Formal verification and some open problems with Virus Machines

Antonio Ramírez de Arellano Marrero

19th Brainstorming Week of Membrane Computing
Research Group on Natural Computing
Dep. of Computer Science and Artificial Intelligence
Universidad de Sevilla, Seville Spain

Seville, Spain, January 25th, 2022

Index

1. Introduction.
   - Why Virus Machines?
   - Brief definitions
2. Formal verification
3. Some open problems

Introduction

Why Virus Machines?

Why Virus Machines?

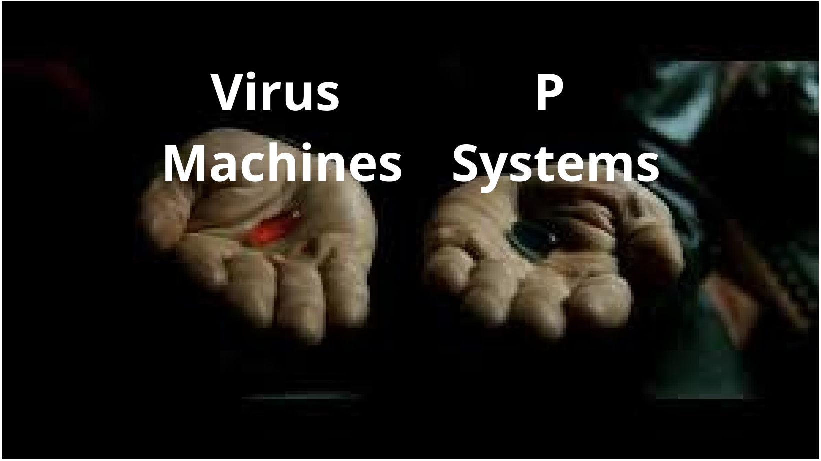


Figure: PhD meeting

Why Virus Machines?

Figure: PhD choose

Why Virus Machines?

- New computing parading.

Why Virus Machines?

- New computing parading.
- COVID-19.

Why Virus Machines?

- New computing parading.
- COVID-19.
- What do all mathematicians love?
    - Solving mathematical problems...

Why Virus Machines?

- New computing parading.
- COVID-19.
- What do all mathematicians love?
    - Solving mathematical problems...
    - And invariants!

Brief definitions

Syntax

Definition (Virus Machine)

A virus machine of degree (p, q), $p \geq 1, q \geq 1$ is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$, where:

- $\Gamma = \{v\}$ is the singleton alphabet:
- $H = \{h_1, \ldots, h_p\}$ (**host**), $I = \{i_1, \ldots, i_q\}$ (**instructions**) are ordered sets and h_{out} represents the **environment**;
- $D_H = (H \cup \{h_{out}\}, E_H, w_H)$ weighted directed graph (WDG);
- $D_I = (I, E_I, w_I)$ is WDG;
- $G_C = (V_C, E_C)$ undirected bipartite graph;
- $n_j \in \mathbb{N}$, for each j, $1 \leq j \leq p$.

X. Chen, M.J. Pérez-Jiménez, L. Valencia-Cabrera, B. Wang, X. Zeng. Computing with viruses. *Theoretical Computer Science*, **623** (2016), 146–159.

$$\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$$

Singleton alphabet

$$\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$$

Singleton alphabet

$$\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \dots, n_p, i_1, h_{out})$$

Ordered sets

Singleton alphabet

WDG

$$\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$$

Ordered sets

Bipartite Graph

Singleton alphabet

WDG

Number of viruses

$$\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$$

Ordered sets

Bipartite Graph

Singleton alphabet

WDG

Number of viruses

$$\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$$

Ordered sets

Bipartite Graph

Initial instruction

$$\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \dots, n_p, i_1, h_{out})$$

