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

Ricardo H. Gracini Guiraldelli

ricardo.guiraldelli@univr.it

http://ricardo.guiraldelli.com/

Vincenzo Manca

vicenzo.manca@univr.it

http://profs.sci.univr.it/~manca/

Università degli Studi di Verona

August 19th, 2015

General Idea

General Idea

Brief Background

Register Machines

- one *computationally universal/Turing complete* model of computation among the several existent
- similar to (real) computer architecture: von Neumann architecture, memory bank, cache memory, assembly, FPGA, ...
- ⚠️ several descriptions; ours is the Shepderson & Sturgis[1] + subprograms
  1. $\texttt{CPY}(R_1, R_2) \equiv R_2 \leftarrow R_1$
  2. $\texttt{ADD}(R_1, R_2, R_3) \equiv R_3 \leftarrow R_1 + R_2$
  3. $\texttt{SUB}(R_1, R_2, R_3) \equiv R_3 \leftarrow R_1 - R_2$

[1] Shepherdson, J. C. and Sturgis, H. E. *Computability of Recursive Functions*. Journal of ACM, 10, pp. 217–255. 1963.

Register Machine: Specification

- triple $\mathcal{R} = (R, O, P)$
  - $R = \{R_1, R_2, \ldots R_m\}$ is the finite set of *registers* (with infinite capacity)
  - $O' = \overbrace{\{\texttt{INC}, \texttt{DEC}, \texttt{CLR}, \texttt{JMP}, \texttt{JZ}, \texttt{JNZ}, \texttt{HALT}\}}^{\text{Instructions}} \cup \overbrace{\{\texttt{CPY}, \texttt{ADD}, \texttt{SUB}\}}^{\text{Subprograms}}$ is the extended, finite set of operations and subprograms;
  - $P = (I_1, I_2, \ldots, I_n)$ is the (finite) program

Metabolic P Systems

- a model of membrane computing (P systems)
- with particular features:
  - generative grammar for temporal series
  - discrete dynamical systems
  - deterministic execution 🙌
- completely described and studied in Manca's book[2] (and tomorrow's talk! 📆)

[2] Manca, V. *Infobiotics: Information in Biotic Systems*. Springer Berlin Heidelberg. 2013.

Metabolic P Systems: Example

Lotka-Volterra

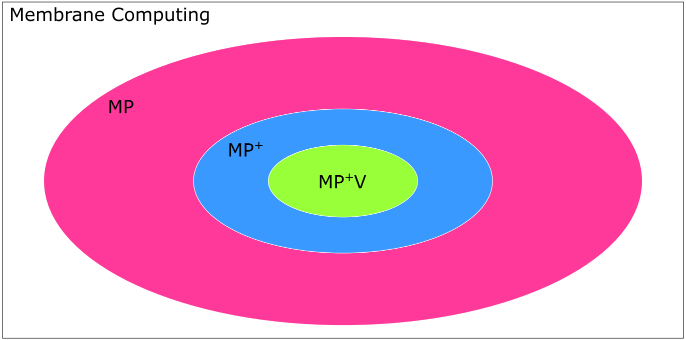
- Rules

$$
\begin{array}{llcl}
(r_1) & X \rightarrow 2X & : & 30 \times 10^{-3} \cdot x \\
(r_2) & X \rightarrow \emptyset & : & 10^{-3} \times (9 + 9 \cdot x \cdot y + 0.1 \cdot x^2 \cdot y) \\
(r_3) & Y \rightarrow 2Y & : & 10^{-3} \times (9 + 15 \cdot x \cdot y + 0.3 \cdot x^2 \cdot y) \\
(r_4) & Y \rightarrow \emptyset & : & 66 \times 10^{-3} \cdot x
\end{array}
$$

- Equation

$$
\underbrace{\begin{bmatrix} x[t+1] \\ y[t+1] \end{bmatrix}}_{\vec{s}[t+1]} = \underbrace{\begin{bmatrix} x[t] \\ y[t] \end{bmatrix}}_{\vec{s}[t]} + \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}}_{\text{Stoichiometric Matrix } \mathbb{A}} \times \underbrace{\begin{bmatrix} \varphi_1(\vec{s}[t]) \\ \varphi_2(\vec{s}[t]) \\ \varphi_3(\vec{s}[t]) \\ \varphi_4(\vec{s}[t]) \end{bmatrix}}_{\vec{U}[t]}
$$

The Universe of the Metabolic Systems



$MP^+V \subseteq MP^+$ Systems

- MP^+ : subclass of MP that is semantically closer to biological metabolism
- restricts quantities and operations to positive numbers $(\mathbb{N}, \mathbb{Q}^+, \mathbb{R}^+, \ldots)$
- and does it using two properties:
  1. every flux ≥ 0, at any time
  2. fluxes cannot remove more quantities than available in the system, at each step

MP$^+$: Mathematically

Definition (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 ${\Phi'}_x^- = \{{\varphi'}_j : \mathrm{mult}^-(x, r_j) > 0, \forall r_j \in R\}$; otherwise, $\varphi'(t_i) = 0, \forall \varphi' \in {\Phi'}_x^-$ at the execution step t_i.

