are their different electrical states and they are the components of the machine, which are reflected in the state of the registers. Thus, the configurations are regarded as the state of the registers. The machine instructions are instructions which are contained in the datasheet of the chip. Each instruction is measured in cycles of machine. Each machine cycle is a Von Neumann Cycle. Thus is sketched an instantiation of an ATMEL MCU in the formalism presented. A configuration in which there is at least one port configured as input (respectively output) is an initial configuration (respectively final). All configurations are materialized in the registers of the MCU. The configurations of the machine can be seen as their registers.

9 Conclusion and Future Work

One of the results of this work was in the construction of a new formalism, called Formal Machine. This formalism is a computational model, and was proved that a lot of computational models can be rewritten as FMs. The new formalism also allowed, using Category theory, to defined what it means to preserve the structure of a computational model. The notion of preserving a structure of an FCS was defined according to the functor concept.

The schema of the Database of an FM was built, and an API was developed to allow translating a Finite Automata for an FM and store it. The algorithm that allows to transform a FCS to a is called a drive. The same work is ongoing for others computational systems such as Pushdown Automata, Turing Machines, URM and so on.

With the aim to implement FMs was designed their computational structure. This structure is what allows to implement different FMs to solve problems. The computation algorithm for the computational structure of the FM was designed in serial and parallel mode. The serial mode already is implemented. The implementation of the parallel mode is ongoing[71].

Two games were built, the Tic Tac Toe game and the Four In Line game, to test how an FM can be implemented to solve a problem, to play a game, and so on ... I plan to implement a game of checkers with an FM acting as one of the players and playing against human players. Furthermore, I expected to implement solutions in FM Technology for a large number of engineering problems.

In this work also was defined, as measures in FMs, many concepts and skills that are associated with intelligent procedures. Thus, arose the Machine Intelligence Quotient (MIQ) as a way of measuring intelligence in FCSs through of the FMs. In fact the MIQ are several measures one for each concept associated with intelligence. These measures can be seen as measures in all computational models. In the generating universes Software, "Generator of Universes and Simulator of Formal Machines", you can simulate the behavior of certain FCSs. The FM yet is no implemented in the software. But, once that the computational models are embedded or rewritten in FMs without losing their structure when I implement the FMs I can simulating the behavior of any computational models. The implementation of the MIQ in the software is another thing that is ongoing.

In this work also was showed that the FMs have more computational powerful than Turing Machines. From this idea has made possible to define the notion of supermachines. The supermachines are machines more power than Turing Machines.

[71]Can happen that while you read this thesis the implementation is ended

Conclusión y trabajos futuros

Uno de los resultados de este trabajo fue la construcción de un nuevo formalismo, llamado Máquina formal, en inglés Formal Machine (FM). Este formalismo es un modelo computacional, y se demostró que una gran cantidad de modelos computacionales se pueden reescribir como FMs. El nuevo formalismo también permitió que, utilizando la teoría de categorías, si ha definido lo que significa preservar la estructura de un sistema computacional. La noción de preservar la estructura de un Sistema Computacional formal, en inglés Formal Computational System (FCS), se define de acuerdo con el concepto functor. En el trabajo fue construida la estructura de la base de datos de una FM, y fue desarrollada una algoritmo para permitir la traducción de un autómata finito para una FM y hacer almacenarla en la base de datos. La implementación de este tipo de algoritmos que permiten transformar las FCSs en FMs se llaman drive, así se ha construido una drive para Autómatas finitos. Lo mismo se está trabajando para otros sistemas computacionales como Autómata de Pila, Máquinas de Turing, URM y así sucesivamente. Con el objetivo de implementar FMs fue diseñada su estructura computacional. Esta estructura es la que permite poner en práctica las FMs para resolver problemas. El algoritmo para la estructura computacional del FM fue diseñado en el modo serie y paralelo. El modo serie ya está implementado. La implementación del modo paralelo está en curso[72]. Dos juegos fueron construidos, el juego de Tic Tac Toe y el juego Cuatro en Línea, para probar cómo en una FM se pueden implementar para resolver un problema; cómo jugar un juego, etc.. Yo tengo la intención de poner en práctica un juego de damas con una FM siendo uno de los jugadores y ponerla a jugar contra jugadores humanos. Además, en el futuro pienso implementar soluciones en Tecnología de FMs para un gran número de problemas de ingeniería. En este trabajo se define también, con mediciones en las FMs, conceptos y habilidades que están asociados con proceder de forma inteligentes. Esas medidas son obtenidas por lo que llamo Cociente de Inteligencia de las Máquinas, en inglés Machine Intelligent Quotient (MIQ). El MIQ es la forma de medir la inteligencia en los sistemas computacionales y es definida en FMs. Se tiene un tipo de MIQ para cada uno de los conceptos que están asociados con la idea de comportamiento inteligente. Al medir estés conceptos en las FMs vamos a tener también una medición para modelos computacionales en general. Yo he desarrollado un software, en inglés "Generator of Universes and Simulator of Formal Machines", en él se puede simular el comportamiento de ciertas FCSs. Las FMs son aunque por implementar en el software. Pero, una vez que los modelos computacionales están incrustados o reescritos en FMs sin perder su estructura cuando yo implementar la FM va a ser posible la simulación de comportamiento cualquier modelos computacionales. La aplicación de los MIQs en el software está otra de las cosas que están en curso. Ha sido demostrado que las FMs son computacionalmente más poderosas que las máquinas de Turing. Basado en esta idea ha sido posible

[72]Puede que sucederá que mientras usted lee esta tesis la implementación ya ha terminado

definir la noción de supermachines. Las supermachines son máquinas con más potencia computacional que las máquinas de Turing.

References

[ArB09] Arora, Sanjeev and Barak, Boaz, 2009 *Computational Complexity: A modern approach*, Cambridge University Press

[Atm13] Atmel, last updated: 02/2013 *ATmega48A/PA/88A/ PA/168A/PA/328 /P Complete*, Datasheet http://www.atmel.com/devices/atmega328.aspx?tab=documents

[Atm14] *ATmega48A/PA/88A/ PA/168A/PA/328 /P Summary*, Datasheet `http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller -ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_ Summary.pdf`

[BDR10] Berstel, Jean; Dominique, Perrin; and Reutenauer, Christophe, 2010, *Codes and Automata (Encyclopedia of Mathematics and its Applications)*, Cambridge University Press.

[BoM82] Bondy, J.A. and Murty, U.S.R., 1982, *Graph Theory with Applications*, North-Holand.

[BSS99] B. Schölkopf, C. J. C. Burges, and A. J. Smola, 1999 *Advances in Kernel Methods: Support Vector Learning* MIT Press, Cambridge, MA

[CaWi07] Mathematical Logic Chiswell, Ian and Wilfrid, Hodges, 2007 *Mathematical Logic*, Oxford University Press, Inc. New York, NY, USA ©2007, page 213-215 ISBN 978–0–19–857100–1 ISBN 978–0–19–921562–1 (Pbk)

[ChAl36a] Church, Alonzo 1936 *A Note on the Entscheidungsproblem* J. Symb. Log. 1(1): 40-41

[ChAl36b] Church, Alonzo, 1936 *Correction to A Note on the Entscheidungsproblem* J. Symb. Log. 1(3): 101-102

[ChAl85] Church, Alonzo, 1985 *The Calculi of Lambda Conversion.* (AM-6) (Annals of Mathematics Studies) Princeton University Press Princeton, NJ, USA ©1985 ISBN:0691083940

[ChNo56] Chomsky, Noam (1956). *Three models for the description of language* IRE Transactions on Information Theory (2): 113–124. doi:10.1109/TIT.1956.1056813.

