“Theory and Applications of Computationally Universal Metabolic P Systems”

3rd-year presentation

Ricardo Henrique Gracini Guiraldelli

University of Verona

2015-11-11

Table of Contents

Section 1

Introduction

Once upon a time. . .

Electrical Circuits $\leftrightarrows$ Metabolism

or *find a bidirectional transformation between electrical circuits and metabolism*.

Once upon a time. . .

Electrical Circuits $\leftrightarrows$ Metabolism

Where does the inspiration come from?

- Terje Lomø's long term potentiation [5];
- Kidney loops and mechanical engineering [4, p. 75];
- Miguel Nicolelis' experiment with monkeys and virtual arms.

Once upon a time...

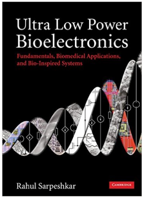
Electrical Circuits $\leftrightarrows$ Metabolism

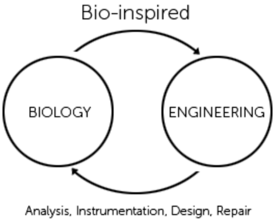
Is it a sound?

- Both are dynamical systems;
- Several living-beings components are modeled after engineering concepts:
  - Circulatory systems $\Leftrightarrow$ fluid mechanics;
  - Skeleton $\Leftrightarrow$ solid mechanics;
  - Muscular moviment $\Leftrightarrow$ electricity;
  - ...
- Correlated to:
  - Biomedical engineering;
  - Systems biology;
  - *Synthetic biology*
- A just-born research field.

Once upon a time. . .

Electrical Circuits $\leftrightarrows$ Metabolism





My intention is. . .

Electrical Circuits $\leftarrow$ Metabolism

- Get a specification of a metabolism;
- Transform it in a specification of an electrical circuit;
- Automatically generate an electrical circuit;
- Reproduce the metabolic behavior in electrical circuit;
- *Tune the behavior in the generated electrical circuit.*[1]

Systems Biology.

[1] Bonus feature.

My intention is. . .

Electrical Circuits → Metabolism

- Get a specification of an electrical circuit;
- Transform it in a specification of a metabolism;
- Automatically generate a metabolism;
- Reproduce the electrical circuit behavior in metabolism;

Synthetic Biology.

Work Breakdown Structure of the PhD research

Electrical Circuits $\leftrightarrows$ Metabolism

- Get a specification of a _________;
- Transform it in a specification of a _________;
- Automatically generate a _________;
- Reproduce the _________ behavior in _________;
- *Tune the behavior in the generated _________.*

Work Breakdown Structure of the PhD research

Electrical Circuits $\leftrightarrows$ Metabolism

- Get a specification of a __________;
  - Metabolism: Metabolic P system.
  - Electrical Circuits:
    - Digital Circuits;
    - Analog Circuits;
    - Algorithms.
- Transform it in a specification of a __________;
- Automatically generate a __________;
- Reproduce the __________ behavior in __________;
- *Tune the behavior in the generated __________.*

Work Breakdown Structure of the PhD research

Electrical Circuits $\leftrightarrows$ Metabolism

- Get a specification of a __________;
  - Metabolism: Metabolic P system.
  - Electrical Circuits:
    - Digital Circuits;
    - Analog Circuits;
    - Algorithms.
- Transform it in a specification of a __________;
  - Theoretical (core) work of the PhD thesis.
- Automatically generate a __________;
- Reproduce the __________ behavior in __________;
- *Tune the behavior in the generated __________.*

Work Breakdown Structure of the PhD research

Electrical Circuits $\leftrightarrows$ Metabolism

- Get a specification of a _________;
  - Metabolism: Metabolic P system.
  - Electrical Circuits:
    - Digital Circuits;
    - Analog Circuits;
    - Algorithms.
- Transform it in a specification of a _________;
  - Theoretical (core) work of the PhD thesis.
- Automatically generate a _________;
  - Practical application of the PhD research.
- Reproduce the _________ behavior in _________;
- *Tune the behavior in the generated _________.*

Work Breakdown Structure of the PhD research

Electrical Circuits $\leftrightarrows$ Metabolism

- Get a specification of a _________;
  - Metabolism: Metabolic P system.
  - Electrical Circuits:
    - Digital Circuits;
    - Analog Circuits;
    - Algorithms.
- Transform it in a specification of a _________;
  - Theoretical (core) work of the PhD thesis.
- Automatically generate a _________;
  - Practical application of the PhD research.
- Reproduce the _________ behavior in _________;
  - Validation of the PhD work.
- *Tune the behavior in the generated _________.*

Electrical Circuits $\leftrightarrows$ Metabolism

- Get a specification of a __________;
  - Metabolism: Metabolic P system.
  - Electrical Circuits:
    - Digital Circuits;
    - Analog Circuits;
    - Algorithms.
- Transform it in a specification of a __________;
  - Theoretical (core) work of the PhD thesis.
- Automatically generate a __________;
  - Practical application of the PhD research.
- Reproduce the __________ behavior in __________;
  - Validation of the PhD work.
- *Tune the behavior in the generated __________.*
  - Users's application.

