Theoretical Computer Science 623 (2016) 146–159

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Computing with viruses

Xu Chen [a], Mario J. Pérez-Jiménez [b], Luis Valencia-Cabrera [b], Beizhan Wang [a], Xiangxiang Zeng [c,*]

[a] School of Software, Xiamen University, Xiamen 361005, Fujian, People's Republic of China
[b] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Seville 41012, Spain
[c] Department of Computer Science, Xiamen University, Xiamen 361005, Fujian, People's Republic of China

ARTICLE INFO

Article history:
Received 12 August 2015
Received in revised form 11 November 2015
Accepted 7 December 2015
Available online 17 December 2015
Communicated by T. Yokomori

Keywords:
Natural computing
Virus machine
Register machine
Turing completeness

ABSTRACT

In recent years, different computing models have emerged within the area of Unconventional Computation, and more specifically within Natural Computing, getting inspiration from mechanisms present in Nature. In this work, we incorporate concepts in virology and theoretical computer science to propose a novel computational model, called *Virus Machine*. Inspired by the manner in which viruses transmit from one host to another, a virus machine is a computational paradigm represented as a heterogeneous network that consists of three subnetworks: virus transmission, instruction transfer, and instruction-channel control networks. Virus machines provide non-deterministic sequential devices. As number computing devices, virus machines are proved to be computationally complete, that is, equivalent in power to Turing machines. Nevertheless, when some limitations are imposed with respect to the number of viruses present in the system, then a characterization for semi-linear sets is obtained.

1. Introduction

The present study can be considered as a contribution to the area of *natural computing*, which is a field of research that investigates both human-designed computing inspired by nature and computing that occurs in nature. That is, this field investigates models and computational techniques based on nature, which enables the development of new computational tools (in software, hardware, or wetware) for problem solving. Such tools can synthesize natural patterns, behaviours and organisms, which may result in designing novel computing systems that use natural media for computation [15,20].

Biological systems are rich sources of ideas for designing computing devices and algorithms. In fact, they have inspired numerous classical computing devices, such as neural nets [11,21,22,24] and evolutionary algorithms [25,26]. In recent years, computing devices inspired by cells (or molecules inside cells, such as DNA) have been thoroughly investigated [1,3,8,19]. Most cell-inspired computing systems have been proved to be universal [4,10,18,23,27] and computationally efficient [6,12,14,16,17].

In virology, a virus is a parasitic biological agent that can only reproduce after infecting a host cell. Every animal, plant, and protist species on this planet has been infected by viruses. The process of viral multiplication (replication) starts for the

* Corresponding author. Tel.: +86 0592 2580033.
E-mail addresses: chenxu31@sina.com (X. Chen), marper@us.es (M.J. Pérez-Jiménez), lvalencia@us.es (L. Valencia-Cabrera), wangbz@xmu.edu.cn (B. Wang), xzeng@xmu.edu.cn (X. Zeng).

http://dx.doi.org/10.1016/j.tcs.2015.12.006
0304-3975/

attachment stage (a virus attaches to the potential host), then the virus must effect entry to be able to replicate (penetration stage) and the replication steps (genome replication, transcription and translation, using ribosomes provided by the host cell) occur next. When the new genomes are produced they come together with the newly synthesized virus proteins to form virus particles (assembly stage). Finally, the particles escape from the cell to infect other cells.

Viruses can transmit from one host cell to another in various ways. For instance, viruses in plants are usually transmitted from a plant to another by insects (e.g., aphids), which feed on plant sap. Viruses in animals can spread by blood-sucking insects (e.g., mosquitoes). Coughing and sneezing may spread influenza viruses. HIV, a well-known virus, can transmit through sexual contact or by exposure to infected blood. Viruses can only reproduce in living cells. After infecting a host cell, viruses use the machinery and metabolism of the host cell to produce multiple copies of themselves, while the original infecting virus is dismantled. For additional details on viruses, refer to [7].

Besides biological viruses, there is a special kind of "virus", called *computer virus*, which is similar to the natural virus. A computer virus is a malicious software that can spread to other computers through networks. Furthermore, it can replicate by inserting copies of itself into other computer programs, data files, and so on. For additional details on computer viruses, refer to [2].

In this study, we introduce a new computing model, called *Virus Machine*, which is inspired from the transmissions and replications of viruses. This system consists of several *hosts* (processing units), connected to each other by *channels*. Each host can be viewed as a group of cells (being part of a colony, organism, system, organ or tissue). *Viruses* are placed in the hosts and are assumed to evolve. Each virus can transmit from one host to another by passing through a channel, and can replicate itself while transmitting. Such transmissions and replications in the system are controlled by several instruction units, which are attached to the channels. A virus machine can be viewed as a heterogeneous network that consists of the following three subnetworks:

- A *virus transmission network*, which is a weighted directed network wherein each node represents a *host* and each arc represents a *transmission channel* through which viruses can transmit from one host to another or to the environment. In addition, a natural number (the *weight* of the channel) is associated with each channel, indicating the number of viruses that will be transmitted to the tail host of the channel after transmission (i.e., a virus may replicate itself while transmitting).
- An *instruction transfer network*, which is a weighted directed network wherein each node represents a *control instruction unit* and each arc represents an optional *instruction transfer path* with a natural number (the *weight* of the instruction transfer) associated to it.
- An *instruction-channel control network*, which is an undirected bipartite network, wherein each node represents either a control instruction or a channel and each edge represents a control relationship between an instruction and a channel.