[ChNo59] Chomsky, Noam, 1959 *On certain formal properties of grammars* Information and Control 2 (2): 137–167. doi:10.1016/S0019-9958(59)90362-6.

[CoE11] Courcelle, Bruno and Engelfriet, Joost, 2011, *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach*, Cambridge University Press.

(pag 46) Theorem 1.16 A set of terms over a Finite signature is MS-definable if and only if it is recognizable, i.e., accepted by a Finite automaton.

[Cut97] Cutland, N. J., 1997, *Computability, An introduction to recursive function theory*, Cambridge: Cambridge University Press.

[DaE12] Darmois, E. and Elloumi, O., 2012, *Introduction to M2M, in M2M Communications: A Systems Approach (eds D. Boswarthick, O. Elloumi and O. Hersent)*, John Wiley and Sons, Ltd, Chichester, UK. doi: 10.1002/9781119974031.ch1

[DHS01] R.O. Duda, P.E. Hart, D.G. Stork, 2001 *Pattern Classification* Wiley, ISBN 0-471-05669-3

[Eil74] Eilenberg, Samuel, 1974, *Automata, languages, and machines, Volume A*, Academic Press.

[MaDa06] Davis, Martin, 2006 textitThe Incompleteness Theorem, in Notices of the AMS vol. 53 no. 4 (April 2006), p. 414.

[HeDO49] Hebb, D. O., 1949 *The Organization of Behavior: A Neuropsychological Theory* New York: Wiley and Sons. ISBN 9780471367277.

[HiD02] Hilbert, David, 1902 *Mathematical Problems*, Bulletin of the American Mathematical Society, vol. 8, no. 10 (1902), pp. 437-479. Earlier publications (in the original German) appeared in Göttinger Nachrichten, 1900, pp. 253-297, and Archiv der Mathematik und Physik, 3dser., vol. 1 (1901), pp. 44-63, 213-237.

[Hop08] Hopcraft, John E., 2008, *Introduction to Automata Theory, Languages, And Computation, 3/E*, Pearson Education.

[Iketal15] Ikeda Yoshinori, Omatu Sigeru, Chamoso Pablo,Pérez Alberto and Javier Bajo, 2015 *Multi-agent Systems for Classification of E-Nose Data*, Ambient Intelligence - Software and Applications Advances in Intelligent Systems and Computing Volume 376, 2015, pp 183-192

[KPPK] *Artificial Neural Networks: An Introduction*, Por Kevin L. Priddy,Paul E. Keller pag 11 International Society for Optinal Engineering 2005

[Laf03] Lafore, Robert, 1997, *Data Strutures and Algorithms in Java* Sams Publishing

[Lot97] Lothaire, M., 1997, *Combinatorics on Words* Cambridge University Press

[Lothaire, 2005] Lothaire, M., 2005, *Applied Combinatorics on Words (Encyclopedia of Mathematics and its Applications)*, Cambridge University Press

[Lothaire, 2002] Lothaire, M., 2002, *Algebraic Combinatorics on Words (Encyclopedia of Mathematics and its Applications)*, Cambridge University Press; 1 edition (May 20, 2002)

[Ma98] Mac Lane, Saunders, 1998 *Categories for the Working Mathematician* Springer (Graduate Texts in Mathematics) ISBN 0-387-98403-8

[Mac10] Mac Lane, Saunders, 2010, *Categories for the Working Mathematician*, (Graduate Texts in Mathematics). Berlin: Springer.

[MiPa69] MARVIN MINSKY and SEYMOUR PAPERT, 1969 *Perceptrons. An Introduction to Computational Geometry* M.I.T. Press, Cambridge, Mass., 1969. vi + 258 pp.

[Neu45] Neumann, John Von, 1945 *Journal IEEE Annals of the History of Computing archive* Volume 15 Issue 4, October 1993 Page 27-75

[RaPa14] Raatikainen, Panu, "Gödel's Incompleteness Theorems", The Stanford Encyclopedia of Philosophy (Spring 2014 Edition), Edward N. Zalta (ed.), URL = ¡http://plato.stanford.edu/archives/spr2014/entries/goedel-incompleteness/¿.

[Raat14] Raatikainen, Panu, 2014, *Gödel's Incompleteness Theorems*, The Stanford Encyclopedia of Philosophy (Spring 2014 Edition), Edward N. Zalta (ed.),
URL = ¡http://plato.stanford.edu/archives/spr2014/entries/goedel-incompleteness/¿.

[RoF58] Rosenblatt, Frank, 1958, *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408. doi:10.1037/h0042519.

[RuMc86a] Rumelhart, D. E., McClelland, J. L., and the PDP research group, 1986 *Parallel distributed processing: Explorations in the microstructure of cognition* Volume I. Cambridge, MA: MIT Press.

[RuMc86b] McClelland, J. L., Rumelhart, D. E., the PDP research group, 1986 *Parallel distributed processing: Explorations in the microstructure of cognition* Volume II. Cambridge, MA: MIT Press.

[Sha11] Shaffer, Clifford A., 2011 *Data Structures and Algorithm Analysis*, Dover Edition

[SiS94] Siegelmann, Hava T. and Sontag, Eduardo D., 1994, *Analog computation via neural networks*, Theorectical Computer Science 131(1994) 331-360. Elsevier.

[Sip13] Sipser, Michael, 2013, *Introduction to the Theory of Computation*, Cengage Learning.

[TAM36] Turing, A.M. 1936 *On Computable Numbers, with an Application to the Entscheidungs problem*, Proceedings of the London Mathematical Society. 2 (1937) 42: 230–265. doi:10.1112/plms/s2-42.1.230. (and Turing, A.M. (1938). *On Computable Numbers, with an Application to the Entscheidungsproblem: A correction*, Proceedings of the London Mathematical Society. 2 (1937) 43 (6): 544–6. doi:10.1112/plms/s2-43.6.544.).

[WMWP98] Warren, S. McCulloch, Walter Pitts, 1988 *A logical calculus of the ideas immanent in nervous activity* Neurocomputing: foundations of research, Pages 15-27 MIT Press Cambridge, MA, USA ©1988 ISBN:0-262-01097-6

[WeD03] MathML 2.0 DTD, 2003,
http://www.w3.org/Math/DTD/mathml2/mathml2.dtd,
W3C, www.w3.org.

[WeD] Recommended List of Doctype declarations, DTD,
`http://www.w3.org/QA/2002/04/valid-dtd-list.html`,
W3C, `www.w3.org`.

[WDS] DTD Specification,
`http://www.w3.org/XML/1998/06/xmlspec-report.htm`,
W3C, `www.w3.org`.

[Wer74] P. J. Werbos, P. J., 1974 *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* PhD thesis, Harvard University.

[Wer94] Werbos, P. J., 1994 *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting* ISBN: 978-0-471-59897-8

[WFM] Focus Group of the ITU for study M2M, FG M2M,
`http://www.itu.int/en/ITU-T/focusgroups/m2m/Pages/default.aspx`,
W3C, `www.w3.org`.