Singleton alphabet

WDG

Number of viruses

Environment

Ordered sets

Bipartite Graph

Initial instruction

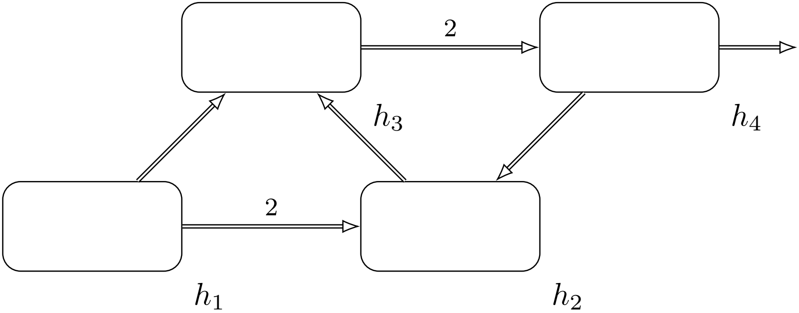


Figure: Host Graph of a Virus Machine

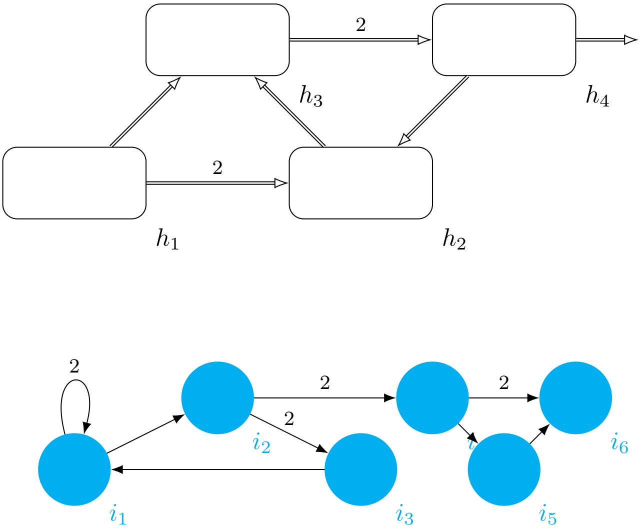


Figure: Host Graph and Instruction Graph of a Virus Machine

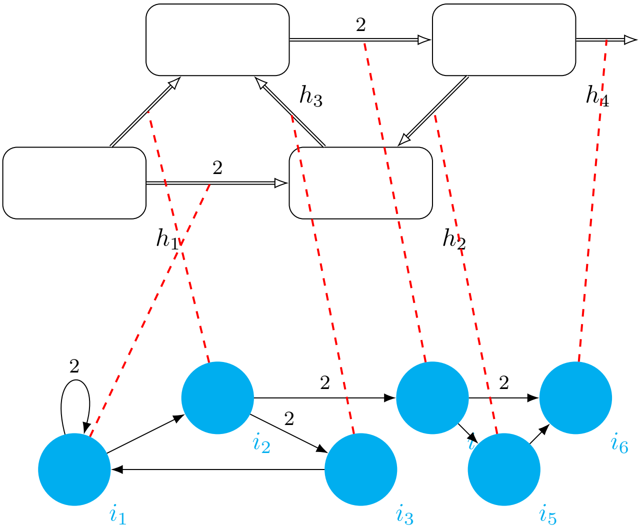


Figure: Heterogeneous network of a Virus Machine

Semantics

Definition

An *instantaneous description* or a *configuration* $\mathcal{C}_t$ at an instant t is a tuple $(a_{1,t}, \ldots, a_{p,t}, u_t, a_{0,t})$ where $a_{0,t}, a_{1,t}, \ldots, a_{p,t}$ are natural numbers and $u_t \in I \cup \{\#\}$.

Semantics

Definition

An *instantaneous description* or a *configuration* $\mathcal{C}_t$ at an instant t is a tuple $(a_{1,t}, \dots, a_{p,t}, u_t, a_{0,t})$ where $a_{0,t}, a_{1,t}, \dots, a_{p,t}$ are natural numbers and $u_t \in I \cup \{\#\}$.

Definition

A *computation* $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ of a virus machine Π is a (possibly infinite) sequence of configurations such that $\mathcal{C}_0$ is the initial configuration of Π and for each $t \in \mathbb{N}, \mathcal{C}_t \Rightarrow_\Pi \mathcal{C}_{t+1}$. A computation $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_k)$ is called a *halting computation* if there exists a k such that $\mathcal{C}_k$ is a halting configuration; that is, $u = \#$.