MP^+V Systems

- MP^+V : a minimalist MP^+ , **still** Turing complete!
- arose as a pattern on the equivalence between MP^+ and register machines[3]
- rule: at most **one single variable** at each side of it
- flux: either **one single variable** or **a subtraction of two variables**

[3] Guiraldelli, R. and Manca, V. *The Computational Universality of Metabolic Computing*. arXiv:1505.02420. 2015.

MP^{+}V Systems: Formally Speaking

Definition (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:
   1.1 $\emptyset \rightarrow v''$;
   1.2 $v' \rightarrow \emptyset$; or
   1.3 $v' \rightarrow v''$;
2. $\forall \varphi \in \Phi$ and $m', m'' \in M$, the flux has either the form $\varphi = m'$ or $\varphi = m' - m''$.

The Translation

Translation: Disclamer

- no surprises: $\mathrm{MP}^{+}\mathrm{V} \equiv$ register machine $\equiv$ Turing machine $\Rightarrow \exists$ compilation 😳
- however, the current focus is **compilation**, not computational power[3]. 😉

[3] Guiraldelli, R. and Manca, V. *The Computational Universality of Metabolic Computing*. arXiv:1505.02420. 2015.

Not So Fast: Problems

MP systems differ from register machine in three main ways:

1. unordered application of rules
2. parallel application of rules
3. positivity control

Not So Fast: Problems

MP systems differ from register machine in three main ways:

1. unordered application of rules
2. parallel application of rules
3. positivity control

Solution 1–2

Command block and/or Monad

Not So Fast: Problems

MP systems differ from register machine in two main ways:

1. application of rules
2. positivity control

Solution 1

Command block and/or Monad

Not So Fast: Problems

MP systems differ from register machine in two main ways:

1. application of rules
2. positivity control

Solution 2

Inclusion of a **subprogram**

Runtime of a Computational Step

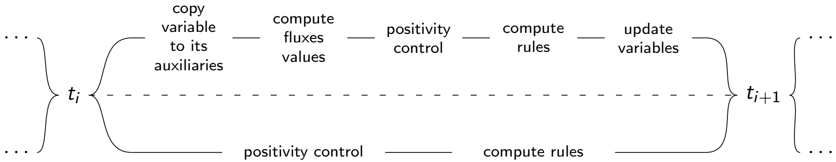


Figure: Representation of a computation step MP$^+$V systems (lower part) and its equivalent register machine (upper part).

Standard MP$^+$V Rules

Rule $\emptyset \to V_1$

$$\emptyset \to V_1 : \varphi$$

1 $\texttt{ADD}(R_{V_1}, R_\varphi, R_{aux})$
2 $\texttt{CPY}(R_{aux}, R_{V_1})$

Rule $V_1 \to \emptyset$

$V_1 \to \emptyset : \varphi$

1 $\texttt{SUB}(R_{V_1}, R_\varphi, R_{aux})$
2 $\texttt{CPY}(R_{aux}, R_{V_1})$

Rule $V_1 \to V_2$

$$\begin{gathered} V_1 \to V_2 : \varphi \\ \equiv \\ \begin{cases} V_1 \to \emptyset : \varphi \\ \emptyset \to V_2 : \varphi \end{cases} \end{gathered}$$

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 $\texttt{CPY}(R_{aux}, R_{V_2})$

Rule $V_1 \rightarrow HALT$

$V_1 \rightarrow HALT : \varphi$

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 "Exoskeleton" of $MP^{+}V$

Exoskeleton Is **The** Important Part

- With experience, basic rules arises fast
  - `HALT` requires some reasoning
- Exoskeleton is the tricky part
  - always there to ensure the proper/correct overall execution
  - the hidden dynamics of the system
  - $4/5$ of the process, most of the generated source code

The Exoskeleton Rules

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 $\neq 0$

The Exoskeleton Rules

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 $\neq 0$

Values Update in Computational Step

- copy variables values to auxiliaries registers
- compute fluxes values for current computational step
- update the variables values with computed ones

Values Update in Computational Step

- copy variables values to auxiliaries registers

$$\forall V \in M, \quad \forall t \in \mathbb{N}, \quad \exists V_{aux} : V_{aux}|_{t^-} \leftarrow V|_t$$

```
1    CPY(R_V, R_V_aux)
```

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

- compute fluxes values for current computational step
- update the variables values with computed ones

Values Update in Computational Step

- copy variables values to auxiliaries registers
- compute fluxes values for current computational step

$$\forall \varphi \in \Phi,$$
$$\forall t \in \mathbb{N},$$
$$\varphi[t] = \varphi(\vec{V}|_t)$$