Work Breakdown Structure of the PhD research

Theory

1. How can I represent metabolism?
2. How can I represent circuit?
3. Can I map every metabolism to circuit?
4. Can I map every circuit to metabolism?
5. What is the map procedure? (Both.)
6. Do I have restrictions?
7. Is the mapping optimal? (In which sense?)

Practice

1. Instance of a metabolism as an electrical circuit.
2. Instance of an electrical circuit as a metabolism.
3. Automatic mapping of metabolism to electrical circuit.
4. Automatic mapping of electrical circuit to metabolism.

Section 2

Basic Knowledge

Basic Knowledge

- To understand the work, it is required to have in mind two concepts:
  1. Metabolic P systems;
  2. Computationally Universal Devices.
- The rest of the work is self-contained.

Metabolic P systems

Static

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

- set of substances M;
- set of rules R;
- initial state I;
- set of fluxes Φ.

Dynamic

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

- Metabolic P grammar G;
- Period of the dynamics, τ;
- Number of conventional mole μ;
- Vector of mole masses ν;
- Update recurrent equation (*Equational Metabolic Algorithm*).

Metabolic P systems

Static

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

- set of substances M;
- set of rules R;
- initial state I;
- set of fluxes Φ.

Dynamic

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

- Metabolic P grammar G;
- Period of the dynamics, τ;
- Update recurrent equation (*Equational Metabolic Algorithm*).

Metabolic P systems

- Subset of P systems (membrane computing);
- Discrete dynamical system;
- Deterministic computation;
- 👍Very mature as numerical algorithm;
- 👎 *Few theoretical computer science results*.
  - Necessary for PhD hypothesis.

Metabolic P systems

- Subset of P systems (membrane computing);
- Discrete dynamical system;
- Deterministic computation;
- 👍Very mature as numerical algorithm;
- 👎 *Few theoretical computer science results*.
  - Necessary for PhD hypothesis.

Computationally Universal Devices

- Computationally universal devices $\Leftrightarrow$ Turing-complete
- Recognizes the highest level of the Chomsky-Schützenberger hierarchy

Grammar	**Language**	**Automaton**
Type-0	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine
Type-2	Context-free	Non-deterministic pushdown automaton
Type-3	Regular	Finite state automaton

- There are several computationally universal models. Register machine was picked.
    - Simple;
    - Easy to reason about;
    - von Neumann architecture-like;
    - Low-level programming.

Register Machine

$$\mathcal{R} = (R, O, P) \text{ [9]}$$

- R is the finite set of *registers* (with infinite capacity)
- $O = \{\texttt{INC}, \texttt{DEC}, \texttt{JNZ}, \texttt{HALT}\}$ is the finite set of operations;
- $P = (I_1, I_2, \ldots, I_n)$ is the (finite) program.
  - Instructions are "applied operations" to registers, instruction-pointer, both or none (`HALT`);
  - Restricted to the set of natural numbers.

Register Machine

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

- R is the finite set of *registers* (with infinite capacity)
- $O' = \overbrace{\{\mathtt{INC}, \mathtt{DEC}, \mathtt{CLR}, \mathtt{JMP}, \mathtt{JZ}, \mathtt{JNZ}, \mathtt{HALT}\}}^{\text{Instructions}} \cup \overbrace{\{\mathtt{CPY}, \mathtt{ADD}, \mathtt{SUB}\}}^{\text{Subprograms}}$ is the extended, finite set of operations and subprograms;
- $P = (I_1, I_2, \ldots, I_n)$ is the (finite) program.
  - Instructions are "applied operations" to registers, instruction-pointer, both or none (HALT);
  - Restricted to the set of natural numbers.

Section 3

Theory

Recalling the Guiding Questions

1. **Q:** How can I represent metabolism?
   A: *Metabolic P systems.*
2. **Q:** How can I represent circuit?
   A: *~~Analog~~, digital circuits or algorithms.*
3. **Q:** Can I map every metabolism to circuit?
4. **Q:** Can I map every circuit to metabolism?
5. **Q:** What is the map procedure? (Both.)
6. **Q:** Do I have restrictions?
7. **Q:** Is the mapping optimal? (In which sense?)