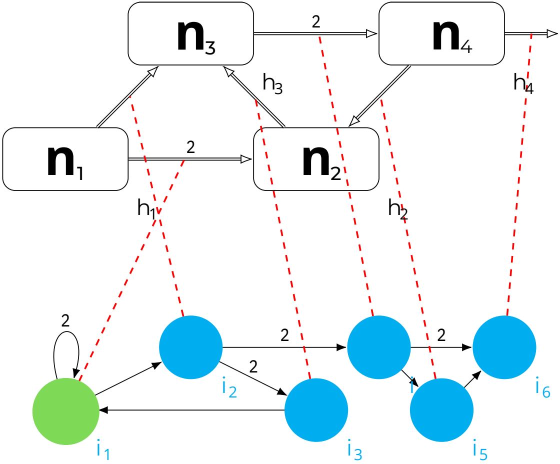


Figure: Configuration: $C_0 = (n_1, n_2, n_3, n_4, i_1, 0)$

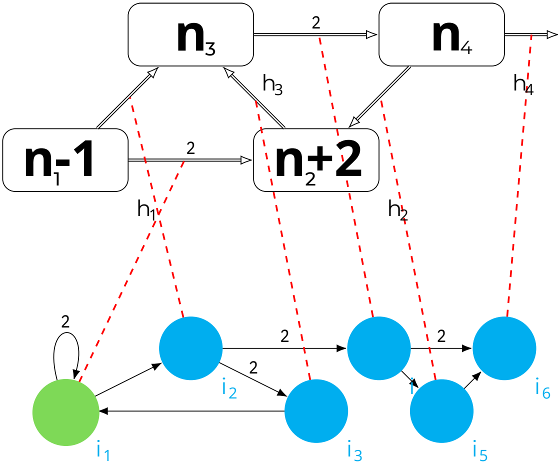


Figure: Configuration: $C_1 = (n_1 - 1, n_2 + 2, n_3, n_4, i_1, 0)$

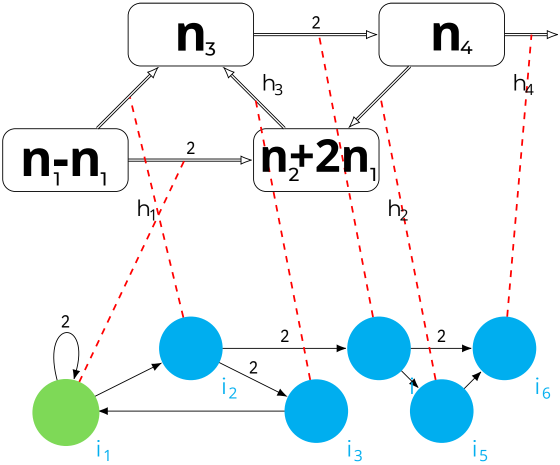


Figure: Configuration: $C_{n_1} = (n_1 - n_1, n_2 + n_1, n_3, n_4, i_1, 0)$

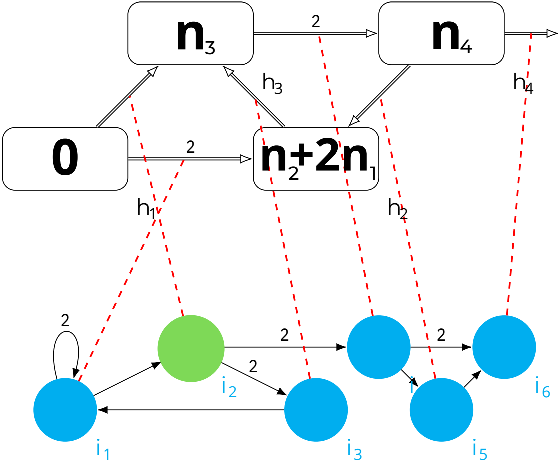


Figure: Configuration: $C_{n_1+1} = (0, n_2 + 2n_1, n_3, n_4, i_2, 0)$

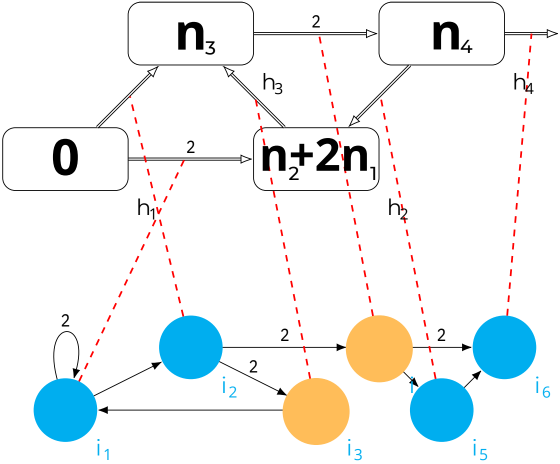


Figure: Non-deterministic behavior

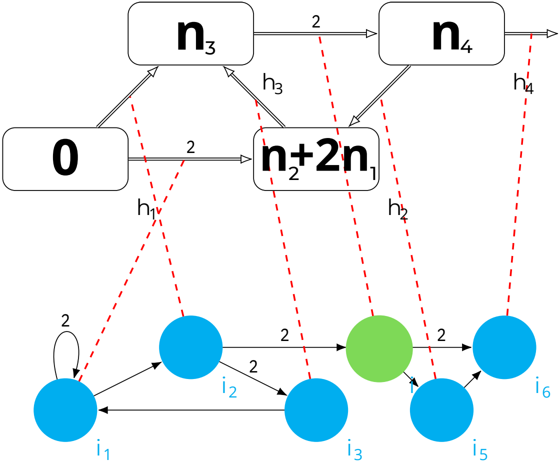


Figure: Configuration: $C_{n_1+2} = (0, n_2 + 2n_1, n_3 - 1, n_4 + 2, i_4, 0)$

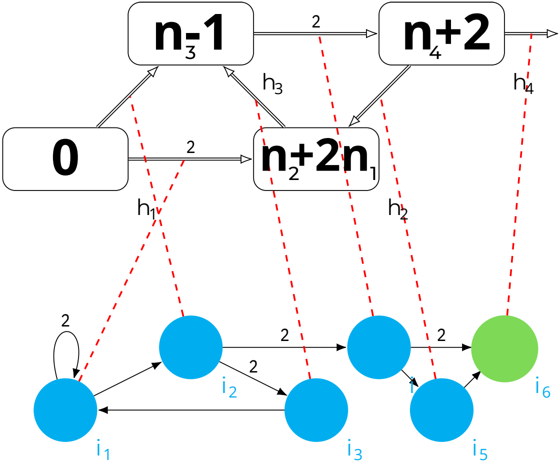


Figure: Configuration: $C_{n_1+3} = (0, n_2 + 2n_1, n_3 - 1, n_4 + 2, i_6, 0)$

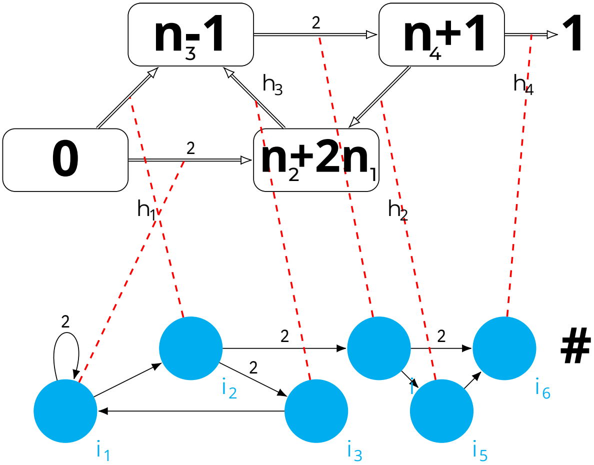


Figure: Configuration: $C_{n_1+3} = (0, n_2 + 2n_1, n_3 - 1, n_4 + 1, \#, 1)$

Computing

Definition

A *virus machine with input of degree* (p, q, r) [4], $p, q, r \geq 1$ is a tuple $\Pi = (\Gamma, H, H_r, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$, where:

- $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ is a virus machine of degree (p, q); and
- $H_r = \{h_{j_1}, \ldots, h_{j_r}\} \subseteq H$ is the ordered set of r *input host* and $h_{out} \notin H_r$.

Computing

Definition

A *virus machine with input of degree* (p, q, r) [4], $p, q, r \geq 1$ is a tuple $\Pi = (\Gamma, H, H_r, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$, where:

- $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ is a virus machine of degree (p, q); and
- $H_r = \{h_{j_1}, \ldots, h_{j_r}\} \subseteq H$ is the ordered set of r *input host* and $h_{out} \notin H_r$.

The initial configuration of a virus machine with input $(a_1, \ldots, a_r) \in \mathbb{N}^r$ is given by $(n_1, \ldots, n_{j_1} + a_1 \ldots, n_{j_r} + a_r, \ldots, n_p)$, and it will be denoted by $\Pi + (a_1, \ldots, a_r)$ ($\Pi + a$ for single inputs).

Formal Verification

What is formal verification?

Formal verification[1] is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

[1] en.wikipedia.org/wiki/Formal_verification

What is formal verification?

Formal verification[1] is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

A method to formally verify that a computational device of a model solves a given problem is to find **invariant formulas** in some relevant loops of the device, in such a way that the veracity of those formulas at the end of the loops provides relevant information.

