Automatic Translation of MP$^+$V Systems to Register Machines

Ricardo Henrique Gracini Guiraldelli and Vincenzo Manca

University of Verona.
Strada Le Grazie, 15, 37134, Verona, Italy.
{ricardo.guiraldelli, vincenzo.manca}@univr.it

Abstract. The present work proposes a translation of MP systems into register machines. The already proved universality of MP grammars [6] results here in minimal terms by means of a suitable subclass of these grammars. This minimality suggests a specification of the metabolic computational paradigm of MP grammars at low (register) level, which is a first step toward a circuit-based implementation of these systems.

1 Introduction

Metabolic P (MP) systems have been evolving since its conception from a modelling language for biological systems using the nature inspired P system [11] to a computational framework for diverse mathematical activities, such as arithmetical operations [10, Section 3.8.1] or regression of temporal series [13]. Consequently, questions concerning the computational power of MP systems have driven the research to some correspondent models [3, 12] without a complete equivalence to Turing machines or more powerful devices [5, 19].

From the synthetic and systems biology perspective, on the other hand, there is a rise on the number of methods seeking to import to those fields already established methodologies in engineering, in a kind of *computer-aided biology*. Component modelling [17], hardware design [7, 16] and compiler techniques [1] are some of them. Although well-intentioned, these suggestions are reinterpretations of engineering practices and fail to communicate in a biological well-understood language.

Trying to unify both standpoints (discrete metabolic computing framework and formal design of systems and synthetic biology), the present work closes the cycle (started with [6]) of MP system as a universal computational model of discrete and deterministic metabolic computing, providing the means to convert a rule based system coded as a particular MP system (Definition 9) into an equivalent register machine description as well as formally describe the core algorithms implemented in our compiler software under development[1]; hence,

[1] The current version of the software (command-line application) may be downloaded at `http://ricardo.guiraldelli.com/resources/software/compiler/regtomp.zip`. In near future, a web version of it will be also available.

a bidirectional bridge between computational and mathematical modelling and chemical and biological worlds is built and, as a consequence, also a base to develop new tools that connect them.

To introduce our mechanism of translation between models, the present paper is divided as follows: Section 2 introduces the concept of *register machine* and specifies the one used as target modelling language in the text; then, there is a review on classical MP systems followed by the presentation of the MP^+ class of systems, our reference model. Section 4 scrutinize the details of the translation between MP^+V systems and register machine. Finally, we make final remarks on the Section 5.

2 Register Machine

The literature enumerates several Turing-powerful models of computation [9, Chapter 4], each one suitable for different context and applications given their own particularities. For the case in which the circumstance requires a great proximity to the bare-metal real computers, the *register machine* is one the most convenient models to be used.

A register machine is defined by a finite set R of registers, a finite set O of operations over the registers and a program P, an indexed sequence of applied operations. Each of the registers $r \in R$ has infinite capacity, storing numbers of arbitrary length and precision (in our case, any number $n \in \mathbb{N}$) [14, Section 11.1]. The set of operations must, at least, provide the features to define and reproduce recursive functions [9, Section 4, p. 196], free access to resources (memory units or program instructions) as the unconstrained head of the Turing machine, a signalization of end of computation and be restricted to the set $\mathbb{N}$. Therefore, the set O can be defined using four operations: *zero*, *successor*, *decrements or jump* and *halt* [14, Table 11.1-1]. Nevertheless, we have chosen to embrace both the standard and the extended[2] Shepherdson's and Sturgis' register machine model [18, Sections 2 and 4].

Definition 1 (Standard Register Machine). *A (standard) register machine $\mathcal{R}$ is a computational device defined as*

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

where:

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

[2] The original model of what we call Shepherdson's and Sturgis' extended register machine model [18, Sections 2] does not contain the `JNZ` instruction: it is introduced later and is shown that both `JMP` and `JZ` can be rewritten in terms of `JNZ` [18, Section 4, p. 225].