Recalling the Guiding Questions

1. **Q:** How can I represent metabolism?
   A: *Metabolic P systems.*
2. **Q:** How can I represent circuit?
   A: *Algorithms.*
3. **Q:** Can I map every ~~metabolism~~ MP system to ~~circuit~~ algorithm?
4. **Q:** Can I map every ~~circuit~~ algorithm to ~~metabolism~~ MP system?
5. **Q:** What is the map procedure? (Both.)
6. **Q:** Do I have restrictions?
7. **Q:** Is the mapping optimal? (In which sense?)

Subsection 1

Algorithms $\mapsto$ Metabolic P systems

Algorithms $\mapsto$ Metabolic P systems

Q: Can I map every algorithm to MP system?

Algorithm

- Representation of register machine;
- Recursively enumerable language;
- Sequential execution;
- Self-reference at run time (*e.g.* , `JNZ`);
- Operations $\mathbb{N} \mapsto \mathbb{N}$;
- Finite-set of operations.

Metabolic P system

- Dynamical system;
- Could be context-sensitive language [1, 8]. More ambitious attempts [7] has failed.
- Parallel execution;
- Reference to previous-state only;
- Operations $\mathbb{R} \mapsto \mathbb{R}$;
- No restriction to usage of functions.

The Easy Part

Register machine	**Metabolic P grammar**
$\mathcal{R} = (R, O', P)$	$G = (M, R, I, \Phi)$
Set of registers R	Set of metabolites M
Program (sequence) P	Set of rules R Set of fluxes Φ
Initial state of the registers	Initial state I

Restricting the Operations

- Restrict MP systems to $\mathbb{N}$;
- Create a new class of MP systems that manage it correctly: fluxes and rule-application.

Definition (MP$^+$ Grammar)

An 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 or equal to zero, the set of consuming fluxes of the metabolite x defined as $\Phi'^{-}_{x} = \{\varphi'_j : \mathrm{mult}^{-}(x, r_j) > 0\,,\ \forall r_j \in R\}$, and G' respects the following restrictions at every computational step t_i:

1. $\forall \varphi \in \Phi : \varphi'(t_i) = \begin{cases} \varphi(t_i) & , \text{ if } \varphi(t_i) \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$;
2. $\sum_{\varphi' \in \Phi'^{-}_{x}} \varphi'(t_i) \leq x(t_i)$; otherwise $\varphi'(t_i) = 0, \forall \varphi' \in \Phi'^{-}_{x}$ at the execution step t_i.

Hidden Members

- Sequential execution and self-reference:
  - For each instruction I_j of program P, there will be a respective metabolite I_j representing the *instruction pointer*, active or not, at that instruction;
  - For each instruction I_j of the type `JZ` or `JNZ`, there will be a respective metabolite L_j;
  - To signalize the halt of operation of the device, there will be a special (fixed-point) metabolite $HALT$.

Mapping Rules and Fluxes

Register machine	Metabolic P grammar
if I_j is $\mathtt{INC}(R_i)$	$I_j \to I_{j+1} : I_j$ $\emptyset \to R_i : I_j$
if I_j is $\mathtt{DEC}(R_i)$	$I_j \to I_{j+1} : I_j$ $R_i \to \emptyset : I_j$
if I_j is $\mathtt{JNZ}(R_i, I_k)$	$I_j \to L_j : I_j$ $L_j \to I_k : L_j - I_{j+1}$ $L_j \to \emptyset : I_{j+1}$ $\emptyset \to I_{j+1} : I_j - R_i$
if I_j is $\mathtt{HALT}$	$I_j \to HALT : I_j$

- At begining, $I_1 = 1$;
- $HALT + \sum_{j=1}^{p} I_j = 1$
- $0 \leq \sum_{L_j \in M} L_j \leq 1$

Mapping Rules and Fluxes

Register machine	Metabolic P grammar
if I_j is INC(R_i)	$I_j \to I_{j+1} : I_j$ $\emptyset \to R_i : I_j$
if I_j is DEC(R_i)	$I_j \to I_{j+1} : I_j$ $R_i \to \emptyset : I_j$
if I_j is JNZ(R_i, I_k)	$I_j \to L_j : I_j$ $L_j \to I_k : L_j - I_{j+1}$ $L_j \to \emptyset : I_{j+1}$ $\emptyset \to I_{j+1} : I_j - R_i$
if I_j is HALT	$I_j \to HALT : I_j$

- At begining, $I_1 = 1$;
- $HALT + \sum_{j=1}^{p} I_j = 1$
- $0 \leq \sum_{L_j \in M} L_j \leq 1$

Finally, the Theorem

Theorem (Translation of Register Machine to MP^{+})

For any register machine $\mathcal{R}$ exists an equivalent positively controlled MP grammar $\mathcal{G}_{+}$.