[1] en.wikipedia.org/wiki/Formal_verification

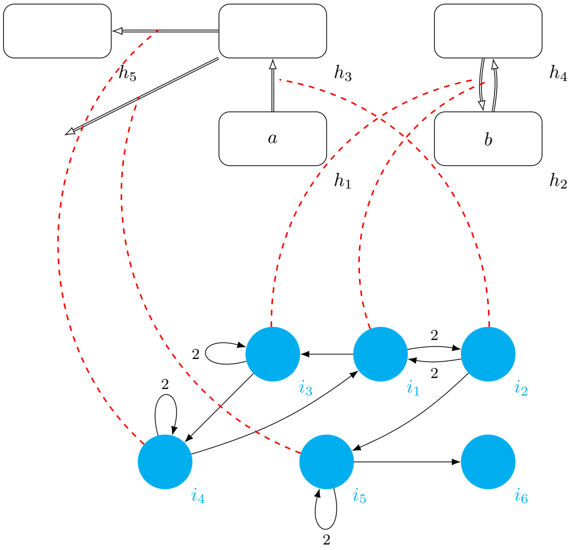


Figure: Virus machine with input $\Pi_{rem} + (a, b)$ computing the remainder of the integer division

Formal verification

Let (a, b) be the input of Π and $a = q \cdot b + r$. Then the following invariant holds:

$$\phi(k) \equiv \mathcal{C}_{k(4b+3)} = (a - b \cdot k, b, 0, 0, b \cdot k, i_1, 0), \text{ for } 0 \leq k \leq q$$

Formal verification

Let (a, b) be the input of Π and $a = q \cdot b + r$. Then the following invariant holds:

$$\phi(k) \equiv \mathcal{C}_{k(4b+3)} = (a - b \cdot k, b, 0, 0, b \cdot k, i_1, 0), \text{ for } 0 \leq k \leq q$$

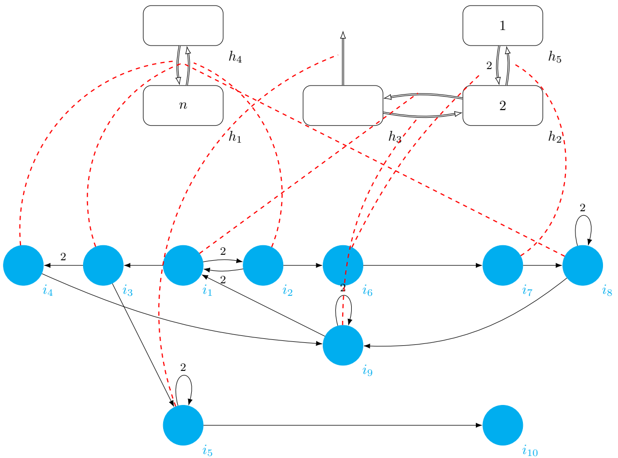
In particular, $\phi(q)$ is true, that is

$$\phi(q) \equiv \mathcal{C}_{q(4b+3)} = (\underbrace{a - b \cdot k}_{r}, b, 0, 0, b \cdot q, i_1, 0),$$

The halting configuration is

$$\mathcal{C}_{q \cdot (4b+3)+r+3} = (0, b - (r+1), 0, r+1, b \cdot q, \#, r)$$

Computing the Least Prime Divisor



i_1 activates $h_2 \to h_3$

i_2 activates $h_1 \to h_4$

i_3 activates $h_1 \to h_4$

i_4 activates $h_4 \to h_1$

i_5 activates $h_3 \to env$

i_6 activates $h_5 \to h_2$ (2)

i_7 activates $h_2 \to h_5$

i_8 activates $h_4 \to h_1$

i_9 activates $h_3 \to h_2$

Figure: Virus machine solving the LPD problem

In the case that the input n of the virus machine is any odd natural number, that is $n = p_1 \cdot q_{p_1}$ where $p_1 \neq 2$ is the minimum prime factor and $q_{p_1} = \dfrac{n}{p_1}$. Let us consider the following notation:

1. For every $j \in \mathbb{N}$, such that $j \geq 2$, we consider:

$$\alpha_j = q_j(3 \cdot j + 4) + 3r_j + n + 7$$

   where q_j and r_j are the quotient and the reminder of the integer division between n and j, i.e. they satisfy $n = j \cdot q_j + r_j$.