[Vap95a] V. Vapnik, 1995, *The Nature of Statistical Learning Theory* Springer

[Vap95b] V. Vapnik, 1998 *Statistical Learning Theory* Wiley-Interscience, New York

[Vieira15a] Paulo Vieira, and Juan Corchado, 2915 *A Formal Machines as a Player of a Game*. DCAI: Distributed Computing and Artificial Intelligence, DCAI 2015: 137-147

[Vieira15b] Paulo Vieira, Juan Corchado and Sigeru Omatu, 2015 *Formal Machines and the Back Propagation Neural Networks*. submitted (on preparation) to: “Machine Learning Journal”, Springer 2015

[Vieira15c] Paulo Vieira, Adérito Alcaso, Carlos Carreto, Juan Corchado and Sigeru Omatu, 2015 *Embedded Systems, Artifical Intellgience and Fornal Machines*. submitted to: “Applied Intelligence”, Springer 2015

[YuB09] Yoshua Bengio, 2009 *Learning Deep Architectures for AI* Journal Foundations and Trends in Machine Learning archive Volume 2 Issue 1, January 2009 Pages 1-127

10 Appendix

10.1 Figure

The following figure illustrates the point b) of the definition of the instruction, I, of a FM (page 16)

$$\vec{c} = (c^1 \quad , \ldots, \quad \overbrace{(c_1^j, \ldots, c_{t_1}^j, \ldots, c_{t_{r_{iv}}}^j, \ldots, c_n^j)}^{c^j} \quad , \ldots, \quad c^n)$$

$$\downarrow I, \quad u \in R_{iv}, \quad R_{iv}(u_{r_{iv}+l}) = C_{iv}$$

$$c_P = \{c_1 \quad , \ldots, \quad \underbrace{(c_{i1}, \ldots, c_{w_1}, \ldots, c_{w_{s_{iv}}}, \ldots, c_{in})}_{c_i} \quad , \ldots, \quad c_t\}$$

with $u = (\underbrace{c_{t_1}^j}_{u_1}, \ldots, \underbrace{c_{r_{iv}}^j}_{u_{r_{iv}}}, \underbrace{c_{iw_1}}_{u_{(r_{iv}+1)}}, \ldots, \underbrace{c_{iw_l}}_{u_{(r_{iv}+l)=c_{iv}}}, \ldots, c_{iw_{s_{iv}}}) \in R_{iv}$

10.2 Data Structure of FA_FMs

When in the arguments of the *extension* and *understanding* methods is necessary to introduce a set as for example $\{a_1, a_2, a_3\}$ its introduced "$a_1;a_2;a_3$". Following I describe the DS of FA_FMs.

Methods that implement the DS of a FA_FM [73]

Methods for implementing the set $CompM_B$:
public void extensionM_B(String FM,String N_C,String U_C,String v_set)
public void understandingM_B(String FM,String N_C,String U_C,String px)

Methods for implementing the set $CompM_R$:
public void extensionM_R(String FM,String N_R,String Up,String relation)
public void understandingM_R(String FM,String N_R,String Up,String px)

Methods for implementing the set $ConfM$:
public void extensionConf(String FM,String N_conf,String conf)
public void understandingConf(String FM,String N_conf,String px)

Methods for implementing the set $InstM$:
public void extensionInst(String FM,String N_I,String _conf,String set_conf)
public void understandingInst(String FM,String N_I,String U_I,String domain,String Ic)

[73] The code and the documentation of the class ds_fa_fm can be queried and downloaded at: `http://www.ipg.pt/user/~pavieira/SW_Documentation/index.html`

The data_type conf:

```
public class conf {
        public char i;
        public char f;
        public String kind_of_conf;
        public String Q;
        public String T_I;
        public String p_I;
        /*
        * The constructor of the class data_type, conf
        * @param m m is 00, 01,10 or 11
        * @param q q is a state, q belongs to Q
        * @param u u is a word that is in the T_I of the FA_FM
        * @param n n is the place in the T_I where the pointer points
        */
        public conf(String m,String q, String u,String n){
                i=m.charAt(0);
                f=m.charAt(1);
                kind_of_conf=m;
                Q=q;
                T_I=u;
                p_I=n;
        }
}
```

The data_type productU:

```
    public class productU {
        String _q_1st;
        String _q_2nd;
        String _word;
        int _p_I_1st;
        int _p_I_2nd;
        /*
        * The constructor of the data_type productU. The set CompM_B of the
FA_FM is CompM_B=Q,T_I,p_I
        * @param _Q1 _Q1 is the component Q of the CompM_B
        * @param _T_I _T_I_I is the component T_I of the CompM_B
        * @param _p_I1 _P_I1 is the component p_I of the CompM_B
        * @param _Q2 _Q2 is the component Q of the CompM_B
        * @param _p_I2 _P_I2 is the component p_I of the CompM_B
        */
        public productU(String _Q1,String _T_I,int _p_I1,String _Q2,int _p_I2){
                _q_1st=_Q1;
                _q_2nd=_Q2;
                _word=_T_I;
```

```
            _p_I_1st=_p_I1;
            _p_I_2nd=_p_I2;
        }
}
```

10.3 Report about the PhD instance

With the aim to obtain to add of my PhD title international mention since April up to July I made a Phd instance in Portugal, in Guarda. the doctoral instance was held in the Technological and Management School[74] of the Polytechnic Institute of Guarda (IPG)[75]. The Phd instance was accompanied by two teachers of the School, by profesor Carlos Carreto (of the UTC of Informatic) and by profesor Adérito Alcaso (of the UTC of Environment and Energy). In the PHD instance I developed a tester of smells board and an Information System. The tester of smells consists in an electronic board with gas sensors and the Information system consists in collected data from the gas sensors and put it in Google Cloud to do an automatic data analysis. The idea was to do the data analysis using computational formal systems. Then I projected the data treatment necessary and I thought in what I can do with formal machines (figure). Thus, I implement a FM that are a union of those computational formal systems. The data analysis is made with three aims; i) to give alerts because the data can indicate that the board is damage, ii) to give alerts because the data can indicate that the sensors are no calibrated or are damage and iii) to give alerts about certain values collected by the sensors. The work consisted in create a way of mechanize all of this. This mechanization will allow to do similar works in easy way. In this report I will describe it.

The build of this system will create databases, in Google Cloud, of values of the gases collected by the sensors. The aim of create databases in Google Cloud, or in another Cloud, with the data collected from real situation and in Real time is to analysis the data, to obtain knowledge from them and to act upon the environment in real time in an intelligent way. I also create an implementation of the CSFM to the Google Cloud and I wrote it in a google script language. This allows that the some of the data analysis can be done by a FM. The interest of doing this through of a FM is because the FM formalism is a computational system where is possible to rewrite any other FCS. The FMs have associated several measures related with the idea of Intelligence, the MIQ (Machine Intelligent Quotient) measures. The idea of MIQ is to have to the machine an analogue of the IQ of the humans or in late sense of the biological beings. The MIQ allows to measure, in formal systems, characteristics that usually is looked as intelligent behaviors. Thus is possible to evaluated the intelligence of the information system that will be mounted for similar situation. I can answer to the question. How much the system is intelligent?

[74]http://portal.ipg.pt/webapps/portal/frameset.jsp-
[75]www.ipg.pt

Based on that I can decided, How much intelligent I want put in a system to solve a problem.

10.3.1 The iGases

The iOlphat system is a personal project and is a project about intelligent smell technologies. The iGases system is part of the iOlfact system. The iGases consists physically of an electronic board connected to an Internet cloud platform. The system measures several levels of gases concentration, processes data in real time and provides the resulting information in the Internet. The iGases is part of my PhD work.