The Proof I

Given a register machine $\mathcal{R} = (R, I, P)$ with $|R| = r$ and $|P| = p$, a positively controlled MP grammar $\mathcal{G}_+ = (M, Ru, I, \Phi)$ is constructed

1. adding a metabolite R_i in the set M for each register $R_i \in R$;
2. adding a metabolite I_j in the set M for each of the instructions in $I_j \in P$;
3. adding a metabolite L_j in the set M for each instruction $I_j \in P$ of the type `JNZ`;
4. adding a $HALT$ metabolite in the set M;
5. defining the initial state of the metabolites R_j equal to the initial values of the registers R_j, the initial values of all the other metabolites to 0 and the initial value of I_1 to 1;
6. adding the rules to Ru and the fluxes to Φ according to the following rules:
    1. if I_j is `INC` or `DEC`, then $I_j \rightarrow I_{j+1} : I_j$;
    2. if I_j is `INC(Ri)`, then $\emptyset \rightarrow R_i : I_j$;
    3. if I_j is `DEC(Ri)`, then $R_i \rightarrow \emptyset : I_j$;
    4. if I_j is `HALT`, then $I_j \rightarrow HALT : I_j$;
    5. if I_j is `JNZ(Ri, Ik)`, then
        1. $I_j \rightarrow L_j : I_j$;
        2. $L_j \rightarrow I_k : L_j - I_{j+1}$;
        3. $L_j \rightarrow \emptyset : I_{j+1}$; and,
        4. $\emptyset \rightarrow I_{j+1} : I_j - R_i$.

The Proof II

From the rules above, it is possible to notice that I_j and L_j instructions controls the execution flow of the system and satisfies

$$HALT + \sum_{j=1}^{p} I_j = 1$$

$$0 \leq \sum_{L_j \in M} L_j \leq 1$$

ensuring no two instructions are executed at the same time, but its execution starts from instruction I_1 and proceeds sequentially (or jumps to another one in case of a satisfying `JNZ` instruction).
All operations are mappings from and to the $\mathbb{N}$ set once both $\mathcal{R}$ and $\mathcal{G}_+$, by definition, restrict their operations to this set.

At last, when a rule $I_j \rightarrow HALT$ is performed, the system is stuck in a fixed point since there is no rules for

"exiting" this state.

The Side-Effect Prize

- Result of the transformation is a **very simple** MP system—MP^+V ;
- Not only *simple*, but equivalent to register machine $\Rightarrow$ computationally universal;
- But $\text{MP}^+\text{V} \subset \text{MP}^+ \subset \text{MP}$ and MP^+V is computationally universal $\Rightarrow$ **MP is computationally universal**!

Definition (MP^+V Grammar)

An 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:
    1. $\emptyset \to v''$;
    2. $v' \to \emptyset$; or
    3. $v' \to v''$;
2. $\forall \varphi \in \Phi$ and $m', m'' \in M$, the flux has either the form $\varphi = m'$ or $\varphi = m' - m''$.

Pre-Print

The Computational Universality of Metabolic Computing

Ricardo Henrique Gracini Guiraldelli and Vincenzo Manca
University of Verona
{ricardo.guiraldelli,vincenzo.manca}@univr.it

May 12, 2015

Abstract

System and synthetic biology are rapidly evolving systems, but both lack tools such as those used in engineering environments to shift the their focus from the design of parts (details) to the design of systems (behaviors); to aggravate, there are insufficient theoretical justifications on the computational limits of biological systems. To diminish these deficiencies, we present theoretical results over the Turing-equivalence of metabolic systems, defines rules for translations of algorithms into metabolic P systems and presents a software tool to assist the task in an automatic way.

Figure: arXiv:1505.02420 [3]

Subsection 2

Metabolic P systems $\mapsto$ Algorithms

Metabolic P systems $\mapsto$ Algorithms

Q: Can I map every algorithm to MP system?

