Overview of the PhD Research 2014/2015

Ricardo Henrique Gracini Guiraldelli
ricardo.guiraldelli@univr.it

Foreword

This document is a replacement for the overview presentation on my PhD research executed in the year of 2014 and the first three months of 2015. It is a suggestion from professor Vincenzo Manca and agreed by all members of my evaluating committee, *i.e.* professors Vincenzo Manca, Gloria Menegaz and Giuditta Franco.

The structure chosen to present the evolution of the PhD research is the same one that will be presented in the thesis that will be composed to the end of the program: each of the following sections represents a planned chapter in the future thesis and their contents are brief description of the contents of the thesis. This presentation style has the advantage of (i) showing the overall results of the PhD research; (ii) focusing on the developed activities in the last two years; (iii) specifying the limits of the current research; (iv) explaining the strategy to properly finish the on-going tasks. Also, at the end, there are two attachments: a table of the present state of the main activities and the submitted article to the *special edition on synthetic biology of the IEEE Transactions on Biomedical Circuits and Systems.*

As a summary document, the contents are briefly explained in order to clarify to the committee the evolution of my research work in 2014 and first months of 2015; it is possible, nonetheless, that some information is not as clear to the readers as it is to author, mistaken or even missing. In this case, I would like to apologise in advance and state that I am totally available for you to comment, justify or explain any of your questions.

1 Introduction

The present document is a discourse on the scientifical aspect of metabolic systems and their computational nature. It goes beyond the standard and pervasive procedure of modelling biological systems with assistance of powerful computational resources. It is an application of computational theory to metabolic systems in order to establish, solidify and support the synergy between biology and technology, in a bidirectional way.

For this purpose, the main results of the present work are of theoretical nature. A series of definitions, theorems and abstract-level algorithms build a solid foundation for existing, proposed and future development of products in this interface field. Mostly, this part of the research is presented in the Sections 2 to 5.

Nonetheless, we also present real world applications in a variety of forms, either as algorithm (Section 6), hardware (Section 5) or software (Section 7), making it clear the vast influence and scope of the present work.

Although very thrilling, it is important to remember the present work is, as well as the whole research field, very fresh and still on-going. However, its potential and auspiciousness is splendid, what can be verified by the increase interest in natural and membrane computing [15], bio-inspired devices [19] and synthetic biology [22].

2 Membrane Computing through Metabolic P Theory

Metabolic P (or MP) system is a kind of membrane computing formalism adapted to model biological process [11, p. 64], specially the metabolical one. As one of the cell-like systems of membrane computing, it inherits the three basic structures [16, Section 2]: (i) a *membrane structure*, which serves as an isolator of the system from the outer environment as well as a gateway for input and output of objects; (ii) *multisets* of objects, which represent the objects that may be manipulated by the system; and (iii) *rules*, which define the way the objects are transformed and have biochemical inspiration.

It is also defined as a discrete, dynamical system that, at each step of execution, parallelly process all its rules and updates the values of its variables [10, Chapter 3]. Both perspectives of MP system are present in its formal definition.

For a clear and constructive comprehension of the system, let us see some fundamental and concepts that composes the MP system.

Definition 1 (MP grammar)**.** An MP grammar G is a generative grammar for time series defined as

$$G = (M, R, I, \Phi)$$

where:

1. $M = \{x_1, x_2, \ldots, x_n\}$ the finite set of substances (or metabolites), and $n \in \mathbb{N}$ the quantities of substances.
2. $R = \{\alpha_j \to \beta_j \mid 1 \leq j \leq m\}$ the set of rules (or reactions), with α_j and β_j multisets over M, and $m \in \mathbb{N}$ the number of reactions.
3. $I = (x_1[0], x_2[0], \ldots, x_n[0])$ is the vector of initial values of substances or the metabolic state at initial step (step 0).
4. $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$ is a set of functions (also called regulators), in which every $\varphi_j : \mathbb{R}^n \mapsto \mathbb{M}$, for $1 \leq j \leq m$, is associated with a rule $r_j \in R$.

MP grammar is the fundamental (static) component of MP systems and it presents all the elements to define the model as a membrane computing one. The grammar G itself specifies the membrane as the confinement of all the operations; the set of metabolites M along with the initial state I characterise the multisets. Finally, the sets of rules R and regulators Φ establish the membrane computing rules.

In the fixed definition of grammar there is also a hint for dynamical operations: the regulators in the set Φ. The regulators are defined as recurrent functions [10, Section 3.1] which are useful for updating variable values in discrete dynamical systems [7, Chapter 2]. Nonetheless, for a full definition of the system, more dynamical elements must be introduced.

Definition 2 (MP system)**.** A *MP system* M is a discrete dynamical defined as

$$\mathcal{M} = (G, \tau)$$

with

1. G being an MP grammar following the definition 1;
2. $\tau \in \mathbb{R}$, the period (amount of time) of a computational step;

The original definition of MP system [10, p. 109] required the specification of other two components, the number ν of conventional mole and the vector μ of mole masses. These, however, are marginally profitable for the model and are ignored without loss of generality.

Important, though, is the procedure to compute the discrete steps of the dynamical MP system. The mathematical expression responsible for it is the *equational metabolic algorithm* which is founded on the concept of the *stoichiometric matrix.*

Definition 3 (Stoichiometric matrix)**.** Let $r_i = \alpha_i \to \beta_i$, where α_i (with an equivalent for β_i) is represented as $\sum k^+_{i,j} \times X_j \mid k_{i,j} \in \mathbb{N} \wedge X_j \in M$.