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

Definition 2 (Extended Register Machine). *An extended register machine $\mathcal{R}$ is a standard register machine as in Definition 1 with the set of operations O defined as*

$$O = \{\texttt{INC}, \texttt{DEC}, \texttt{CLR}, \texttt{JMP}, \texttt{JZ}, \texttt{JNZ}, \texttt{HALT}\}$$

The concept of *instruction* of a register machine $\mathcal{R}$ (as referenced in Definition 1) is simply a convenient notation to name the operations in O over addressed registers $R_i \in R$ or other instructions; the behaviour of those mentioned so far is described in Definition 3.

Definition 3 (Instructions). *Let the content of register R_i be equal to x. Then, the definition of the instructions in the set I derived of the operations in O for the register machine $\mathcal{R}$ is:*

1. $\texttt{INC}(\texttt{R}_\texttt{i}) \equiv R_i \leftarrow x + 1$
2. $\texttt{DEC}(\texttt{R}_\texttt{i}) \equiv \begin{cases} R_i \leftarrow x - 1 & , \text{ if } x > 0 \\ R_i \leftarrow 0 & , \text{ otherwise} \end{cases}$
3. $\texttt{CLR}(\texttt{R}_\texttt{i}) \equiv \texttt{R}_\texttt{i} \leftarrow \texttt{0}$
4. $\texttt{JMP}(\texttt{I}_\texttt{j})$ *change the execution flow of $\mathcal{R}$ setting I_j as the next instruction to be executed;*
5. $\texttt{JZ}(\texttt{R}_\texttt{i}, \texttt{I}_\texttt{j})$ *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;*
6. $\texttt{JNZ}(\texttt{R}_\texttt{i}, \texttt{I}_\texttt{j})$ *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;*
7. `HALT` *ends the computation of $\mathcal{R}$.*

We also define three subprograms, `CPY`, `ADD` and `SUB`[3], to simplify some of the algorithms included in this text. `CPY` simply copy the contents of the origin register R_1 to the destination one, R_2, overwriting its values; the other two algorithms represent respectively the arithmetical operations of addition and subtraction and allow the destination register (R_3 in the Algorithms 2 and 3) to be one of the terms of the operation (R_1 or R_2), a desired property that also contributes for the conciseness of the codes in the Section 4.

The above definitions of register machine specify a very simple and powerful computation model very close to the real implementation of hardware architecture and can be translated to a series of other models, such as software instructions, digital circuits and metabolic systems [6], serving as an useful intermediate language.

[3] The exponential notation used in the algorithms are repetition of the commands as originally defined by Shepherdson and Sturgis [18, Sections 2].

Algorithm 1 `CPY` subprogram, where $R_2 \leftarrow R_1$

1 $\texttt{CLR}(R_\alpha)$
2 $\texttt{CLR}(R_2)$
3 $\{\texttt{INC}(R_\alpha), \texttt{INC}(R_2)\}^{R_1}$
4 $\{\texttt{INC}(R_1)\}^{R_\alpha}$

Algorithm 2 `ADD` subprogram, where $R_3 \leftarrow R_1 + R_2$

1 $\texttt{CLR}(R_\alpha)$
2 $\texttt{CLR}(R_\beta)$
3 $\texttt{CPY}(R_1, R_\alpha)$
4 $\{\texttt{INC}(R_\alpha), \texttt{INC}(R_\beta)\}^{R_2}$
5 $\{\texttt{INC}(R_2)\}^{R_\beta}$
6 $\texttt{CPY}(R_\alpha, R_3)$

3 Metabolic P Systems

Metabolic P (for short, MP) systems are a particular type of formalism inside the class of membrane computing. Originally, it has been developed inspired by the purposes of P systems but specialized to the modelling of biological dynamics [11, p. 64] (particularly, *metabolic* ones).

Concerning its features, MP systems inherit the basic structures of its superclass [15, Section 2] (membrane structure, multisets of objects and rules), but also possess those of discrete dynamical systems [8, Chapter 2] such as *discrete step of execution*, *parallel execution of all of their rules* and *feedback-like update of their state variables*. The elements and the influence of both perspectives mentioned above can be seen in the formal (and modular) definition of the MP systems [10, Chapter 3] given below.