- According to previous theorem, yes!
- Even using register machines and $MP^{+}V$, there are some complications:
  1. unordered application of rules
  2. parallel application of rules

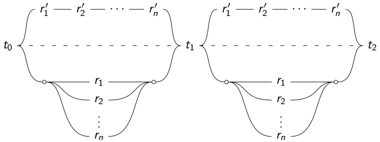


Figure: Graphical representation of the block of execution. [2]

  3. positivity control

Q: Can I map every algorithm to MP system?

- According to previous theorem, yes!
- Even using register machines and MP$^+$V , there are some complications:
  1. unordered application of rules
  2. parallel application of rules

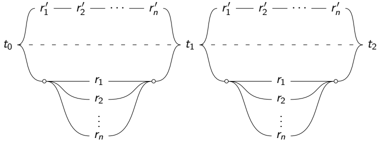


Figure: Graphical representation of the block of execution. [2]

  3. positivity control

Q: Can I map every algorithm to MP system?

- According to previous theorem, yes!
- Even using register machines and $MP^{+}V$, there are some complications:
  1. application of rules
  2. positivity control

Solution 1

Command block and/or Monad

Q: Can I map every algorithm to MP system?

- According to previous theorem, yes!
- Even using register machines and $MP^{+}V$, there are some complications:
  1. application of rules
  2. positivity control

Solution 2

Inclusion of a **subprogram**

Runtime of a Computational Step

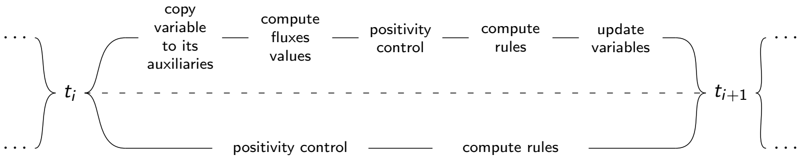


Figure: Representation of a computation step MP$^+$V systems (lower part) and its equivalent register machine (upper part). [2]

Pseudo-code of MP$^+$V $\mapsto$ Register Machine

```
while R_HALT = 0 do
    for all variable v ∈ M do                    ▷ copy variables to auxiliaries
        R_v' ← R_v
    end for
    for all flux φ ∈ Φ do                         ▷ compute fluxes
        R_φ ← φ(t_i)
    end for
    for all variable v ∈ M do                     ▷ positivity control property
        for all flux φ_v^- ∈ Φ_v^- do
            R_sum ← R_sum + R_{φ_v^-}
        end for
        if R_sum > v then
            for all flux φ_v^- ∈ Φ_v^- do
                R_{φ_v^-} ← 0
            end for
        end if
    end for
    for all rule r do                             ▷ compute rules
        if r is of the form ∅ → v : φ then
            R_v' ← R_v' + φ
        else if r is of the form v → ∅ : φ then
            R_v' ← R_v' − φ
        else          ▷ hence, it must be of the form v_1 → v_2 : φ
            R_{v_1'} ← R_{v_1'} + φ
            R_{v_2'} ← R_{v_2'} − φ
        end if
    end for
    for all variable v ∈ M do                     ▷ update variables
        R_v ← R_v'
    end for
end while
```

Rules are Easy. . .

$\emptyset \rightarrow V_1 : \varphi$	1 $\texttt{ADD}(R_{V_1}, R_\varphi, R_{aux})$ 2 $\texttt{CPY}(R_{aux}, R_{V_1})$
$V_1 \rightarrow \emptyset : \varphi$	1 $\texttt{SUB}(R_{V_1}, R_\varphi, R_{aux})$ 2 $\texttt{CPY}(R_{aux}, R_{V_1})$
$V_1 \rightarrow V_2 : \varphi$	1 $\texttt{SUB}(R_{V_1}, R_\varphi, R_{aux})$ 2 $\texttt{CPY}(R_{aux}, R_{V_1})$ 3 $\texttt{ADD}(R_{V_2}, R_\varphi, R_{aux})$ 4 $\texttt{CPY}(R_{aux}, R_{V_2})$
$V_1 \rightarrow HALT : \varphi$	1 $\texttt{JNZ}(R_{HALT}, 3)$ 2 $\texttt{JMP}(4)$ 3 $\texttt{HALT}$ 4 $\texttt{SUB}(R_{V_1}, R_\varphi, R_{aux})$ 5 $\texttt{CPY}(R_{aux}, R_{V_1})$ 6 $\texttt{ADD}(R_{HALT}, R_\varphi, R_{aux})$ 7 $\texttt{CPY}(R_{aux}, R_{HALT})$

. . . The Surroundings Aren't!