Motivation:
There are three types of motivations to do this work. First, I have a personal project the iOlphat about to design and develop smell technologies. Second, the act of smell or sniff I see as an act of prove smells. Prove smells can be very useful, with this capacity you is able to distinguish fragrances, odors, smells in general. The smells provoke in biological beings a lot of sensations, allow to them distinguish substances and can be a qualitative way of distinguish different levels of a concentration of a gas. Some gases concentrations can be dangerous and harmful to biological life and can react with other substances. The other type of motivation is because the system is projected to use formal programming methods. The use of formal methods, in systems that are designed to solve engineering problems is very important since a lot of mathematical mechanisms are available for use in the solutions. This is important in the moment that the problem is to be solved, because a lot of things are known about the formalism and is possible to use them. Other advantage of this strategy is that other researchers can easily add new functionalities/improvements to the solution found since that the formal methods are known by the scientific community and developers.

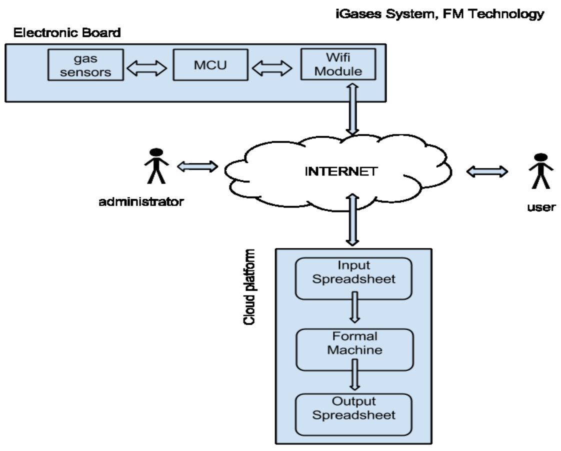


Figure 62: FM Tecnhology: Design of the iGases System

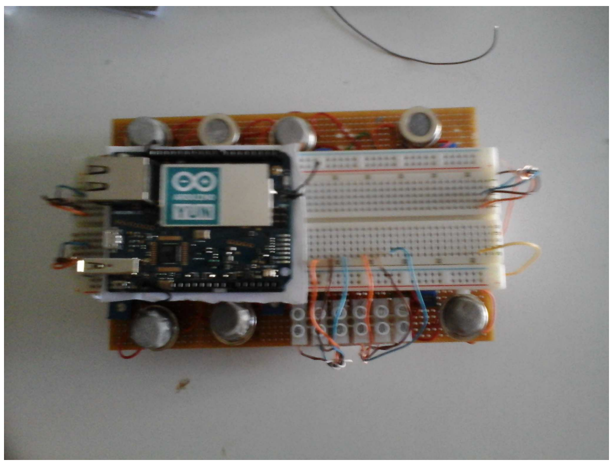
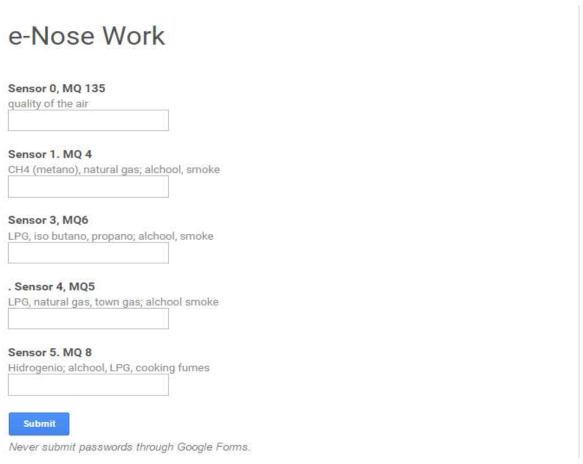


Figure 63: FM Tecnhology: The e-nose board



e-Nose Work

Sensor 0, MQ 135
quality of the air

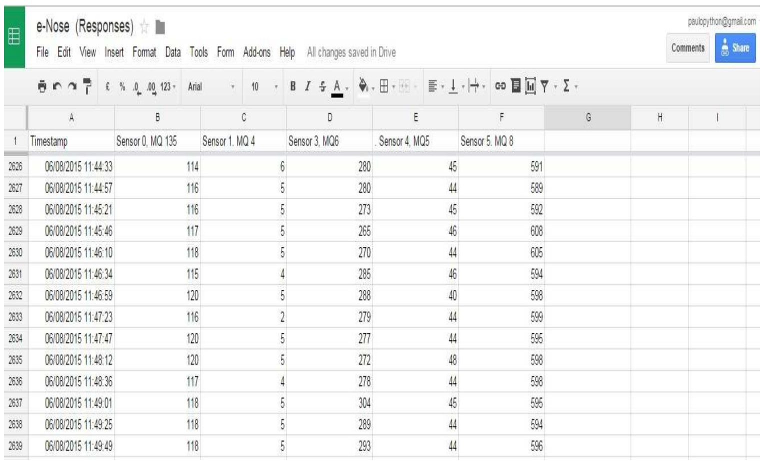
Sensor 1. MQ 4
CH4 (metano), natural gas; alchool, smoke

Sensor 3, MQ6
LPG, iso butano, propano; alchool, smoke

. Sensor 4, MQ5
LPG, natural gas, town gas; alchool smoke

Sensor 5. MQ 8
Hidrogenio; alchool, LPG, cooking fumes

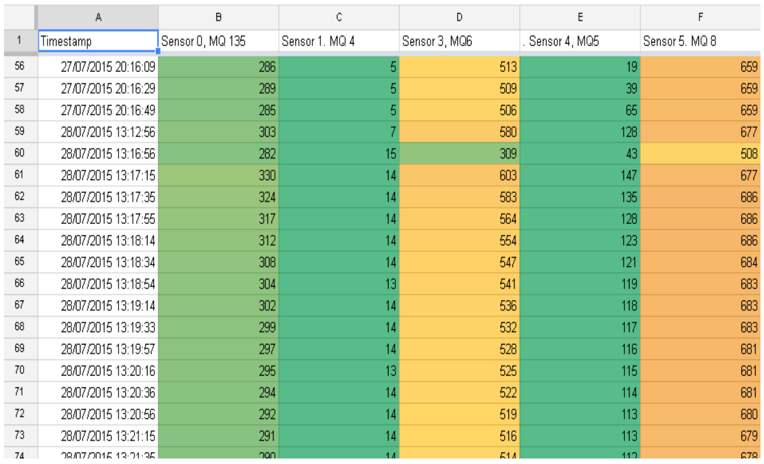
Figure 64: FM Tecnhology: Google cloud. An interface to introduce values in no automatic way. https://goo.gl/oqk51o



e-Nose (Responses)
paulopython@gmail.com
File Edit View Insert Format Data Tools Form Add-ons Help All changes saved in Drive
Comments
Share

	A	B	C	D	E	F
1	Timestamp	Sensor 0, MQ 135	Sensor 1. MQ 4	Sensor 3, MQ6	. Sensor 4, MQ5	Sensor 5. MQ 8
2626	06/08/2015 11:44:33	114	6	280	45	591
2627	06/08/2015 11:44:57	116	5	280	44	589
2628	06/08/2015 11:45:21	116	5	273	45	592
2629	06/08/2015 11:45:46	117	5	265	46	608
2630	06/08/2015 11:46:10	118	5	270	44	605
2631	06/08/2015 11:46:34	115	4	285	46	594
2632	06/08/2015 11:46:59	120	5	288	40	598
2633	06/08/2015 11:47:23	116	2	279	44	599
2634	06/08/2015 11:47:47	120	5	277	44	595
2635	06/08/2015 11:48:12	120	5	272	48	598
2636	06/08/2015 11:48:36	117	4	278	44	598
2637	06/08/2015 11:49:01	118	5	304	45	595
2638	06/08/2015 11:49:25	118	5	289	44	594
2639	06/08/2015 11:49:49	118	5	293	44	596