Definition 4 (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 (variables or metabolites), and* $n \in \mathbb{N}$ *the number of substances.*

Algorithm 3 `SUB` subprogram, where $R_3 \leftarrow R_1 - R_2$

1 $\texttt{CLR}(R_\alpha)$
2 $\texttt{CLR}(R_\beta)$
3 $\texttt{CPY}(R_1, R_\alpha)$
4 $\{\texttt{JZ}(R_\alpha, 5), \texttt{DEC}(R_\alpha), \texttt{INC}(R_\beta)\}^{R_2}$
5 $\{\texttt{INC}(R_2)\}^{R_\beta}$
6 $\texttt{CPY}(R_\alpha, R_3)$

2. $R = \{\alpha_j \to \beta_j : 1 \leq j \leq m\}$ *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 the substances or the metabolic state at initial step (step zero or* t_0*).*
4. $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$ *is a set of functions (also called regulators), in which every* $\varphi_j : \mathbb{R}^n \mapsto \mathbb{R}$*, for* $1 \leq j \leq m$*, is associated with a rule* $r_j \in R$*.*

The above, static definition of MP grammar encompasses all its composing elements as well as all membrane computing features: the membrane structure represented by the grammar G itself, the multisets defined by the metabolites M and their initial states I and the rules through the sets R and Φ. And although the set of fluxes Φ also indicates features of dynamical systems, it is essential to define the recurrent computational process of the state variables (*i.e.*, metabolites quantities) in order to give MP systems a dynamical behaviour.

Definition 5 (Stoichiometric matrix). *Given an MP grammar* $G = (M, R, I, \Phi)$*, let* $r_i \in R$ *be an MP rule of the form* $\alpha_i \to \beta_i$ *as in Definition 4.*

The operator $\text{mult}^+(x_j, r_i)$ *retrieves the multiplicity of the metabolite* x_j *in the right-side of the rule (*i.e.*, in* β_i*). Its counterpart* $\text{mult}^-(x_j, r_i)$ *operates similarly, but over the multiset* α_i*.*

Then, a stoichiometric matrix $\mathbb{A}$ *for the MP grammar* G *has each of its elements defined by*

$$a_{p,q} = \text{mult}^+(x_p, r_q) - \text{mult}^-(x_p, r_q)$$

for $1 \leq p \leq |M|$ *and* $1 \leq q \leq |R|$*.*

Definition 6 (Equational Metabolic Algorithm (EMA)). *At a given time* $t_i \in \mathbb{N}$*, let* $\varphi_j(t_i)$ *be the computed value of the flux* φ_j *at time* t_i *and* $U[t_i] = (\varphi_1(t_i), \varphi_2(t_i), \ldots, \varphi_m(t_i))^\mathsf{T}$ *the vector of all fluxes' values at that step.*

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

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

and the so-called Equational Metabolic Algorithm, which computes the value of any substance in the future time step t_{i+1}*, is computed through the following recurrent equation*

$$X[t_{i+1}] = X[t_i] + \Delta[t_i]$$

Definition 7 (MP system). *A* MP system $\mathcal{M}$ *is a discrete dynamical system defined*[4] *as*

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

with

[4] This definition of MP system is a simplification over the one presented in [10, p. 109]: the concepts of *number* ν *of conventional mole* and *vector* μ *of mole masses* are useful in some circumstances, but not essential—specially in the context of the present work.

1. *G being an MP grammar following the definition 4;*
2. $\tau \in \mathbb{R}$*, the period (amount of time) of a computational step;*

Hence, a static MP grammar becomes an MP (dynamical) system through the existence of a procedure to compute its future states (Definition 6) and an association with a time scale (Definition 7).

3.1 The MP$^+$ Class of Systems

There is a (sub)class of MP systems, introduced in [6], called *positively controlled MP systems* (or, for short, MP$^+$) which restricts the quantities of MP substances to the infimum of zero. With such attribute, this kind of system is suitable for correspondences with biological system, in which quantities are represented in the set $\mathbb{N}$ or $\mathbb{R}^+$, and is enough to present itself as a Turing-powerful model equivalent to register machine [6].