- There to ensure the proper/correct execution;
- Hidden dynamics of the system;
- 80% of the process, most of the generated source code;
- Processes:
  1. Copy variables values to auxiliaries registers;
  2. Compute fluxes values for current computational step;
  3. Perform *positivity control* on every rule;
  4. Update the variables values with computed ones;
  5. Loop the systems up to fixed-point HALT $\neq 0$.

Copy variables values to auxiliaries registers

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

Compute fluxes values for current computational step

if $\varphi = V$ **then**

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

else $\triangleright$ Hence, $\varphi = V_1 - V_2$

1 $\texttt{SUB}(R_{V_1}, R_{V_2}, R_\varphi)$

end if

Update the variables values with computed ones

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

. . . But MP$^+$ Is Hard!

- Two contrains to satisfy:
  1. Fluxes must always belong to the set of positive number;

$$\varphi'(t_i) = \begin{cases} \varphi(t_i) & , \text{ if } \varphi(t_i) \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$$

  2. Sum of all consuming fluxes for a given variable must be smaller or equal to the amount of the variable

$$\sum_{\varphi' \in \Phi'^{-}_{x}} \varphi'(t_i) \leq x$$

. . . But MP$^+$ Is Hard!

- Two contrains to satisfy:
  1. Fluxes must always belong to the set of positive number;

     ✅ Condition satisfied by $\varphi : \mathbb{N} \mapsto \mathbb{N}$

  2. Sum of all consuming fluxes for a given variable must be smaller or equal to the amount of the variable

$$\sum_{\varphi' \in \Phi'^{-}_{x}} \varphi'(t_i) \leq x$$

. . . But MP$^+$ Is Hard!

- Two contrains to satisfy:
  1. Fluxes must always belong to the set of positive number;

     Condition satisfied by $\varphi : \mathbb{N} \mapsto \mathbb{N}$
  2. Sum of all consuming fluxes for a given variable must be smaller or equal to the amount of the variable

$$\sum_{\varphi' \in \Phi'^{-}_{x}} \varphi'(t_i) \leq x$$

... But MP^+ Is Hard!

$$\sum_{\varphi' \in \Phi'^{-}_{x}} \varphi'(t_i) \leq x$$

$$\sum_{\varphi' \in \Phi'^{-}_{x}} \varphi'(t_i) \begin{cases} 1 & \mathtt{CLR}(R_{sum}) \\ 2 & \mathtt{ADD}(R_{\varphi_1}, R_{sum}, R_{sum}) \\ 3 & \mathtt{ADD}(R_{\varphi_1}, R_{sum}, R_{sum}) \\ 4 & \mathtt{ADD}(R_{\varphi_2}, R_{sum}, R_{sum}) \\ & \vdots \\ \text{k+1} & \mathtt{ADD}(R_{\varphi_k}, R_{sum}, R_{sum}) \end{cases}$$

$$\text{Comparison} \begin{cases} \text{k+2} & \mathtt{CPY}(R_V, R_{comparator}) \\ \text{k+3} & \mathtt{JZ}(R_{sum}, 2 \cdot k + 8) \\ \text{k+4} & \mathtt{JZ}(R_{comparator}, k + 8) \\ \text{k+5} & \mathtt{DEC}(R_{sum}) \\ \text{k+6} & \mathtt{DEC}(R_{comparator}) \\ \text{k+7} & \mathtt{JMP}(k + 3) \end{cases}$$

$$\text{Failure: } \forall \varphi', \varphi' \leftarrow 0 \begin{cases} \text{k+8} & \mathtt{CLR}(R_{\varphi_1}) \\ \text{k+9} & \mathtt{CLR}(R_{\varphi_2}) \\ & \vdots \\ 2\cdot\text{k+7} & \mathtt{CLR}(R_{\varphi_k}) \end{cases}$$

Another Theorem

Theorem (Translation of $\mathrm{MP}^{+}\mathrm{V}$ to Register Machine)

For any $MP^{+}V$ grammar $\mathcal{M}_{+}$ exists an equivalent register machine $\mathcal{R}$.

Corollary

For any computable MP grammar $\mathcal{M}$ exists an equivalent register machine $\mathcal{R}$.

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] and the very simple subclass derived from it are used, in here, to present 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.

Figure: Presented at *16th Conference on Membrane Computing (CMC16)* [3].