Figure 65: Google cloud: Input Spreadsheet Day (ISD) of the sensors values. https://goo.gl/z2TJZ1



	A	B	C	D	E	F
1	Timestamp	Sensor 0, MQ 135	Sensor 1. MQ 4	Sensor 3, MQ6	. Sensor 4, MQ5	Sensor 5. MQ 8
56	27/07/2015 20:16:09	286	5	513	19	659
57	27/07/2015 20:16:29	289	5	509	39	659
58	27/07/2015 20:16:49	285	5	506	65	659
59	28/07/2015 13:12:56	303	7	580	128	677
60	28/07/2015 13:16:56	282	15	309	43	508
61	28/07/2015 13:17:15	330	14	603	147	677
62	28/07/2015 13:17:35	324	14	583	135	686
63	28/07/2015 13:17:55	317	14	564	128	686
64	28/07/2015 13:18:14	312	14	554	123	686
65	28/07/2015 13:18:34	308	14	547	121	684
66	28/07/2015 13:18:54	304	13	541	119	683
67	28/07/2015 13:19:14	302	14	536	118	683
68	28/07/2015 13:19:33	299	14	532	117	683
69	28/07/2015 13:19:57	297	14	528	116	681
70	28/07/2015 13:20:16	295	13	525	115	681
71	28/07/2015 13:20:36	294	14	522	114	681
72	28/07/2015 13:20:56	292	14	519	113	680
73	28/07/2015 13:21:15	291	14	516	113	679
74	28/07/2015 13:21:35	290	14	514	112	678

Figure 66: Google cloud: Example of an Output Spreadsheet Day (OSD) of the sensors values. https://goo.gl/90JWv6

Objectives:
i) - To build a system that is able to reading concentrations of different gases and put the information in real time in internet.
ii) - To do data analysis with the data collected.
iii) - The data processing is done through of a formal methods. The formal method used is one that was projected in the work of Paulo's PhD. This formal method is a computational system called Formal Machine (FM).
iv) - The administrator of the system can start, stop and configure the system through the Internet.

Results Obtained: The system created was tested widely in laboratory conditions and their application in real environment will be done in a posterior phase. The system detect gases in environment, and classify (in a qualitative way), different levels of concentrations, generate alerts (alert e-mails), check the states of the sensors (whether they are calibrated or damaged), and verify any damage on the electronic board. All this information is based in the processing that is done from the data sensor values gathered. The output results obtained in real time after FM's processing, contains useful information that is not present in original inputs. This is an evidence of the utility of formal methods in data processing, in particular to the FM computational model.

Electronic board(figure 63): The electronic board is composed of a microcontroller, inexpensive gas sensors, and a Wi-Fi hook up to an Internet connection. The sensors read the concentration levels of gases as a value and the Wi-Fi module inserts these readings in a spreadsheet in a cloud.

Cloud platform(figure 64,65,66): In the cloud platform there are two spreadsheets; the input spreadsheet and the output spreadsheet. The input spreadsheet receives the data from the Wi-Fi module, as a private object. The output spreadsheet shows treated information to the users. I chose to work with spreadsheets because they have a sufficient storage system to the amount of data which are collected and exist a historical API, in Spreadsheets (Excel and so on), that allow a good mathematical work with data. In the cloud, the input spreadsheet receives the sensor data values sent from the electronic board. After this, the data is processed through a Formal Machine (FM). The FM production goes to the output spreadsheet and, based on the data analysis, the FM does a qualitative classification of the environment, generates alerts, checks the sensors and uses a set of algorithms to determine whether they are calibrated or damaged, and verify any damage on the electronic board.

i) **Building the iGases, the e-Nose**

In following you can see some pictures about the building of the e-nose in different moments.

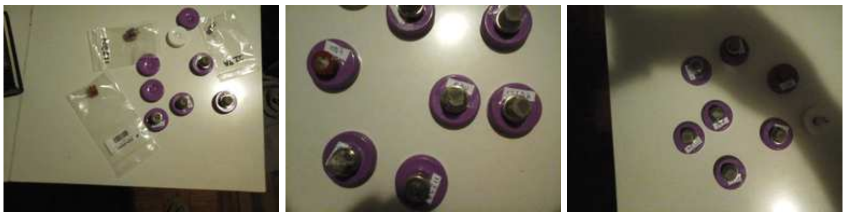
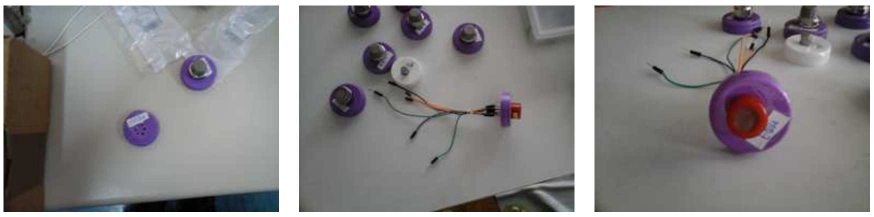


Figure 67: FM Technology: bottle stoppers serves as a socket to gas sensors

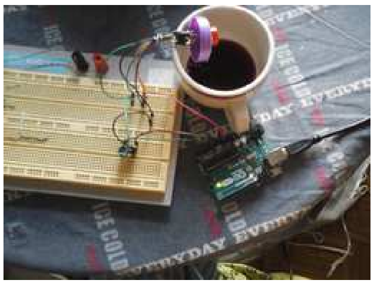


(a) FM Technology: the bottle stopper of the MQ135 gas sensor

(b) FM Technology: the MQ135 gas sensor connectors

(c) FM Technology: The gas sensor MQ135 in a front view

Figure 68: FM Technology: the MQ135 gas sensor


(a) FM Technology: Experience connection, the MQ135 smelling alcohol, 1

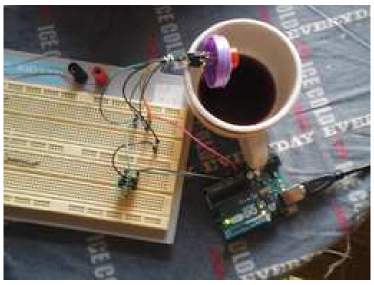
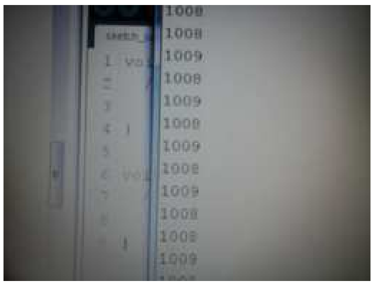
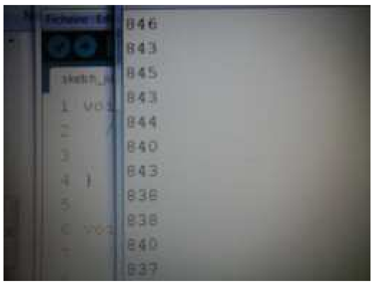

(b) FM Technology: Experience connection, the MQ135 smelling alcohol, 2

Figure 69: FM Technology: MQ135 smelling alcohol, experience



```
1008
1008
1009
1008
1009
1008
1009
1008
1009
1008
1008
1009
```

(a) FM Technology: MQ135 smelling alcohol, values



```
846
843
845
843
844
840
843
838
838
840
837
```