Definition 8 (MP$^+$ Grammar). *A MP$^+$ grammar $G' = (M, R, I', \Phi')$ is a derivation from a (standard) MP grammar $G = (M, R, I, \Phi)$ if its vector of initial values for substances I' has all components greater than zero and G' respects the following restrictions at every computational step t_i:*

1. $\varphi'(t_i) = \begin{cases} \varphi(t_i) & , \text{ if } \varphi(t_i) \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$*, for all* $\varphi' \in \Phi'$ *and their correspondents* $\varphi \in \Phi$*;*
2. $\sum_{\varphi' \in \Phi'^-_x} \varphi'(t_i) \leq x$*, where the set of consuming fluxes of the metabolite x is defined as* $\Phi'^-_x = \{\varphi'_j : \text{mult}^-(x, r_j) > 0, \forall r_j \in R\}$*; otherwise,* $\varphi'(t_i) = 0, \forall \varphi' \in \Phi'^-_x$ *at the execution step t_i.*

From the procedures to transform a register machine (Definition 1) into a MP$^+$ system [6, Section V] arises a pattern which establishes a minimalist and deterministic metabolic P counterpart model of a register machine, the MP$^+$ to variable gap (MP$^+$V).

Definition 9 (MP$^+$V Grammar). *A MP$^+$V grammar $G = (M, R, I, \Phi)$ is a MP$^+$ one in which:*

1. $\forall r \in R$ *and* $v', v'' \in M$*, r must have one of the following shapes:*
   (a) $\emptyset \to v''$*;*
   (b) $v' \to \emptyset$*; or*
   (c) $v' \to v''$*;*
2. $\forall \varphi \in \Phi$ *and* $m', m'' \in M$*, the flux has either the form* $\varphi = m'$ *or* $\varphi = m' - m''$*.*

4 Translation of MP$^+$V Systems into Register Machine Programs

From the equivalence result between MP systems and Turing machines [6], it is easy to realize that any given algorithm $\mathcal{A}$ represented in register machine notation, which we may conveniently call $\mathcal{A}_R$, can also be expressed in MP terms (or, simply, as $\mathcal{A}_M$). From this equivalence also derives the transformation on the other way round, *i.e.* $\mathcal{T}_M : \mathcal{A}_M \mapsto \mathcal{A}_R$, translating any MP algorithm into a register machine one. In fact, $\mathcal{T}_M$ is the present focus of this work.

To simplify its definition, we are going to restrict the input MP system to the MP$^+$V class, without loss of generality (Section 3). This class of systems is chosen because:

1. it is the output from the transformation $\mathcal{T}_R : \mathcal{A}_R \mapsto \mathcal{A}_M$ [6];
2. all of its fluxes $\varphi \in \Phi$ are functions of the type $\mathbb{N} \mapsto \mathbb{N}$, meaning the operations of MP$^+$V systems are performed over the set $\mathbb{N}$ of number as those in the register machine; and,
3. it presents a reduced varieties set of rules with only two types of fluxes: single variable or subtraction of two variables.

Although $\mathcal{T}_M$ may sound a trivial inverse transformation of $\mathcal{T}_R$, its definition poses challenges that require proper treatment in order to provide a correct and total transformation of every $\mathcal{A}_M$ into $\mathcal{A}_R$. Hence, we dedicate the remainder of this section to describe and give mathematical treatment for all of them.

4.1 The Caveats of MP$^+$V

The MP$^+$V systems have two intrinsic properties nonexistent in most of the computational formalism (including the register machines) that require attentive study before any attempt to define a translation procedure between the systems. These properties are the *unordered, parallel application of the rules* and the *positive control* property.

Inheritance of the P systems, the parallel application of the MP$^+$V rules has a meaning in the metabolic systems since it describes different contexts regulated by chemical rules independent among each other, inside the cell fluid and behaving under the Brownian rules; the dependency, when existent, is expressed by the fluxes of the rules.

This parallelism of the rules is converted to sequential steps through the establishment of a *block of execution* (such as those present in program languages) in which all the rules and its fluxes are computed for all the variables over auxiliary values, as if the variable values were frozen while the block is being computed. Figure 1 explains the process.

The positive control, on the other side, is a system's property which requires the variable values to be greater than or equal the sum of all its consuming fluxes; this restriction, thought to not allow negative quantities of metabolites, is not completely satisfied by the subtraction in $\mathbb{N}$ and must be implemented as a special routine when translating MP$^+$V to register machine according to Definition 8.