2. Let $\beta_k = \alpha_2 + \cdots + \alpha_k$, for every natural number $k \geq 2$.

The following invariant holds:

$$\psi(k) \equiv C_{\beta_k} = (n, k+1, 0, 0, 1, i_1, 0), \text{ for } 2 \leq k \leq p_1 - 1$$

Formal verification

In particular $\psi(p_1 - 1)$ is true, that is

$$\psi(p_1 - 1) \equiv C_{\beta_{p_1 - 1}} = (n, p_1, 0, 0, 1, i_1, 0),$$

The halting configuration is

$$C_{\beta_{p_1 - 1} + (q_{p_1})(3p_1 + 4)} = (0, 0, 0, n, 1, i_1, p_1),$$

Some open problems

Efficiency

Efficiency

- Soft Parallelism: A set of instructions can be activated a the same transition step.

Efficiency

- Soft Parallelism: A set of instructions can be activated a the same transition step.

- MC ingredients:
    - Excitation of the host. (Maybe tomorrow)

Efficiency

- Soft Parallelism: A set of instructions can be activated a the same transition step.
- MC ingredients:
  - Excitation of the host. (Maybe tomorrow)
  - Dead hosts, threshold in the number of viruses,mutation...

Soft Parallelism

Definition

A *parallel virus machine of degree* (p, q) , $p, q \geq 1$ is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, I_0, h_{out})$, where:

- $I_0 = \{i_{j_1}, \ldots i_{j_k}\} \subseteq I$ is the ordered set of k initial instructions; and
- $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_{j_r}, h_{out})$ is a virus machine of degree (p, q) for every $i_{j_r} \in I_0$.

Soft Parallelism

Definition

A *parallel virus machine of degree* (p, q) , $p, q \geq 1$ is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, I_0, h_{out})$, where:

- $I_0 = \{i_{j_1}, \ldots i_{j_k}\} \subseteq I$ is the ordered set of k initial instructions; and
- $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_{j_r}, h_{out})$ is a virus machine of degree (p, q) for every $i_{j_r} \in I_0$.

What about semantics? Let represent the semantics of a virus machine as s, that is, if u_t is the instruction that will be activated at ant step t, then $s(u_t) = u_{t+1}$.

Soft Parallelism

Definition

A *parallel virus machine of degree* (p, q) , $p, q \geq 1$ is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \dots, n_p, I_0, h_{out})$, where:

- $I_0 = \{i_{j_1}, \dots i_{j_k}\} \subseteq I$ is the ordered set of k initial instructions; and
- $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \dots, n_p, i_{j_r}, h_{out})$ is a virus machine of degree (p, q) for every $i_{j_r} \in I_0$.

What about semantics? Let represent the semantics of a virus machine as s, that is, if u_t is the instruction that will be activated at ant step t, then $s(u_t) = u_{t+1}$.
The idea of the semantic of parallel virus machines is

$$I_{t+1} = \{s(i_k) \ : \ i_k \in I_t\}$$

Soft Parallelism

Definition

A *parallel virus machine of degree* (p, q) , $p, q \geq 1$ is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \dots, n_p, I_0, h_{out})$, where:

- $I_0 = \{i_{j_1}, \dots i_{j_k}\} \subseteq I$ is the ordered set of k initial instructions; and
- $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \dots, n_p, i_{j_r}, h_{out})$ is a virus machine of degree (p, q) for every $i_{j_r} \in I_0$.

What about semantics? Let represent the semantics of a virus machine as s, that is, if u_t is the instruction that will be activated at ant step t, then $s(u_t) = u_{t+1}$.
The idea of the semantic of parallel virus machines is

$$I_{t+1} = \{s(i_k) \ : \ i_k \in I_t\}$$

This has to be well defined

Let's go to the black(green)board!

Thank you for your attention!

Romero-Jiménez, Á., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: Generating diophantine sets by virus machines. In: Gong, M., Linqiang, P., Tao, S., Tang, K., Zhang, X. (eds.) Bio-Inspired Computing – Theories and Applications. pp. 331–341. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

Chen, X., Pérez-Jiménez, M.J., Valencia-Cabrera, L., Wang, B., Zeng, X.: Computing with viruses. Theoretical Computer Science 623, 146–159 (2016).

Romero-Jiménez, Á., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.:Computing partial recursive functions by virus machines. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) Membrane Computing. pp. 353–368. Springer International Publishing, Cham (2015)

Ramírez-de Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Basic arithmetic calculations through virus-based machines. In: Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Adeli, H. (eds.) Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence. pp. 403–412. Springer International Publishing, Cham (2022).