(b) FM Technology: MQ135 no smelling alcohol, values

Figure 70: FM Technology: reading values from MQ135

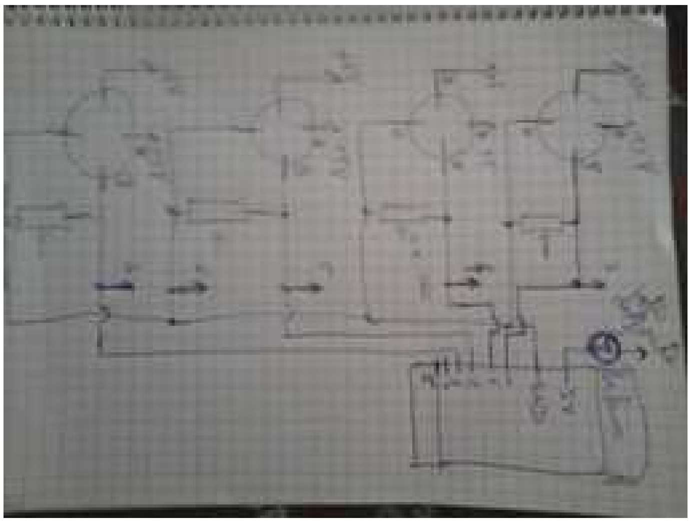
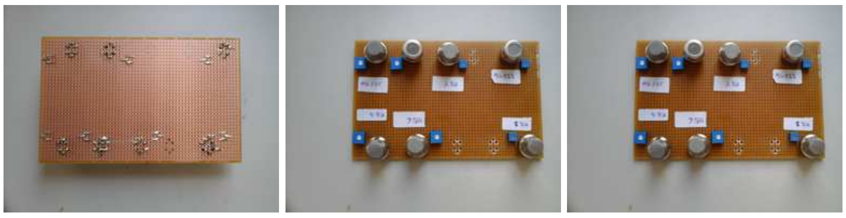


Figure 71: FM Technology: the first e-nose circuit, draft

(a) FM Technology: working in the laboratory, soldering iron (b) FM Technology: the gases sensors in the laboratory (c) FM Technology: the pcb of the e-nose in the laboratory

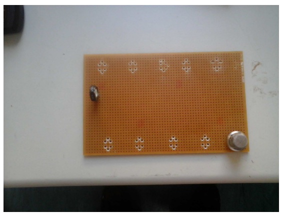
Figure 72: FM Technology: working in the laboratory

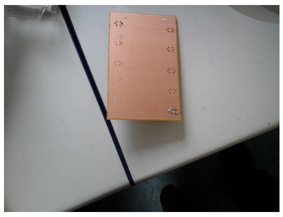


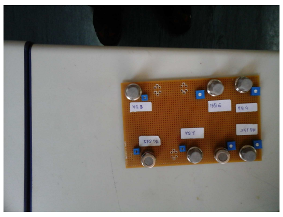
(a) FM Technology: the first e-nose board, back view (b) FM Technology: the first e-nose board, front view 1 (c) FM Technology: the first e-nose board, front view 2

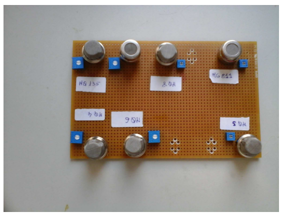
Figure 73: FM Technology: the first e-nose board

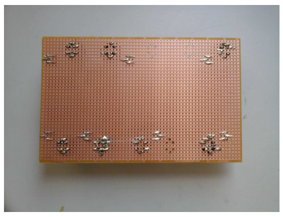
The processes of building the e-nose in pictures

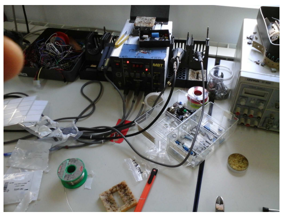

(a) FM Technology: Building the e-nose, 1

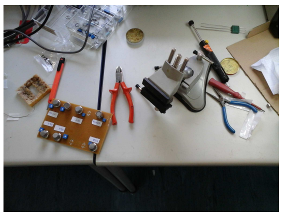

(b) FM Technology: Building the e-nose, 2

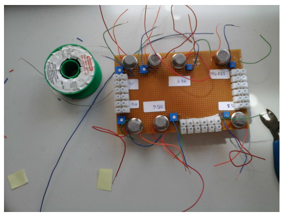

(c) FM Technology: Building the e-nose, 3

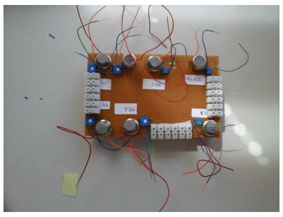

(a) FM Technology: Building the e-nose, 4

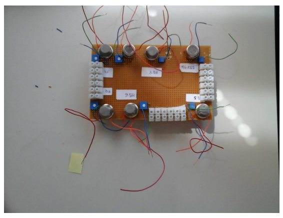

(b) FM Technology: Building the e-nose, 5

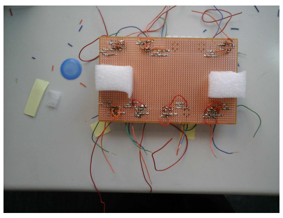

(c) FM Technology: Building the e-nose, 6

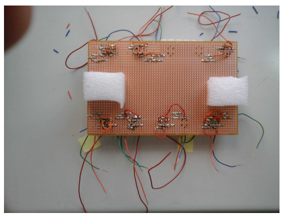

(a) FM Technology: Building the e-nose, 7

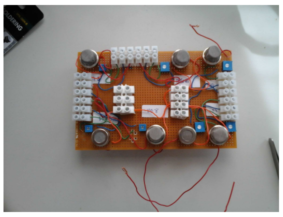

(b) FM Technology: Building the e-nose, 8

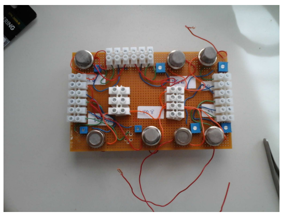

(c) FM Technology: Building the e-nose, 9

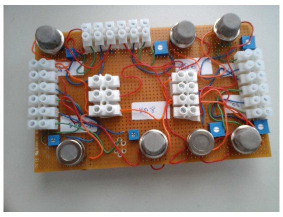

(a) FM Technology: Building the e-nose, 10

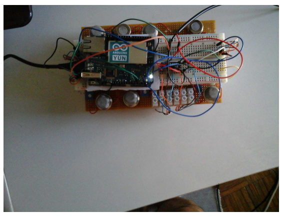

(b) FM Technology: Building the e-nose, 11

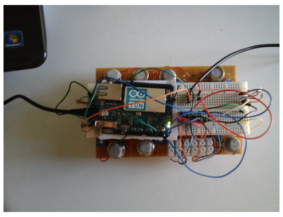

(c) FM Technology: Building the e-nose, 12


(a) FM Technology: Building the e-nose, 13


(b) FM Technology: Building the e-nose, 14


(c) FM Technology: Building the e-nose, 15


(a) FM Technology: Building the e-nose, 16


(b) FM Technology: Building the e-nose, 17

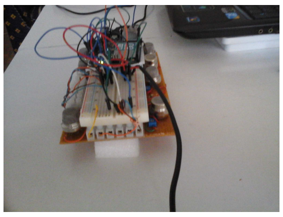

(c) FM Technology: Building the e-nose, 18

Table 30: gas sensors used in the e-nose

Sensors	High sensibility	Small sensibility	features
MQ 5	LPG, natural gas, town gas	alcohol, smoke	Fast response, Stable and long life, simple drive circuit
MQ 6	LPG, iso butane, propane	alcohol, smoke	Fast response, Stable and long life, Simple drive circuit
MQ 4	CH4, Natural gas	alcohol, smoke alcohol, smoke	Fast response, Stable and long life, Simple drive circuit
MQ 135	NH3, NOx, alcohol, Benzene, smoke,CO2 ,etc		Wide detecting scope, Fast response, High sensitivity Stable, long life Simple drive circuit
MQ 8	Hydrogen (H2)	alcohol, LPG,cooking fumes	Stable and long life
MG 811	Good sensitivity and selectivity to CO2		Low humidity, temperature dependency, Long stability, reproducibility

ii) Building the iGases, the Cloud

In iGases system, seen as an information system, the system alerts are implemented with processing done by a FM. The alerts system is the first implementation of the FM Technology in the cloud. This implementation involves an architecture that includes a Finite Automaton (FA) and three Hypotheses Testing (HT); HT1, HT2 and HT3. The finite automaton is responsible for identify whether there are sensor values that exceed certain limits. The hypotheses testing are responsible for identifying the states of the sensors and the board. The HT1 is used to identify whether there are Sensors are Out of Calibration (SOC), the HT2 is used to identify whether there are Damaged Sensors (DS) and HT3 is used to identify whether the Board is Damaged in (BD).