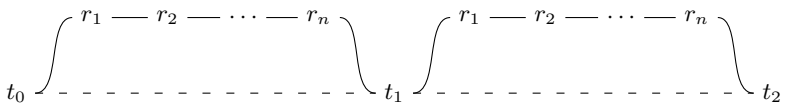

Fig. 1. Graphical representation of the block of execution. The continuous lines represent the actual, sequential execution steps while the dotted ones represent the standard, parallel computation steps of MP systems.

4.2 The MP$^+$V Rules

The MP$^+$V systems, as a result of the $\mathcal{A}_R \mapsto \mathcal{A}_M$ translation procedure [6, Theorem 1], generates simply four kinds of MP rules that add quantity to a variable V_1 (*i.e.*, $\emptyset \to V_1$), remove quantity from it ($V_1 \to \emptyset$), transfer quantity to another variable V_2 (in the form of the rule $V_1 \to V_2$) or, halts the computation ($V_1 \to HALT$), which requires the additional, special purpose metabolite $HALT$ to signalize the end of the procedure[5]. All of them may be combined solely with fluxes controlled by a single referenced variable (*e.g.* V_1) or a subtraction of two variables ($V_1 - V_2$).

The translation procedures of $\mathcal{T}_M$ for each of the above MP rules resemble inverse versions of those in $\mathcal{T}_R$ [6, Section V]. Nonetheless, additional operations are added in order to provide the correct behavior for any inputted MP$^+$V system, not only those outputted from a $\mathcal{T}_R$ transformation.

The strictly increasing MP rule, the one of the form $\emptyset \to V_1 : \varphi$, simply add the value of the $\varphi(t)$ (at time t) to the variable V_1; as a recurrent expression, it may be expressed as $V_1[t+1] = V_1[t] + \varphi(t)$. In the context of the extended register machine, with R_{V_1} as the register address of the variable V_1, R_φ as the one for $\varphi(t)$ and R_{aux} the address for an auxiliary variable, we have the translation of strictly increasing MP rule as

Algorithm 4 Strictly increasing MP rule as Register Machine code.

```
1  ADD(R_{V_1}, R_φ, R_aux)
2  CPY(R_aux, R_{V_1})
```

Similarly, the strictly decreasing MP rule $V_1 \to \emptyset : \varphi$ is mathematically represented as $V_1[t+1] = V_1[t] - \varphi(t)$ and produces a translation code of the form

The transfer MP rule, $V_1 \to V_2 : \varphi$, can be seen as a composition of both previous rules according to I/O MP systems [10, Chapter 3]; hence, it is not a single recurrent equation, but two of them (Equation 2). Consequently, as can be seen in Algorithm 6, the register machine subprogram for its transformation is simply a concatenation of the previous Algorithms 4 and 5.

[5] In order to differentiate the two "halts" in this paper, `HALT`will represent the halting instruction in register machines while $HALT$ the metabolite in MP$^+$V systems.

Algorithm 5 Strictly decreasing MP rule as Register Machine code.

```
1  SUB(R_{V_1}, R_φ, R_aux)
2  CPY(R_aux, R_{V_1})
```

$$V_1[t+1] = V_1[t] - \varphi(t) \tag{1}$$
$$V_2[t+1] = V_2[t] + \varphi(t) \tag{2}$$

Algorithm 6 Transfer MP rule as Register Machine code.

```
1  SUB(R_{V_1}, R_φ, R_aux)
2  CPY(R_aux, R_{V_1})
3  ADD(R_{V_2}, R_φ, R_aux)
4  CPY(R_aux, R_{V_2})
```

Finally, the halting rule $V_1 \rightarrow HALT : \varphi$ simply signalizes the end of the computation when the quantity of the $HALT$ variable is greater than zero. Hence, the produced code verifies if there is a halting situation (and halts if there is), otherwise updates the quantity of V_1 and halts the execution.

Algorithm 7 Halting MP rule as Register Machine code.

```
1  JNZ(R_HALT, 3)
2  JMP(4)
3  HALT
4  SUB(R_{V_1}, R_φ, R_aux)
5  CPY(R_aux, R_{V_1})
6  ADD(R_HALT, R_φ, R_aux)
7  CPY(R_aux, R_HALT)
```

As already discussed and graphically shown in Figure 1, one computational step of MP$^+$V systems is expanded in several register machine instructions which also include a framework of operations reflecting properties of MP$^+$V nonexistent in the addressed architecture. These additional operations are, now, studied in details.