Let $\text{mult}^+(X_j, r_i) = k^+_{i,j}$ be the multiplicity, for the right side (α_i) of the rule r_i, of the substance X_j in the rule. Similarly, there is $\text{mult}^-(X_j, r_i) = k^-_{i,j}$ for the left side (β_i) of the rule.

A stoichiometric matrix $\mathbb{A}$, of dimension $|M| \times |R|$, has each of its elements defined by

$$a_{l,m} = \text{mult}^+(X_l, r_m) - \text{mult}^-(X_l, r_m)$$

with , $1 \leq l \leq |M|$ and $1 \leq m \leq |R|$.

Definition 4 (Equational Metabolic Algorithm—EMA)**.** Let $U[i] = (\varphi_1(i), \varphi_2(i), \ldots, \varphi_m(i))^T$ be the vector of values, in the time step i, of all regulators, and $\mathbb{A}$ the stoichiometric matrix.

The *vector of substance variation at step* i, $\Delta[i]$, is computed by the equation

$$\Delta[i] = \mathbb{A} \times U[i]$$

so-called *Equational Metabolic Algorithm* whom computes the value of any substance in the time future time step $i + 1$ through the recurrent equation

$$X[i + 1] = X[i] + \Delta[i]$$

As we can see, the stoichiometric matrix $\mathbb{A}$ correlates the metabolites of M with all the rules belonging to R, clearly indicating whether the objects are consumed or produced by each rule. It is a MP system analogy to incidence matrix from graph theory [1, p. 54] or Petri networks [17, Section 11.2].

The EMA, on the other hand, is the core of the dynamical process. For each step $i \in \mathbb{N}$, it computes the discrete variation of each variable (represented by $\Delta[i]$) and updates their values in the recurrence equation.

It may be clear at first sight, but this new membrane computing and dynamical system formalism presents a series of advantageous properties for the modelling task. In order to name a few,

- it uses *recurrence equations* in the place of differential equations, which shifts the model from continuous to discrete domain of time to ease either the design and computation tasks;
- the definition of the *grammar* precisely describe the system, its elements and explicit all their relations;
- the notation used to describe the *rules* have a intentional resemblance with biochemical equations and, at the same time, with formal grammars, which allows it to be a bridge between these two different fields of knowledge.

Nevertheless, in its general form, MP system is too broad for application on some scenarios and a more convenient variation is required. This is particularly true when modelling cellular behaviour, when three constraints are of central importance: (i) the fluxes must be computable functions, an absent restriction in the general definition of MP systems; (ii) no metabolite can have negative quantity; (iii) inactivation of rules when its (right-hand side) metabolites are insufficient to perform it, resulting in a chain of broken reactions. For this kind of situations, we have defined a subclass of MP systems which we have named as *Positively Controlled Metabolic P systems*, or MP$^+$ for short, which presents few controls to ensure all operations are properly computed in the set $\mathbb{N}$ of natural numbers.

Definition 5 (MP$^+$ Grammar)**.** A MP$^+$ grammar $\mathcal{G}_+$ is a standard MP grammar $\mathcal{G}$ respecting the following restrictions, at each computational step s_i:

1. $\varphi|_{s_i} = \begin{cases} \kappa & , \text{ if } \kappa \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$, for all $\varphi \in \Phi$;
2. $\sum_{\varphi|_{s_i} \in \Phi_x^-} \varphi \leq x$, with Φ_x^- the set of all fluxes related to rules in which the metabolite x is in the left-hand side of the rule; otherwise, $\varphi = 0, \forall \varphi \in \Phi_x^-$ at the execution step.

This kind of definition, as it will be seen later, is deeply related with the notion of computability in computer science [9, Section 4.7; 14, Chapter 11].

3 Computational Devices

Computational devices are mathematical models for automatic execution of procedures (which are better interpreted as *devices*). The procedures are detailed instructions of how to perform the task programmed for the model. The modern concept of computational device derives from Turing in its attempt to answer the *Entscheidungsproblem*.

The Turing machine is the reference model for computability [3, Section 2.4]. It is a finite-state machine composed of tape divided in unitary blocks (its memory units) and a head that performs four operations: (i) read the tape; (ii) write in the tape; (iii) move left along the tape; or (iv) move right along the tape. Each of its operations are restrict to a single unitary block and just operation can be executed at time.

This simplistic model, however, leads to the strong result of which procedures are computable and the notion of algorithm. According to the Church-Turing thesis, every computable procedure that halts on all inputs is an algorithm and, hence, executed by a Turing machine (or equivalent models) [9, p. 246; 21, Figure 3.22, p. 183]. This thesis makes the Turing machine (and equivalent models) as the most powerful computational devices in algorithms (hence, *procedures that halts*) term, including general purpose computers, digital circuits implementations, advanced mathematical simulations among several other examples.

Although useful in a series of mathematical manoeuvres, the Turing machine model is less suitable for situations in which abstractions closer to digital computer (von Neumann) architectures or pseudo-code is required because of the highly detailed level of its instructions, designed to manipulate the head and the single alphabet value (bit) of its tape. In such situations, different but equivalent models are used.

One of these Turing-equivalent models that resembles a digital computer architecture is the *register machine* [14, Chapter 11]. There are several variants of register machine, but all of them present (i) a certain amount of registers, finite or not; (ii) a limited amount of instructions which manipulates the registers' values; (iii) an instruction for manipulation of the flow of the program; (iv) infinity present in some feature of the memory units [21, p. 181]. For the present work, the model of choice is the Shepherdson's one [20] which is formally described by Definitions 6 and 7.