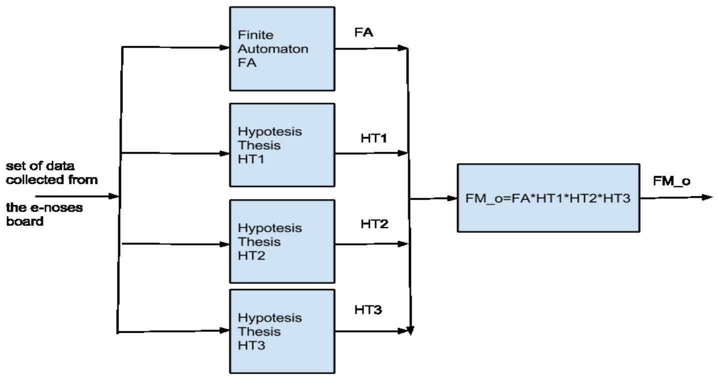


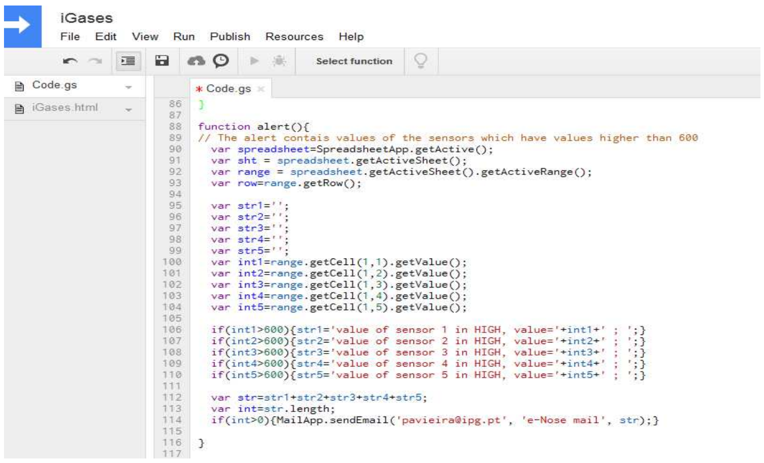
Figure 80: FM Technology: Architecture of the System implemented in the FM

So, if there are sensors out of calibration SOC = 3, if there are not SOC = 1. For damaged sensors DS = 5, if there are not DS = 1. If the board is damaged BD = 7, if not BD = 1. The output value of the FM (FM_o) is given by FM_o = FA*SOC*DS*BD and runs every hour.

Table 31: FM Technology: Table of output values of the FM

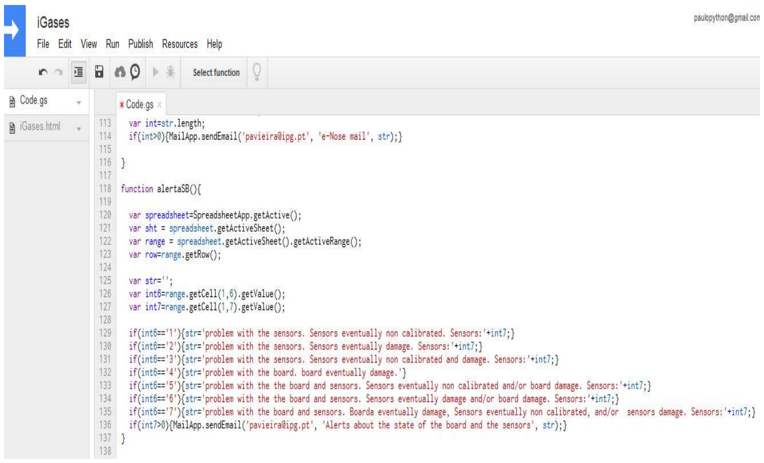
Computational Systems	Output, no problem value	Output, problem value
FA	FA=1	FA=2
HT1	SOC=1	SOC=3
HT2	DS=1	DS=5
HT3	BD=1	BD=7

There is a script that executes the FM every hour. If FM_o = 1 nothing is executed if FM_o>1 are called two script functions for sending alerts; alertSensorValues() and alertDamage(). In the implementation of the FM were chosen prime numbers to identify the different types of problems and the output value of the FM is the multiplication of these numbers. The decomposition in prime factors of the output value obtained allows, to the scripts alertSensorValues() and alerDamage(), to know the problems identified by the FM.



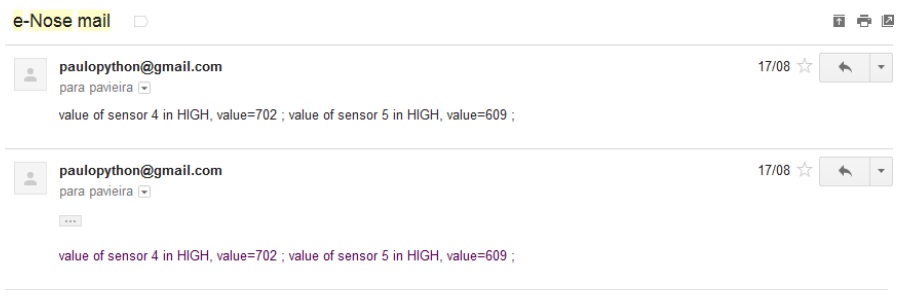
```
}

function alert(){
// The alert contais values of the sensors which have values higher than 600
  var spreadsheet=SpreadsheetApp.getActive();
  var sht = spreadsheet.getActiveSheet();
  var range = spreadsheet.getActiveSheet().getActiveRange();
  var row=range.getRow();

  var str1='';
  var str2='';
  var str3='';
  var str4='';
  var str5='';
  var int1=range.getCell(1,1).getValue();
  var int2=range.getCell(1,2).getValue();
  var int3=range.getCell(1,3).getValue();
  var int4=range.getCell(1,4).getValue();
  var int5=range.getCell(1,5).getValue();

  if(int1>600){str1='value of sensor 1 in HIGH, value='+int1+' ; ';}
  if(int2>600){str2='value of sensor 2 in HIGH, value='+int2+' ; ';}
  if(int3>600){str3='value of sensor 3 in HIGH, value='+int3+' ; ';}
  if(int4>600){str4='value of sensor 4 in HIGH, value='+int4+' ; ';}
  if(int5>600){str5='value of sensor 5 in HIGH, value='+int5+' ; ';}

  var str=str1+str2+str3+str4+str5;
  var int=str.length;
  if(int>0){MailApp.sendEmail('pavieira@ipg.pt', 'e-Nose mail', str);}

}
```