4.3 Surroundings of MP$^+$V Steps

Translating the rules of MP$^+$V systems to register machine subprograms is just one of the parts to correct transform one system into another. As can be seen in Figure 2 (and previously discussed in Section 4.1), it is necessary to define (i) a

sequential, *computation block* referent to the execution of a step in the MP$^+$V dynamics; (ii) guards for the variables to isolate the computation of the future value (*i.e.*, $V[t+1]$) with the actual value $V[t]$; (iii) evaluation of the *positive control* property; and, finally, (iv) the recurrent call of the dynamical system.

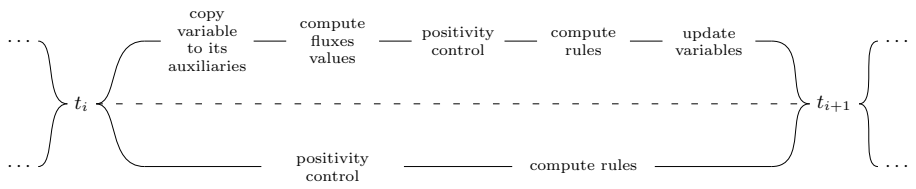

Fig. 2. Representation of a computation step MP$^+$V systems (lower part) and its equivalent register machine (upper part).

The sequential block for computation is nothing more than the simply concatenation of the subprograms (ii) and (iii); it is the upper arc in Figure 2 which, in a sequential way, process all the requirement procedures for the computation of a single MP$^+$V step as a register machine.

The guards, by the other hand, are specialized pieces of code that keep the variables unchanged during the sequential execution of the MP$^+$V step; as can be seen in Figure 2, they represent all the surplus of operations nonexistent in the MP$^+$V track: copy of the values of variables into auxiliary registers, computation of fluxes with assignment to particular registers and update of the variables value after step computation.

There is the necessity to freeze the values of the variables and rely on auxiliary register copies of them because rules and fluxes may reference the quantity of the variable $V[t_i]$ at time t_i, not an intermediary and already changed value $V[t_i + \epsilon]$ in a time $t_i + \epsilon$, with $0 < \epsilon < 1$, solely existing in the sequential (or register machine) context: changes on variables quantities must occur uniquely in the end of the computational step, in accordance to the behaviour of dynamical systems.

A similar pattern arises for the fluxes values. Fluxes must be updated at every MP$^+$V step and their values must be promptly available for the re-computation of the variable quantities according to the application of the rules. Moreover, fluxes values are subject to the positivity control (Definition 8) which, in case of nonsatisfiability of the property, sets the appropriate fluxes to zero; hence, instead of performing computations of fluxes and their verification at each rule application, it is enough to consult their values in their respective dedicated registers.

All these guards demand simplistic algorithms: the copy and update of variables rely on the `CPY` instruction, while the fluxes, depending of their nature, either on the `CPY` or the subtraction subprogram `SUB`.

Algorithm 8 Copy variable to its auxiliary register.

1 $\texttt{CPY}(R_V, R_{V_{aux}})$

Algorithm 9 Update variable with its auxiliary register value.

1 $\texttt{CPY}(R_{V_{aux}}, R_V)$

The translation of the *positivity control* property to the register machine specification, though, is slightly more intricate: while in the MP$^+$V systems it is enough to state as a system's property, in the register machine it must be ensured by actually coding both constraints stated in Definition 8.

The first of them is a statement, in mathematical terms, that any flux φ is a function with co-domain equals to the natural set of numbers $\mathbb{N}$, an easily satisfied requirement since the values of the registers (and, hence, variables) are restricted to $\mathbb{N}$ by the register machine definition (Definition 1) and the fluxes are restricted to monomials or subtraction of variables. As we have already seen, the SUB subprogram has mechanism to guarantee no value goes below zero (Section 2).