Recalling the Guiding Questions

1. **Q:** How can I represent metabolism?
   A: ✅ *Metabolic P systems.*
2. **Q:** How can I represent circuit?
   A: ✅ *Algorithms.*
3. **Q:** Can I map every MP system to algorithm?
   A: ✅ *Yes, since they are computable.*
4. **Q:** Can I map every algorithm to MP system?
   A: ✅ *Yes.*
5. **Q:** What is the map procedure? (Both.)
   A: ✅ *Proof of theorems.*
6. **Q:** Do I have restrictions?
   A: ✅ *Yes: MP must be computable.*
7. **Q:** Is the mapping optimal? (In which sense?)
   A: ✅ *Yes:* MP^+V *is a minimalist set.*

Section 4

Practical Applications

Guiding Goals

1. Instance of a metabolism as an electrical circuit.
2. Instance of an electrical circuit as a metabolism.
3. Automatic mapping of metabolism to electrical circuit.
4. Automatic mapping of electrical circuit to metabolism.

Subsection 1

Bidirectional Compiler

Compiler $\equiv$ Automatic Translation

- Bidirectional compiler:
  1. Register machine $\mapsto$ MP$^+$V ;
  2. MP$^+$V $\mapsto$ register machine.
- Available in three flavors:
  1. Library;
  2. Standalone command-line application;
  3. Standalone web interface.
- $\approx$ 100% coded in Haskell;
  - 34 files, 1802 lines-of-code;
  - Except few Javascript, CSS and HTML code for web interface.

Compiler ≡ Automatic Translation

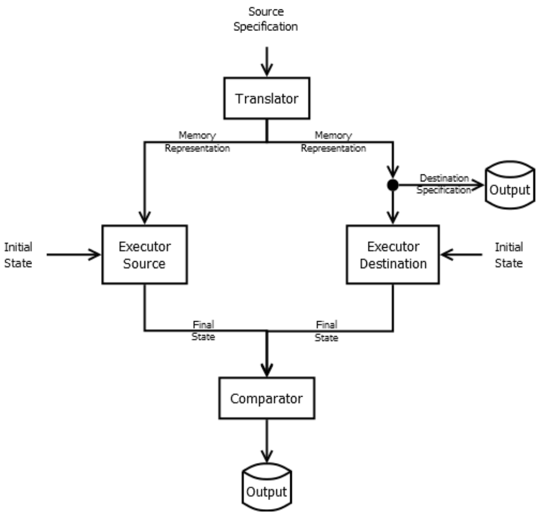


Figure: Relation among modules in the compiler.

Compiler ≡ Automatic Translation

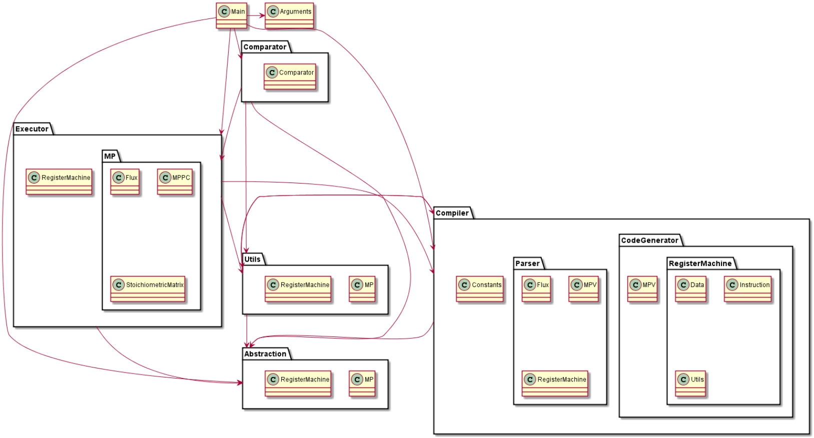


Figure: Relation among modules in the compiler.

Live-Action!

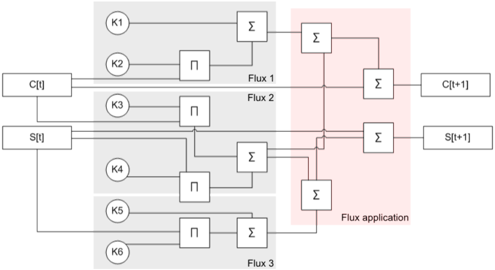
Subsection 2

Digital Circuit

VHDL Hardware Implementation

- MP system $\mapsto$ VHDL $\mapsto$ FPGA;
  - VHDL is algorithmic representation of the digital circuit;
  - FPGA is the digital circuit *per se*.
- Derived from a general framework discovered;
- 100% done at Vilniaus Gediminos Technikos Universitetas when in Erasmus Plus ;
  - They are starting a team on the field with a PhD student.