Figure 81: FM Technology: alert sent about high values collected by the sensors. script function alertSensorValues().



```
  var int=str.length;
  if(int>0){MailApp.sendEmail('pavieira@ipg.pt', 'e-Nose mail', str);}

}

function alertaSB(){

  var spreadsheet=SpreadsheetApp.getActive();
  var sht = spreadsheet.getActiveSheet();
  var range = spreadsheet.getActiveSheet().getActiveRange();
  var row=range.getRow();

  var str='';
  var int6=range.getCell(1,6).getValue();
  var int7=range.getCell(1,7).getValue();

  if(int6=='1'){str='problem with the sensors. Sensors eventually non calibrated. Sensors:'+int7;}
  if(int6=='2'){str='problem with the sensors. Sensors eventually damage. Sensors:'+int7;}
  if(int6=='3'){str='problem with the sensors. Sensors eventually non calibrated and damage. Sensors:'+int7;}
  if(int6=='4'){str='problem with the board. board eventually damage.'}
  if(int6=='5'){str='problem with the the board and sensors. Sensors eventually non calibrated and/or board damage. Sensors:'+int7;}
  if(int6=='6'){str='problem with the the board and sensors. Sensors eventually damage and/or board damage. Sensors:'+int7;}
  if(int6=='7'){str='problem with the board and sensors. Boarda eventually damage, Sensors eventually non calibrated, and/or  sensors damage. Sensors:'+int7;}
  if(int7>0){MailApp.sendEmail('pavieira@ipg.pt', 'Alerts about the state of the board and the sensors', str);}
}
```

Figure 82: FM Technology: Alert sent about the state of the sensors and the board. script function alertDamage().


e-Nose mail

paulopython@gmail.com
para pavieira
17/08
value of sensor 4 in HIGH, value=702 ; value of sensor 5 in HIGH, value=609 ;

paulopython@gmail.com
para pavieira
17/08
value of sensor 4 in HIGH, value=702 ; value of sensor 5 in HIGH, value=609 ;

(a) FM Technology: Alert send about high values of the sensor

The system administrator can interact with the system in two ways; locally and through of a web application. Locally is on the device through of buttons to connected, disconnected and restored the iGases system. In the application web is through of an url.

In the application web, the system administrator can interact with the two parties that make up the system; on the e-nose device and on the information system. The actions on the device are done through the commands implemented in a web interface. The action in the device are made through of the iGases Device Commands. The device can be connected, disconnected or restarted. The actions in the information system are made through of the iGases Information System Commands. The information system can be connected, disconnected or restarted.

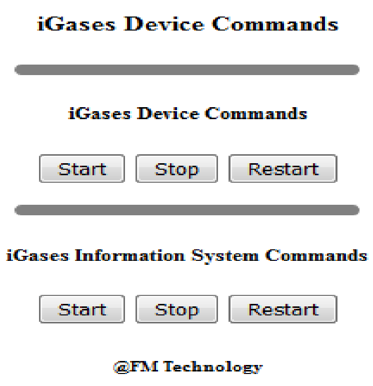


Figure 84: FM Technology: Architecture System about the FM implementation

The iGases system have control priorities. The local commands are the higher priority commands. Only if the e-nose is connected is possible to use the device commands, and the information system commands. The device commands and the information system commands are independent and are connected by default.

iii) How working the iGases system

Table 32: FM Technology: Temporal diagram of the Spreadsheets on cloud

Temporal diagram	8 am	...	9 am	...	10 am	...	1 pm	...	2 pm	...	8 pm	...	9 pm	...
ISD Input Spreadsheet Day (figure 65)		collected data	sends data to the OSD	collected data	sends data to the OSD	collected data	sends data to the OSD	collected data	sends data to the OSD	collected data	sends data to the OSD	auto clean the ISD sends data to the AS	sleeping	sleeping
OSD Output Spreadsheet Day (figure 66)	sleeping	sleeping	receives data from OSD Eventually sends alerts	sleeping	receives data from OSD Eventually sends alerts	sleeping	receives data from OSD Eventually sends alerts	sleeping	receives data from OSD Eventually sends alerts	sleeping	receives data from OSD Eventually sends alerts	auto clean the OSD	sleeping	sleeping
AS Archive Spreadsheet `https://goo.gl/sX91RE`	sleeping	sleeping	sleeping	sleeping	sleeping	sleeping	sleeping	sleeping	sleeping	sleeping	sleeping	receives data from ISD	sleeping	sleeping

The iGases system in an automated working day wakes up at 8 am and closes at 9pm. Then goes into hibernation until 8am the next day. 8am to 8pm the system collects data and sends alerts if necessary. 8pm to 9pm, in back-office, the system performs a set of tasks of the end day work and prepares the next day.

The data collection of 8am to 8pm, is done by the e-nose board collects, the e-nose puts the data into the Input Spreadsheet Day. Every hour there is a script in the Input Spreadsheet Day that copies the data to the Output Spreadsheet Day. The FM is implemented in the Input Spreadsheet Day. From the output Spreadsheet Day are eventually sent alerts to the users. In the system there are two types of users the PRIME users who have access to Input and Output Spreadsheets Day and to the Database and the others who have access only to Output Spreadsheet Day.

8pm to 9pm the system copies the data from the Input Spreadsheet Day to a file and deletes the data of the Input and Output Spreadsheets Day. The data file is a spreadsheet and is on a folder in the Cloud. In this folder they are stored spreadsheets named by month and year. In each spreadsheet the data are stored by day, each day correspond to a sheet of the spreadsheet. In each day the data are stored by the time stamp, timestamp example 27/07/2015 19:19:17.

The iGases is turned on in automatic mode when the e-nose board in the manual mode is ON and in the web platform the e-nose and the information system are ON. When the e-nose is turned on in manual mode and one of the systems on the web platform is disconnected, the system is in configuration mode. When the e-nose is off in manual mode, the system is off.

iv) Conclusion and Future work

As initially I said, this work the iGases is part of a personal project called iOlphat that consists in create intelligent technology to smell. The iGases system developed is not in the process of being placed on the market, for that it needs to be more robust, smaller and more versatile. The next version of iGases will be directed to have a prototype to place on the market. Thus, in the next version I will improve and I will work on the following:

- Create in the web application a system configuration button and implement it
- Make the e-nose board more robust in the electronic level
- Make a the e-nose to smaller size
- Make e-nose board more versatile in the sense that make it is easy to change the gas sensors
- Put on the e-nose board a connector to enable connection to smartphones and other mobile devices
- Create a smartphone application that takes advantage of sensors are in mobile device.

Thus, I end the description of my job in my PhD instance in the Polytechnic Institute of Guarda, Portugal. In this instance I developed the first version of the system that I called iGases.

10.4 Terms and Abbreviations

BPNN - Back Propagation Neural Network
DB - Database
DBs - Databases
DS - Data Structure
DSs - Data Structures
CVN - Cycle Von Neumann
CVNs - Cycles Von Neumann
DTD - Document Type Definition
FA_FM - Finite Automaton_Formal Machine
FA_FMs - Finite Automata_Formal Machines
FCS - Formal Computational System
FCSs - Formal Computational Systems
FM - Formal Machine
FM's DB - Formal Machine's Database
FMs - Formal Machines
FOL - First Order Logic
HOL - High Order Logic
IoT - Internet of Things
ML - Machine Learning
MLs - Machines Learning
m2M - man-to-Machine
M2M - Machine-to-Machine
NNM - Neural Network Machine
NNMs - Neural Network Machines
PL - Propositional Logic
VN_{Alg} - Von Neumann's Algorithm
wff - well Formed Formula
wffs - well Formed Formulas