Definition 6 (Register Machine)**.** A register machine $\mathcal{R}$ is a computational device defined as

$$\mathcal{R} = (R, O, P)$$

where:

1. $R = \{R_1, R_2, \ldots, R_n\}$ is a finite set of infinite capacity registers, with $n \in \mathbb{N}$;

2. $O = \{\texttt{INC}, \texttt{DEC}, \texttt{JNZ}, \texttt{HALT}\}$ is the set of operations;

3. $P = (I_1, I_2, \ldots, I_n)$ is the program, with $n \in \mathbb{N}$.

The execution of the program P always start at the first instruction I_1 and procedures sequentially (unless for programmed execution re-route).

Definition 7 (Instructions)**.** Let the content of register R_i be equal to x. Then, it is possible to define the instructions I of the register machine $\mathcal{R}$ as following:

1. $\texttt{INC}(\texttt{R}_\texttt{i}) = x + 1$;

2. $\texttt{DEC}(\texttt{R}_\texttt{i}) = \begin{cases} x-1 & \text{, if } x > 0 \\ 0 & \text{, otherwise} \end{cases}$;

3. $\texttt{JNZ}(\texttt{R}_\texttt{i}, \texttt{I}_\texttt{j})$ change the execution flow of $\mathcal{R}$ setting I_j as the next instruction to be executed in case $x > 0$; otherwise, the execution flow keeps sequential;

4. HALT ends the computation of $\mathcal{R}$.

At last, a word on equivalence of computational devices. According to the Church-Turing thesis, if a given type of computational device has the same power of a Turing machine, it is equivalent to a Turing machine. It is, however, an equivalence between *classes* of devices, not about the behaviour of two particular, given devices. For this kind of equivalence, we have to resort on the *bisimulation* [18, Section 1.4] concept.

Definition 8 (Bisimilarity)**.** A binary relation $\mathcal{R}$ on the states of a labelled transition systems is a *bisimulation* if, whenever $P\mathcal{R}Q$, for all μ:

1. $\forall P' : P \xrightarrow{\mu} P', \exists Q' : Q \xrightarrow{\mu} Q' \wedge P'\mathcal{R}Q'$;

2. the converse, on the transitions emanating from Q, *i.e.* , $\forall Q' : Q \xrightarrow{\mu} Q', \exists P' : P \xrightarrow{\mu} P' \wedge P'\mathcal{R}Q'$.

Bisimilarity, written $\sim$, is the union of all bisimulations; thus $P \sim Q$ holds if there is a bisimulation $\mathcal{R}$ with $P\mathcal{R}Q$ [18, Definitions 1.4.1 and 1.4.2].

In other words, two computational devices are bisimilar if, for all inputs, both systems departs from equivalent states and arrive in other equivalent ones.

4 From Discrete Computational Devices to MP Systems

The title of this section, *"From Discrete Computational Devices to MP Systems"*, is interesting by itself since it indicates the existence of a procedure to translate ubiquitous systems around us (computational devices) in a particular representation of metabolic origin (MP system). However, to deep dive in the subject, it is necessary to start from the discussion from the fundamentals of computer science.

The Church-Turing thesis state the universality of the Turing machine and equivalent models. In other words, any computable function, procedure or algorithm can be computed by a Turing machine. The definition of algorithm, by the way, is actually tied to this concept.

The strong result of the Church-Turing thesis, nonetheless, is unidirectional in the sense that any computable procedure is computed by some model in the highest class of the computational devices, not by *any* model, which implies that there are (classes of) problems that are not solved by some (classes of) computational devices.

In fact, there is a direct map between the classes of algorithms (languages or grammars) and the classes of computational devices that can perform (accept or recognise) them. Hence, it is possible to decide the appropriate model for a particular function as well as the classes of algorithms which a specific device can work to.

Aware of this knowledge, we can make use of them to restate the goals of this section into a work flow of four questions:

(1) What is the class of algorithms that can be implemented in discrete computational devices?

(2) What is the class of algorithms that can be implemented in MP systems?

(3) Is there a map between computational devices and MP systems?

(4) What is the procedure of this mapping?

By discrete computational devices, we assume as any machine that performs an algorithm in discrete time, *i.e.* step-by-step. We are not concerned about the finitude nor the numerical base of its representation. Our concerns are simply on the computability and complexity level or, in the other words, what and how (long) can a procedure be executed.

In a brief way, our definition of discrete computational devices include a big part of the real world gadgets we are surrounded nowadays, including all digital electronics, such as digital computers, smartphones, embedded systems and others. (Analog electrical systems, for instance, do not play a part in the discrete computational devices.) And the relaxation on the boundedness of the representation allows us to include, also, theoretical model such as the Turing machine itself or models equivalent models as the *register machines*. Hence, we can assume the *discrete computational devices are as powerful as the Turing machine.*

On the other hand, we have the MP systems. To define its class, several strategies exists, but some previous studies [2, 12] indicates the best approach may be a top-down (on the hierarchy of classes), equivalence one. The chosen idea is to take a minimalist model belonging to the class under evaluation and try to represent this model as MP one. In case of success, simulations are carried out in order to verify the correctness of the models correspondence. As one may notice, this approach may actually answer the questions (2) to (4).

The register machine defined by Shepherdson's [20] was the model of choice to be translated to MP system since it presents to desirable features: (i) it is a Turing-equivalent model [14, Chapter XXX; 20] and (ii) it has a minimalist representation consistent of four instructions—*increment register of 1*, *decrement register of 1*, *jump to instruction if register is not zero* and *halt*. Also, it has a intrinsic *instruction counter* that keeps the track of the (number of the) instruction to be executed in a sequential way.