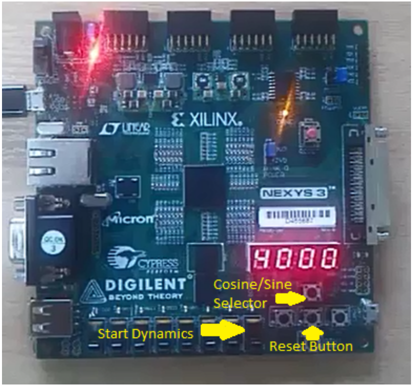
VHDL Hardware Implementation




```
-- 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;
```

VHDL Hardware Implementation



Subsection 3

Discrete Fourier Transform

MP-DFT

- Discrete Fourier transform using MP power to:
    1. generate periodic signals (here, sine and cosine);
    2. numeric regression (LGSS).
- Frenquencies:
    - In a fixed-range;
    - Dynamically computed using τ of MP and Nyquist frequency.
- Benchmark:
    - Accuracy is better than MATLAB/FFTW;
    - Speed is not so promising, one order of magnitude slower;
    - MATLAB + JVM *vs.* native *divide-and-conquer* code.

MP-DFT

Signals	Frequency	Numerical Frequency	FFT	MP-FFT	MP-FFT (MATLAB)
1	20	20	20	20	20
2	$20 + df$	20.5	20.5	20.5	20.5
3	$20 + \frac{3}{4} \cdot df$	20.375	20.5	{20, 20.5}	{20 20.5}
4	$\{20, 47\}$	$\{20, 47\}$	{20, 47.5}	{20, 47, 47.5}	{20 47}
5	$\{20 + df, 47 + 3 \cdot df\}$	$\{20.5, 48.5\}$	{20.5, 49}	{20.5, 48.5, 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}	{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	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	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	20.5

Recalling Guiding Goals

1. ✅ Instance of a MP system as an electrical circuit and algorithm.
2. ✅ Instance of an algorithm as a MP system.
3. ✅ Automatic mapping of MP system to algorithm.
4. ✅ Automatic mapping of algorithm to MP system.

Section 5

Conclusions

Conclusions

- MP is more sound, theoretically speaking;
  - Computationally universal, solving past pendencies [6, 7];
  - Minimalistic subclass $MP^{+}V$.
- Definition of translation procedures in both-ways.
- Practical examples, hardware and software.
- Open-field for new studies, such as optimization (of translation) and super-computation (using $MP^{+}V$).

Thank you!

Obrigado!

Grazie!

Ačiū!

Bibliography I

Alberto Castellini, Giuditta Franco, and Vincenzo Manca.
Hybrid functional Petri nets as MP systems.
Natural Computing, 9:61–81, 2010.

Ricardo Henrique Gracini Guiraldelli and Vin Manca.
Automatic translation of $MP^{+}V$ systems to register machines.
In G. Rozenberg, A. Salomaa, J. Sempere, and C. Zandron, editors, *Sixteenth Conference on Membrane Computing (CMC16)*, Lecture Notes in Computer Science. Springer, 2015.
To appear.

Ricardo Henrique Gracini Guiraldelli and Vincenzo Manca.
The computational universality of metabolic computing.
Available as pre-print at `http://arxiv.org/abs/1505.02420`, 2015.

Frans Johansson.
The Medici effect: what elephants & epidemics can teach us about innovation.
Harvard Business Review Press, 2006.

Terje Lømo.
The discovery of long-term potentiation.
Philosophical transactions of the Royal Society of London. Series B, Biological sciences, 358(1432):617–20, April 2003.

Bibliography II

Vincenzo Manca, Luca Bianco, and Federico Fontana.
Evolution and oscillation in P systems: Applications to biological phenomena.
In Giancarlo Mauri, Gheorghe Păun, Mario J. P'erez-Jim'enez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 3365 of *Lecture Notes in Computer Science*, pages 63–84. Springer Berlin Heidelberg, 2005.

Vincenzo Manca and Rosario Lombardo.
Computing with multi-membranes.
In Marian Gheorghe, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Sergey Verlan, editors, *Membrane Computing*, volume 7184 of *Lecture Notes in Computer Science*, pages 282–299. Springer Berlin Heidelberg, 2012.

Tadao Murata.
Petri nets: Properties, analysis and applications.
Proceedings of the IEEE, 77(4):541–580, 1989.

J. C. Shepherdson and H. E. Sturgis.
Computability of recursive functions.
Journal of the ACM, 10:217–255, 1963.