Conversely, the other constraint requires a special subprogram to satisfy its conditions. It sets to zero all consuming fluxes for a certain variable V if the sum of them are greater than the actual available quantity of V. In mathematical terms, if Φ_V^- is the set of consuming fluxes of V, $\sum_{\varphi \in \Phi_V^-} \varphi(t_i) \leq V_{t_i}$ at time t_i; otherwise $\forall \varphi \in \Phi_V^-, \varphi(t_i) = 0$. Since the compiler knows both the variable V and the consuming fluxes $\Phi_V^- = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ (with $k = \left|\Phi_V^-\right|$), the generated subprogram does not have to seek for this information and, for each variable, can have the form of Algorithm 11.

The explicit references to the fluxes in Algorithm 11 in both computation of the sum (lines 2 to $k+1$) and the invalidation of the fluxes (lines $k+8$ to $2 \cdot k + 7$) makes positivity control property the biggest contributor for the line of codes in the equivalent register machine version of a MP$^+$V system.

Finally, we must ensure the recurrent computation of the MP$^+$V in the $\mathcal{T}_M$ transformation. From examples such as Goniometricus or Sirius [10, Chapter 3], we know that some MP systems work as infinite series or signal generator, indefinitely computing values without an explicit procedure for stopping them; in contrast, functions such as $\max(R_1, R_2)$ implemented in [6, Figure 4] converges

Algorithm 10 Update the flux value.

if $\varphi = V$ **then**
  1 $\texttt{CPY}(R_V, R_\varphi)$
else ▷ Hence, $\varphi = V_1 - V_2$
  1 $\texttt{SUB}(R_{V_1}, R_{V_2}, R_\varphi)$
end if

Algorithm 11 Positivity control algorithm.

1	$\texttt{CLR}(R_{sum})$
2	$\texttt{ADD}(R_{\varphi_1}, R_{sum}, R_{sum})$
3	$\texttt{ADD}(R_{\varphi_1}, R_{sum}, R_{sum})$
4	$\texttt{ADD}(R_{\varphi_2}, R_{sum}, R_{sum})$
	$\vdots$
k+1	$\texttt{ADD}(R_{\varphi_k}, R_{sum}, R_{sum})$
k+2	$\texttt{CPY}(R_V, R_{comparator})$
k+3	$\texttt{JZ}(R_{sum}, 2 \cdot k + 8)$
k+4	$\texttt{JZ}(R_{comparator}, k + 8)$
k+5	$\texttt{DEC}(R_{sum})$
k+6	$\texttt{DEC}(R_{comparator})$
k+7	$\texttt{JMP}(k + 3)$
k+8	$\texttt{CLR}(R_{\varphi_1})$
k+9	$\texttt{CLR}(R_{\varphi_2})$
	$\vdots$
2·k+7	$\texttt{CLR}(R_{\varphi_k})$

to a particular fixed-point (halt) state which signalizes the end of the calculation process of the system [6, Section V-2-c]. The differentiation between them, nonetheless, relies solely in the existence of a variable $HALT$ in the latter system, as well as a strictly increasing or transfer rule which increments the value of $HALT$. Hence, in terms of register machine, these systems diverge in the existence of a R_{HALT} register and an instruction to stop the execution when $R_{HALT} \neq 0$.

As can be seen in Figure 2, all the register machine code generated by the transformation $\mathcal{T}_M : \mathcal{A}_M \mapsto \mathcal{A}_R$ is enclosed inside two sequential steps t_i and t_{i+1}, except by the recurrent call of the dynamical system. In fact, all the computation performed by the equivalent register machine (*i.e.*, all the algorithms seen so far) is, actually, the computation of a single MP$^+$V step. The objective of the recurrent call is, then, to recall the computation procedures up to the moment $R_{HALT} \neq 0$; to achieve it, the procedure verifies the state of the R_{HALT} before step reckoning and chooses if the program should be halt (a jump to the last line of the program, ℓ, where a `HALT` is always present) or continue with normal execution; in case of the latter, the penultimate line of the program $(\ell - 1)$ redirects the execution back to the first line. In algorithm terms:

Algorithm 12 Loop control of the dynamical systems that halts.

1	$\texttt{JNZ}(R_{HALT}, \ell)$
	$\vdots$
ℓ-1	$\texttt{JMP}(1)$
ℓ	`HALT`

The pseudo-code produced by the transformation is finally represented in Algorithm 13.