The procedure, then, can be described by the following steps:

(S-1) for each instruction I_i in the program P of the register machine, exists a metabolite I_i in the set M of metabolites of the counterpart MP system;

(S-2) for each instruction R_i used in P, exists a metabolite R_i in the set M of metabolites;

(S-3) add a single metabolite H to the set M representing the halting state;

(S-4) for each j^{th}-indexed *jump to instruction if register is not zero* instruction, there is a metabolite L_j in the set M;

(S-5) for each instruction I_i in the program P, added the the rules to the following rules:

(S-5.1) if I_i is a *increment register* R_i *of 1* instruction, then add the rule $\emptyset \rightarrow R_i : I_i$. If there is an instruction I_{i+1}, then the rule $I_i \rightarrow I_{i+1} : I_i$ should also be added;

(S-5.2) if I_i is a *decrement register* R_i *of 1* instruction, then add the rule $R_i \rightarrow \emptyset : I_i$. If there is an instruction I_{i+1}, then the rule $I_i \rightarrow I_{i+1} : I_i$ should also be added;

(S-5.3) if I_i is a *halt* instruction, then add the rule $I_i \rightarrow H : I_i$; otherwise,

(S-5.4) if I_i is a *jump to instruction* I_k *if register* R_i *is not zero* instruction, then add the next four rules:

(i) $I_i \rightarrow L_i : I_i$;

(ii) $L_i \rightarrow I_k : L_i - I_{i+1}$;

(iii) $L_i \rightarrow \emptyset : I_{i+1}$;

(iv) $\emptyset \rightarrow I_{i+1} : I_i - R_i$;

(S-6) for each metabolite R_i, defines its initial value to the same value of its register counterpart (register R_i);

(S-7) the initial value for the instruction $I_1 = 1$ and $I_i = 0, \forall i \neq 1$;

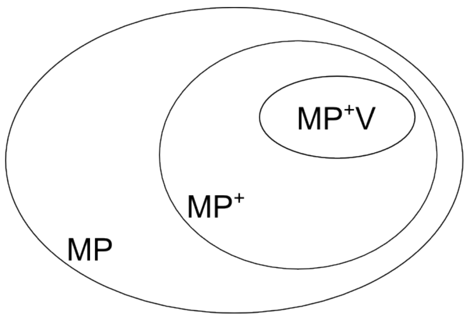
(S-8) the initial value of all $L_i = 0$, as well as for H.

The above algorithmic description is able to convert a register machine not to a conventional MP, but to the particular subset of MP$^+$ (Definition 5). This necessity arises from the fact translations of the `JNZ` instruction, particularly the ones represented in steps (ii) and (iv) of (S-5.4), subtractions that may produce negative differences and put the system in a unknown state. Nonetheless, this restriction does not pose a problem since computational devices are usually defined to operate in the set $\mathbb{N}$ of the natural numbers[3, Chapter 2; 9, Section 4.7; 14, Chapter 11; 20].

An attentive look to fluxes derived of the translation, nonetheless, let us observe that solely *two kinds of functions* are present in the whole grammar: monomial functions and subtraction of monomials, always dependent on the metabolites. This system, which we may call *positively controlled variable gap MP systems* (or MP$^+$V for short), is not only a strict subset of general MP systems, but also of MP$^+$ and, still, present the Turing-completeness property. Therefore, this result allows us immediately to conclude the computational universality of MP$^+$ and general MP systems as well.

The simplicity of the MP$^+$V class may open possibilities for the application of MP systems in a series of fields, including the translation of MP systems to other computational devices such as analog and digital hardware systems.

Figure 1: Venn diagram representing the diverse classes of MP systems.



5 From Metabolical P Systems to Computational Devices

From the results of Section 4, we have established that MP$^+$ (and, by extension, the conventional, general MP) systems are computational devices as powerful as Turing machines and designed a procedure to convert any register machine to this model. Conversely, it is possible to define a general algorithm to convert MP$^+$ systems to register machine.

The discourse of the algorithm passes through the equivalence of any MP system to MP$^+$ and, even more specific, MP$^+$V . However, this subject requires further research development in order to present concrete results.

Meanwhile, there exist two different approaches for the translation of MP systems to computational devices:

1. the (automatic) compilation of MP systems, represented by functions in programming language, to machine code as performed by tools such as MpTheory Java Library[1]; or,

2. the hardware implementation of MP systems, either by components architecture or hardware specification language such as VHDL.

The latter one was worked out in the present research work because of the growing interest to develop bio-inspired hardware [19] as well as to provide cell-on-chip [6] and engineering techniques to all biological research fields, specially the recent synthetic biology [22].

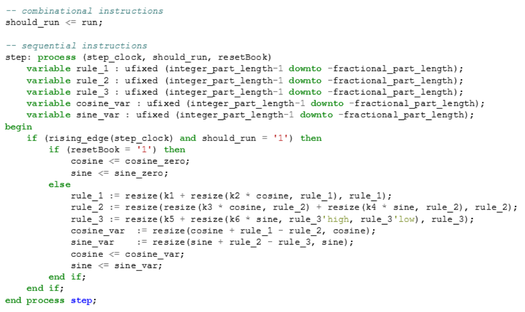
The implementation of the hardware (as can be seen in Figure 2) is fairly simple using the VHDL specification giving the structure of MP grammars: a series of rules, each of them with a well-specified function to perform its variation calculation. However, some mathematical facilities such as the stoichiometric matrix (Definition 3) and the equational metabolic algorithm (Definition 4) had to be substituted for more suitable structures to be implemented in hardware.

The substitution of the aforementioned mathematical led to the discover of a general architecture of MP dynamics that is convenient for hardware implementation in a component-based way[2]. As highlighted in Figure 3 for the

[1]MpTheory Java Library is available at `http://mptheory.scienze.univr.it/`.