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

- update the variables values with computed ones

Values Update in Computational Step

- copy variables values to auxiliaries registers
- compute fluxes values for current computational step
- update the variables values with computed ones

$$\forall V \in M,$$
$$\forall t \in \mathbb{N},$$
$$\exists V_{aux} : V|_{t+1} \leftarrow V_{aux}|_{t^+}$$

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

Positivity Control

- must satisfy two contrains
  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. the sum of all consuming fluxes for a given variable must be smaller or equal to the amount of the variable

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

Positivity Control 🛠️

- must satisfy two contrains
  1. fluxes must always belong to the set of positive number

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

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

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

Positivity Control 🛠

- must satisfy two contrains
  1. fluxes must always belong to the set of positive number

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

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

Positivity Control

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

$\sum_{\varphi' \in \Phi'^{-}_{x}} \varphi'(t_i)$	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	$\texttt{ADD}(R_{\varphi_2}, R_{sum}, R_{sum})$
		⋮
	k+1	$\texttt{ADD}(R_{\varphi_k}, R_{sum}, R_{sum})$
Comparison	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	$\texttt{JMP}(k + 3)$
Failure: $\forall \varphi', \varphi' \leftarrow 0$	k+8	$\texttt{CLR}(R_{\varphi_1})$
	k+9	$\texttt{CLR}(R_{\varphi_2})$
		⋮
	2·k+7	$\texttt{CLR}(R_{\varphi_k})$

Perpetuum Mobile, or Not

- as dynamics, it shouldn't stop—unless it stucks in a fixed-point[3] 😉
- as computational process, it **must** stop—unless of halting problem ⚠️
- is it possible to differ `while(true)` from `for(i = 0; i < limit; i++)` at *compile time*?

[3] Guiraldelli, R. and Manca, V. *The Computational Universality of Metabolic Computing*. arXiv:1505.02420. 2015.

Perpetuum Mobile, or Not

- as dynamics, it shouldn't stop—unless it stucks in a fixed-point[3]
- as computational process, it **must** stop—unless of halting problem
- is it possible to differ `while(true)` from `for(i = 0; i < limit; i++)` at *compile time*?
  - Yes, it is!
  - Trick #1: require HALT variable and $V_i \rightarrow HALT$ rule!
  - Trick #2: HALT $\neq 0$ is the *signal to halt*—and since there aren't any $HALT \rightarrow V_i$ rules, it is guaranteed

[3] Guiraldelli, R. and Manca, V. *The Computational Universality of Metabolic Computing*. arXiv:1505.02420. 2015.

Perpetuum Mobile, or Not

$\nexists$HALT variable

$\vee$

$\nexists V_i \rightarrow HALT$

1	`CPY(...,...)`
	$\vdots$
ℓ-1	`JMP(1)`
ℓ	`HALT`

$\exists$HALT variable

$\wedge$

$\exists V_i \rightarrow HALT$

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

Pseudo-code of a Translation from MP$^+$V to 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_v1' ← R_v1' + φ
            R_v2' ← R_v2' − φ
        end if
    end for
    for all variable v ∈ M do                                ▷ update variables
        R_v ← R_v'
    end for
end while
```

Conclusions

Conclusions

- paradigm change possibly brings *big exoskeleton*
  - *critical insight* over the failures of the past
  - metabolic $\mapsto$ computational
  - parallel $\mapsto$ sequential
  - local (pair substances) $\mapsto$ global (execution control)
- $MP \supseteq MP^{+} \supseteq MP^{+}V$, and all computationally universal

Conclusions

- paradigm change possibly brings *big exoskeleton*
  - *critical insight* over the failures of the past
  - metabolic $\mapsto$ computational
  - parallel $\mapsto$ sequential
  - local (pair substances) $\mapsto$ global (execution control)
- MP $\supseteq$ MP$^+$ $\supseteq$ MP$^+$V , and all computationally universal

Conclusions

- bidirectional, automatic translation between von Neumann architecture (register machine) and metabolic computing ($MP^{+}V$ systems)
  - Can we extend it to real metabolism? Yes, we can... Theoretically.[3]
- open door for new translations, including *hardware description languages*, programming languages, visual representations, etc
- lead the way to implementation of circuits based on metabolic (MP) systems

[3] Guiraldelli, R. and Manca, V. *The Computational Universality of Metabolic Computing*. arXiv:1505.02420. 2015.

Conclusions

- bidirectional, automatic translation between von Neumann architecture (register machine) and metabolic computing (MP^+V systems)
  - Can we extend it to real metabolism? Yes, we can... Theoretically.[3]
- open door for new translations, including *hardware description languages*, programming languages, visual representations, etc
- lead the way to implementation of circuits based on metabolic (MP) systems

[3] Guiraldelli, R. and Manca, V. *The Computational Universality of Metabolic Computing*. arXiv:1505.02420. 2015.

Thank you!

Obrigado!

¡Gracias!

Grazie!

Ačiū!