Algorithm 13 Complete translation procedure from MP$^+$V system to register machine

while $R_{HALT} = 0$ **do**
  for all variable $v \in M$ **do** ▷ copy variables to auxiliaries
    $R_{v'} \leftarrow R_v$
  end for
  for all flux $\varphi \in \Phi$ **do** ▷ compute fluxes
    $R_\varphi \leftarrow \varphi(t_i)$
  end for
  for all variable $v \in M$ **do** ▷ positivity control property
    for all flux $\varphi_v^- \in \Phi_v^-$ **do**
      $R_{sum} \leftarrow R_{sum} + R_{\varphi_v^-}$
    end for
    if $R_{sum} > v$ **then**
      for all flux $\varphi_v^- \in \Phi_v^-$ **do**
        $R_{\varphi_v^-} \leftarrow 0$
      end for
    end if
  end for
  for all rule r **do** ▷ compute rules
    if r is of the form $\emptyset \rightarrow v : \varphi$ **then**
      $R_{v'} \leftarrow R_{v'} + \varphi$
    else if r is of the form $v \rightarrow \emptyset : \varphi$ **then**
      $R_{v'} \leftarrow R_{v'} - \varphi$
    else ▷ hence, it must be of the form $v_1 \rightarrow v_2 : \varphi$
      $R_{v'_1} \leftarrow R_{v'_1} + \varphi$
      $R_{v'_2} \leftarrow R_{v'_2} - \varphi$
    end if
  end for
  for all variable $v \in M$ **do** ▷ update variables
    $R_v \leftarrow R_{v'}$
  end for
end while

5 Conclusion

There were no doubts of the possibility to translate MP systems to register machine or any other Turing-powerful formalism: the discrete and deterministic characteristics of these metabolic systems are exactly those that guarantee the feasibility of this translation. The novelty, on the other hand, relies in the direct transformation of modelling languages with completely different paradigms: from the metabolic, parallel and centered on pair of substances transformations to a

computational and holistically oriented sequence of instructions standpoint; from MP systems to von Neumann architecture.

It is worth to note that it is not the target language (here the register machine, but equivalently for hardware description or programming ones), but the idea of an algorithmic transformation between models that permits the effortless and automatic translation of metabolic systems into either pieces of software (*e.g.*, for simulation purposes), hardware (such as [4, 7, 16]), visual representations (automata) or any other common use case for exogenous model transformation [2, Chapter 8].

Finally, the availability of the bidirectional translation and equivalence between MP^+V systems and register machines open the way for implementation of circuits based on metabolic (MP) systems.

Bibliography

[1] Jacob Beal, Ting Lu, and Ron Weiss, *Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks.*, PloS one **6** (January 2011), no. 8, e22490.

[2] Marco Brambilla, Jordi Cabot, and Manuel Wimmer, *Model-driven software engineering in practice*, Synthesis Lectures on Software Engineering **1** (2012Sep), no. 1, 1–182.

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

[4] Luis Fernandez, Víctor J. Martinez, Fernando Arroyo, and Luis F. Mingo, *A hardware circuit for selecting active rules in transition P systems*, Seventh international symposium on symbolic and numeric algorithms for scientific computing (synasc'05), 2005, pp. 4 pp.

[5] Marian Gheorghe and Mike Stannett, *Membrane system models for super-Turing paradigms*, Natural computing, August 2012, pp. 253–259.

[6] Ricardo Henrique Gracini Guiraldelli and Vincenzo Manca, *The Computational Universality of Metabolic Computing*, 2015. Available as pre-print at `http://arxiv.org/abs/1505.02420`.

[7] Lauren Gravitz, *Cell on a Chip*, MIT Technology Review, 2009. Available online at `http://www.technologyreview.com/news/414622/cell-on-a-chip/`.

[8] 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.

[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, 2005, pp. 63–84 (English).

[12] Vincenzo Manca and Rosario Lombardo, *Computing with multi-membranes*, Membrane computing, 2012, pp. 282–299 (English).

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

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

[17] ———, *Analog synthetic biology.*, Philosophical transactions. Series A, Mathematical, physical, and engineering sciences **372** (March 2014), no. 2012, 20130110.

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

[19] Hava T. Siegelmann and Shmuel Fishman, *Analog computation with dynamical systems*, Physica D: Nonlinear Phenomena **October** (1998), 1–38.