[2]The same structure was also found and published, independently, by [13, Figures 7 and 8], but no further studied was performed.

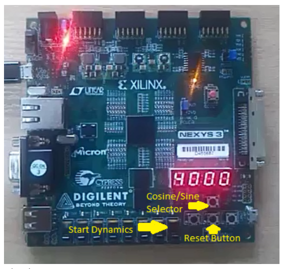
Figure 2: Hardware specification (in VHDL) and implementation for the Goniometricus MP dynamics[§ 3.3.1][10].



```vhdl
-- combinational instructions
should_run <= run;

-- sequential instructions
step: process (step_clock, should_run, resetBook)
    variable rule_1 : ufixed (integer_part_length-1 downto -fractional_part_length);
    variable rule_2 : ufixed (integer_part_length-1 downto -fractional_part_length);
    variable rule_3 : ufixed (integer_part_length-1 downto -fractional_part_length);
    variable cosine_var : ufixed (integer_part_length-1 downto -fractional_part_length);
    variable sine_var : ufixed (integer_part_length-1 downto -fractional_part_length);
begin
    if (rising_edge(step_clock) and should_run = '1') then
        if (resetBook = '1') then
            cosine <= cosine_zero;
            sine <= sine_zero;
        else
            rule_1 := resize(k1 + resize(k2 * cosine, rule_1), rule_1);
            rule_2 := resize(resize(k3 * cosine, rule_2) + resize(k4 * sine, rule_2), rule_2);
            rule_3 := resize(k5 + resize(k6 * sine, rule_3'high, rule_3'low), rule_3);
            cosine_var  := resize(cosine + rule_1 - rule_2, cosine);
            sine_var    := resize(sine + rule_2 - rule_3, sine);
            cosine <= cosine_var;
            sine <= sine_var;
        end if;
    end if;
end process step;
```

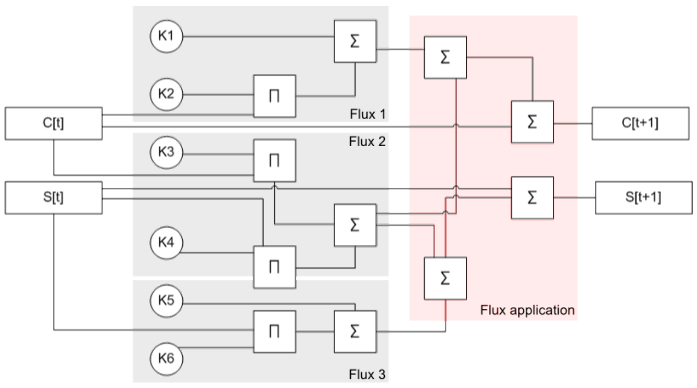
(a) The VHDL specification of the dynamics.



(b) Digilent Nexus V3 board running dynamics.

Goniometricus dynamics, it is a feedback network segmented in two distinct parts: (i) the fluxes block, in the figure highlighted as the gray background, which is specific for each fluxes; and (ii) the fluxes application block, highlighted as the red background, which has a constant topology dependent on the number of variables and presents uniquely summation[3] components.

Figure 3: An arithmetical network of the Goniometricus MP dynamics.



The hardware version, although taking advantage of automatic tools of hardware synthesis, presents great value not only for showing the potential use of MP systems outside biological and mathematical modelling, but also as a bridge connecting the hardware synthesis algorithms for translation (and minimisation) of specifications into components to the future algorithm for translation of MP^{+}V to register machine program.

[3]Actually, subtraction components are also present. However, it is easy to transform a summation in a composition of summation and negation components if the correct numeral representation is chosen.

6 Discrete Fourier Transform as MP System

Fourier transform is an important mathematical tool for analog circuits analysis once it provides an almost mechanical procedure to transform, in both directions, integro-differential equations into an algebraic ones which are much easier to be solved. Its core idea is to change the domain of the problem from the time domain to the frequency domain [8]Section 11.

Nevertheless, Fourier transform operates in the continuous domain of time and its definitions cannot be directly used in the procedures executed in discrete computational devices (for instance, real world computers). Hence, diverse algorithmic approximations were developed (as discrete Fourier transform [8, Section 11.9]) and one the classes has a particular importance for its temporal efficient for computation of the transformer: the so-called *fast Fourier transform*, or FFT for short [8, p. 531].

Both the continuous and the discrete versions of the Fourier transforms rely on the trigonometrical functions of sine and cosine because of the decomposition of the original signal (problem) in harmonic frequencies. However, it is known fact that MP systems are mathematical formalism particularly adapted for generation of time series with oscillatory behaviour, with one of its most trivial examples (the Goniometricus oscillator [10, Section 3.3.1]) to be the generator of the two fundamental trigonometric functions.

More than naturally, therefore, the question if MP systems may compute the discrete Fourier transform of a given signal arises. However, for this attempt, the MP regression [10, Section 3.4] approach was chosen for the algorithmic base because of its simplicity to validate the idea with the available tools, such as MATLAB and MpTheory Java Library.

The strategy to efficient discrete Fourier transform using MP regression relies on the definition of Fourier series, which represents any periodic function in terms of sine and cosine [8, Sections 11.1 and 11.4] in a first-order polynomial [8, Equation (5), p. 476], and the frequencies that can be found in a sampled signal (according to the Nyquist-Shannon sampling theorem [4, Theorem 9.3.1]). The existing procedure can be summarised by the following steps:

1. receive the signal χ and the time T as a vector of data points;
2. calculates the period $\tau = T[i+1] - T[i]$ for $0 \leq i \leq |T| - 2$;
3. calculate all the frequencies $F = \{f : 0 \leq n \times f_{step} \leq f_{Nyquist}\}$ that may exist in the signal, with $f_{step} = \frac{1}{\tau \times |T|}$, $f_{Nyquist} = \frac{1}{2\times\tau}$ and $n \in \mathbb{N}$;
4. generate the sines $S = \{S_f = \sin(2\pi \times f \times t) : f \in F, t \in T\}$ and cosines $C = \{C_f = \cos(2\pi \times f \times t) : f \in F, t \in T\}$;
5. generate a grammar $G = (M, R, I, \Phi)$ given that
   - $M = \{X\}$, representing the signal χ;
   - $R = \{r_1 = \emptyset \to X, r_2 = X \to \emptyset\}$;
   - $I = \chi[0]\}$;
   - $\Phi = \emptyset$;
6. set the regressor dictionary D with monomials representing all the parameter sines and cosines;

7. perform the MP regression on (G, χ, D, S, C, τ).

As the result of the above regression, it will be inferred the signal $\chi = \varphi_1 - \varphi_2$ as a first-order polynomial dependent on the S_f and C_f of frequencies f present in χ, precisely as defined by the Fourier series in [8, Equation (5), p. 476].

As can be seen in Table 1, the proposed technique for discrete Fourier transform using MP regression is more accurate than the state-of-art fast Fourier transform algorithm [5]. Nonetheless, it is important to note the MP version cannot be considered a FFT algorithm because of its obviously bigger computational complexity than $O(N \log_2 N)$ [8, p. 531] because of intensive matrix operations. Optimisations, nevertheless, are planned in order to bring the MP implementation not only competitive in terms of accuracy, but also performance.

Table 1: The retrieved frequencies using the FFT and MP-DFT methods.

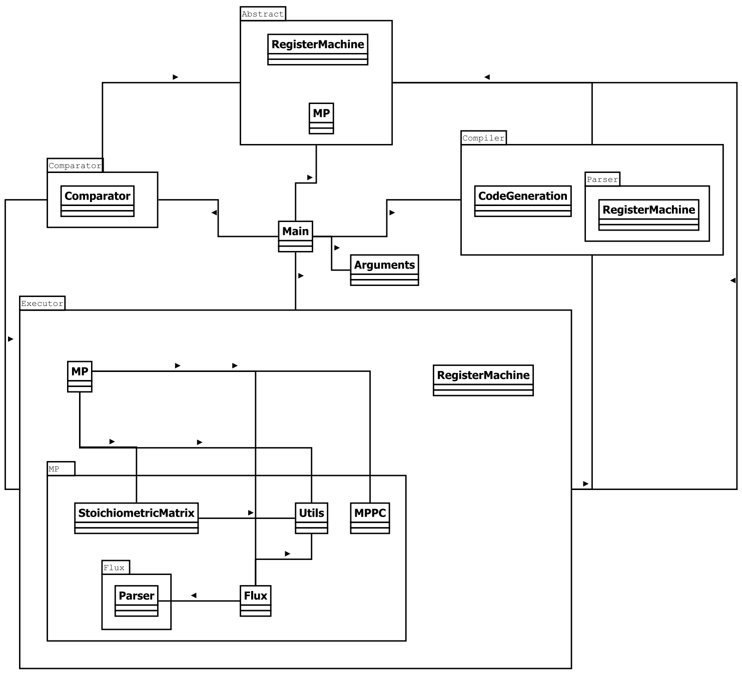
Signals	**Frequency**	**Numerical Frequency**	**FFTW**	**MP-DFT**
1	20	20	20	20
2	$20 + df$	20.5	20.5	20.5
3	$20 + \frac{3}{4} \cdot df$	20.375	20.5	{20 20.5}
4	$\{20, 47\}$	$\{20, 47\}$	{20, 47.5}	{20 47}
5	$\{20 + df, 47 + 3 \cdot df\}$	$\{20.5, 48.5\}$	{20.5, 49}	{20.5 48.5}
6	$\{20 + \frac{3}{4} \cdot df, 47 + \frac{2}{3} \cdot df\}$	$\{20.375, 47.\overline{3}\}$	{20.5, 47.5}	{20.5 47.5}
7	20 + noise	20 + noise	{2, 3, 5, 7, 8.5, 10, 12, 14, 15, 16.5, 17.5, 18.5, 20, 22.5, 23.5, 24.5, 25.5, 26.5, 27.5, 29, 30.5, 32.5, 34, 35, 36.5, 38, 39.5, 41.5, 44, 45, 47, 48}	20
8	$20 + df$ + noise	20.5 + noise	{1, 3.5, 4.5, 6, 7.5, 10, 12, 14, 15.5, 17.5, 19, 20.5, 21.5, 23, 24, 25.5, 27, 28, 30, 32, 33, 34.5, 36, 37.5, 40, 42, 43, 44, 47, 48, 49.5}	20.5
9	$20 + \frac{3}{4} \cdot df$ + noise	20.375 + noise	{1.5, 2.5, 5.5, 6.5, 8, 9.5, 11, 12, 13, 15, 16, 17.5, 20.5, 22.5, 23.5, 25.5, 26.5, 27.5, 28.5, 29.5, 31, 32, 33.5, 34.5, 36, 37, 38, 40, 42, 43, 44.5, 46, 47, 49}	20.5

7 Software Toolkit

In order to complement, verify and stress the theoretical accomplishments of the present research, a functional software toolkit was developed. In the present state, it is a command-line tool specialised in three activities: (i) the translation of algorithms expressed using the Shepherdson's register machine instruction set (Definitions 6 and 7) to a MP$^+$ grammar; (ii) simulator of both register machine and MP$^+$ systems; and (iii) comparator of simulations' output to verification of equivalence of the computational devices (register machine and MP$^+$).

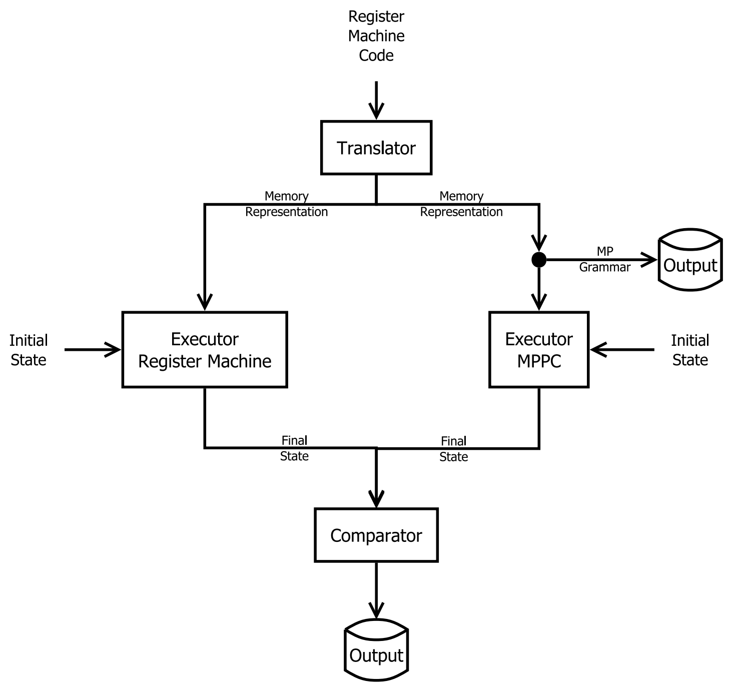
As depicted in Figure 4, the toolkit is modularly constructed so each of its functions can work both in individual or connected (batch) mode and the strong interdependence is solely present in the abstract representation of the computational devices, reflecting the core importance of the theoretical aspects of the work. Additionally, the detachment of the modules permits an internal parallelism of the computation of the different tasks, providing a considerate performance improvement on the computational activities, as highlighted in parallel executors of the Figure 5.

Figure 4: Architecture of the software, highlighting the interconnection among modules or inside it when existing.



For an example of the existing software toolkit, let us see the max function, represented as pseudo-code by Figure 6a and as register machine program by Figure 6b, for the initial vector $(R_1, R_2, R_3, R_4, R_5) = (5, 3, 0, 0, 0)$. With the input as the register machine specification, three possible outcomes is possible: (i) the equivalent MP$^+$ grammar; (ii) the simulation of the register machine program; (iii) the comparison of the simulations of both register machine and equivalent MP$^+$ programs. Figure 7 shows the produced output

Figure 5: Data flow of the software.



(MP^+ grammar) from the execution when solely the program in Figure 6b is provided; Figure 8, by the other hand, shows the output of the computation $(R_1, R_2, R_3, R_4, R_5) = (5, 3, 0, 0, 0) \overset{*}{\vdash} (2, 0, 1, 0, 3)$ for both devices, writing 1 at R_3 to indicate the value of the register $R_1 \geq R_2$ (otherwise, $R_4 = 1$ to indicate $R_1 < R_2$).

It is important to note that, by now, the comparison operation does not fully implement the bisimulation definition 8. Instead, it verifies that both computational devices have properly halted and the equality of values among all the registers and metabolites available.

Figure 6: $\max(R_1, R_2)$ function algorithm.

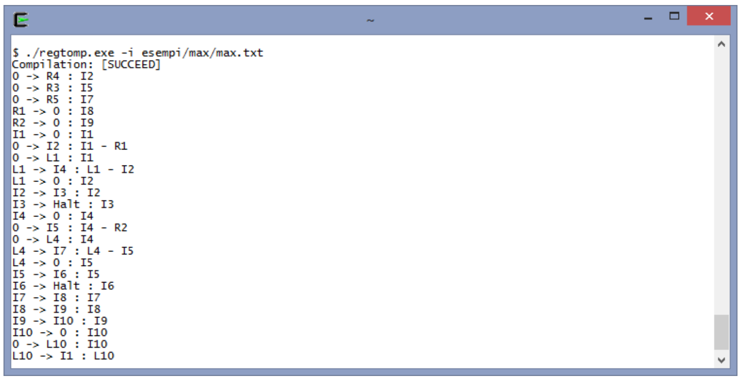
```
function max($R_1, R_2$)
    repeat
        if $R_1 \neq 0$ then
            if $R_2 \neq 0$ then
                $R_5 \leftarrow R_5 + 1$
                $R_1 \leftarrow R_1 - 1$
                $R_2 \leftarrow R_2 - 1$
            else
                $R_3 \leftarrow R_3 + 1$
            end if
        else
            $R_4 \leftarrow R_4 + 1$
        end if
    until $R_3 \neq 0 \wedge R_4 \neq 0$
end function
```

(a) Pseudo-code.

```
01:  JNZ(1, 4)
02:  INC(4)
03:  HALT
04:  JNZ(2, 7)
05:  INC(3)
06:  HALT
07:  INC(5)
08:  DEC(1)
09:  DEC(2)
10:  JNZ(5, 1)
```

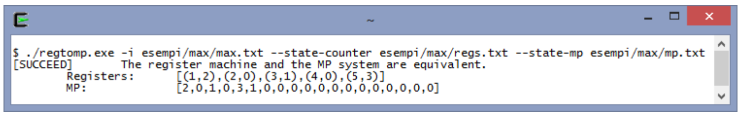
(b) Register machine specification.

Figure 7: Translation from register machine to MP grammar with the command-line tool.



```
$ ./regtomp.exe -i esempi/max/max.txt
Compilation: [SUCCEED]
O -> R4 : I2
O -> R3 : I5
O -> R5 : I7
R1 -> O : I8
R2 -> O : I9
I1 -> O : I1
O -> I2 : I1 - R1
O -> L1 : I1
L1 -> I4 : L1 - I2
L1 -> O : I2
I2 -> I3 : I2
I3 -> Halt : I3
I4 -> O : I4
O -> I5 : I4 - R2
O -> L4 : I4
L4 -> I7 : L4 - I5
L4 -> O : I5
I5 -> I6 : I5
I6 -> Halt : I6
I7 -> I8 : I7
I8 -> I9 : I8
I9 -> I10 : I9
I10 -> O : I10
O -> L10 : I10
L10 -> I1 : L10
```

Figure 8: Simulation and comparison of register machine and equivalent MP grammar with the command-line tool.



```
$ ./regtomp.exe -i esempi/max/max.txt --state-counter esempi/max/regs.txt --state-mp esempi/max/mp.txt
[SUCCEED]       The register machine and the MP system are equivalent.
        Registers:      [(1,2),(2,0),(3,1),(4,0),(5,3)]
        MP:             [2,0,1,0,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

Status Table

Activity	Status
Theory	*In development*
Turing completeness of MP systems	**Finished**
Algorithm for conversion of register machine to MP$^+$V	**Finished**
Algorithm for conversion of MP$^+$V to register machine	*In development*
Bisimilarity of MP$^+$V to register machine	*In development*
Equivalence between MP systems and digital hardware	**Finished**
Hardware	**Finished**
Study of possibility to implement MP systems into analog or digital hardware	**Finished**
Implementation of MP systems in digital hardware	**Finished**
Software	*In development*
Translator of register machine to MP$^+$V	**Finished**
Translator of MP$^+$V to register machine	Not started
Simulator of register machine	**Finished**
Simulator of MP systems	**Finished**
Simulator of MP$^+$	**Finished**
Comparator of systems by final states	**Finished**
Comparator of systems by bisimulation	Not started
Front-end for the software	*In development*
Text	*In development*
Article about computational universality of MP systems	**Finished**
Article about hardware implementation of MP systems	*In development*
Article about software for automatic translation between register machines and MP$^+$V	Not started
PhD Thesis	Not started

References

[1] Béla Bollobás, *Modern Graph Theory*, Vol. 184, Springer, 1998.

[2] Alberto Castellini, Giuditta Franco, and Vincenzo Manca, *Hybrid functional petri nets as MP systems*, Natural Computing **9** (2010), 61–81.

[3] Barry S. Cooper, *Computability Theory*, Chapman and Hall/CRC, 2004.

[4] Joy A. Cover and T. M. Thomas, *Elements of Information Theory*, Wiley, 2006.

[5] Matteo Frigo and Steven G. Johnson, *The design and implementation of FFTW3*, Proceedings of the IEEE **93** (2005), no. 2, 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[6] Lauren Gravitz, *Cell on a Chip*, 2009.

[7] Diederich Hinrichsen and Anthony J. Pritchard, *Mathematical Systems Theory I: Modelling, State Space Analysis, Stability and Robustness*, Texts in Applied Mathematics, vol. 48, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[8] Erwin Kreyszig, *Advanced Engineering Mathematics* (10th, ed.), John Wiley & Sons, 2010.

[9] Harry Lewis and Christos Papadimitriou, *Elements of the Theory of Computation*, 2nd ed., Prentice-Hall, Upper Saddle River, 1997.

[10] Vincenzo Manca, *Infobiotics: Information in Biotic Systems*, Emergence, Complexity and Computation, vol. 3, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[11] Vincenzo Manca, Luca Bianco, and Federico Fontana, *Evolution and Oscillation in P Systems: Applications to Biological Phenomena*, Membrane computing se - 4, 2005, pp. 63–84.

[12] Vincenzo Manca and Luca Marchetti, *Solving dynamical inverse problems by means of Metabolic P systems.*, Bio Systems **109** (July 2012), no. 1, 78–86.

[13] Luca Marchetti and Vincenzo Manca, *Recurrent Solutions to Dynamics Inverse Problems: A Validation of MP Regression*, Journal of Applied & Computational Mathematics **03** (2014), no. 05.

[14] Marvin Minsky, *Computation: Finite and Infinite Machines*, 1st ed., Prentice Hall, 1967.

[15] Gheorghe Păun, *A quick introduction to membrane computing*, Journal of Logic and Algebraic Programming **79** (2010), 291–294.

[16] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, *The Oxford Handbook of Membrane Computing* (Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, eds.), Oxford University Press, 2010.

[17] Wolfgang Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*, Springer, 2013.

[18] Davide Sangiorgi, *Introduction to Bisimulation and Coinduction*, Cambridge University Press, 2012.

[19] Rahul Sarpeshkar, *Ultra-Low Power Bioelectronics*, 1st ed., Cambridge University Press, 2010.

[20] J. C. Shepherdson and H. E. Sturgis, *Computability of Recursive Functions*, Journal of the ACM **10** (1963), 217–255.

[21] Michael Sipser, *Introduction to the Theory of Computation*, 3rd ed., Cengage Learning, Boston, USA, 2012.

[22] Alex Wright, *Molecular moonshots*, Commun. ACM **58** (March 2015), no. 4, 15–17.