Each transmission channel in a virus transmission network is closed by default. It can be opened by a control instruction unit, which is connected to the channel through an edge of the instruction-channel control network, when the instruction is activated. When it is open, the channel allows a virus (only one virus) to transmit through it. The virus may replicate itself while transmitting, which indicates that the number of viruses in the head host of the channel is decreased by one, whereas the number of viruses in the tail host of the channel is increased by n (being n the weight of the opened channel). Instructions are activated individually along the paths in the instruction transfer network, so that only one instruction is enabled in each computation step. That is, an instruction activation signal is transferred to the network to activate instructions in sequence.

In this paper we deal with virus machines having input hosts, allowing us to introduce some additional numbers of viruses (encoding the information) in certain distinguished hosts. Then, instructions are allowed to activate one after another and the system halts when no path is available to transfer the instruction activation signal. We consider virus machines working in the *computing mode*: if the virus machine halts, then the computation is finished and the result is the total number of viruses sent to a distinguished region (a host or the environment) during the computation; otherwise (i.e., if the instruction activation signal is transferred continuously), the computation fails to produce an output.

In this work, the computational completeness of virus machines working in the computing mode is established by showing that they simulate register machines. Nevertheless, by considering a more realistic condition concerning with a limit (an upper bound) on the number of viruses present in any host during a computation (*bounded virus machines*), this restriction diminishes in fact the computational power of the systems. In this case, a characterization of semi-linear sets of numbers is obtained.

As far as we know, this paper is a pioneer work in this area, and we believe that further investigations on this subject are worth conducting. In this regard, some new exciting lines of research for future works are presented at the end of this paper.

The paper is organized as follows. First, some preliminaries are briefly introduced in order to make the work self-contained. Then, in Section 3, we formally define the computing model of Virus Machines and the structure of the model is graphically illustrated. In Section 4, an example is given to illustrate how such systems work. Then, we discuss the power of virus machines in Section 5 and the computational completeness (via simulating register machines) is stated. In Section 6 we also investigate the power of bounded virus machines by giving a characterization of semi-linear sets. Finally, in Sec-

tion 7, the main conclusions of this work are summarized and some suggestions for possible lines of future research are outlined.

2. Preliminaries

In this Section some basic concepts needed throughout this paper are introduced in order to make it self-contained.

2.1. Sets

Given a set A we denote by $\mathcal{P}(A)$ the *power set of* A, that is, $\mathcal{P}(A)$ is the set of all subsets of A. Given a nonempty finite set A, a *weight mapping* w over A is an *additive mapping* from A onto $\mathbb{N}$ (the set of natural numbers). That is, w is a mapping from A onto $\mathbb{N}$ such that it can be extended to the power set of A (the set of all subsets of A) as follows: $w(A') = \sum_{x \in A'} w(x)$, for each subset $A' \subseteq A$.

A set $A \subseteq \mathbb{N}^k$ is a *linear set* if there exist tuples $v_0, v_1, \ldots, v_t \in \mathbb{N}^k$ such that the set A can be described as follows:

$$\{v \in \mathbb{N}^k \mid \text{there exist} \, m_1, \ldots, m_t \in \mathbb{N} \text{ such that } v = v_0 + m_1 v_1 + \cdots + m_t v_t\}$$

A set $A \subseteq \mathbb{N}^k$ is a *semi-linear set* if it is a finite union of linear sets. The family of semi-linear sets of natural numbers is denoted by *SLIN*.

2.2. Alphabets and languages

An *alphabet* Γ is a finite non-empty set, and its elements are called *symbols*. A *string* u over Γ is a mapping from a natural number $n \in \mathbb{N}$ onto Γ. Number n is called *length* of the string u, denoted by $|u|$. The empty string (with length 0) is denoted by λ. If u and v are strings over Γ, then so is their *concatenation* uv, obtained by juxtaposition, that is, writing u and v one after the other. For each string $u \in \Gamma^*$ and $a \in \Gamma$ we denote by $|u|_a$ the number of occurrences of symbol a in the string u.

A *language* over Γ is a set of strings over Γ. For a language L over Γ, the set *length*$(L) = \{|u| \mid u \in L\}$ is called the *length set of* L. The *concatenation of languages* L_1, L_2 over Γ is $L_1 L_2 = \{uv \mid u \in L_1, v \in L_2\}$. Given an ordered alphabet $\Gamma = \{a_1, \ldots, a_k\}$, a language L over Γ is *semi-linear* if $\{(|u|_{a_1}, \ldots, |u|_{a_k}) \mid u \in L\}$ is a semi-linear set. The set *SLIN* is equal to the length set of semi-linear languages.

2.3. Grammars

A *grammar* G is a tuple $(N, T, A, \mathcal{R})$, where N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, $A \in N$ is the *axiom* and $\mathcal{R}$ is a finite set of *rewriting* rules of the form $u \to v$, with $u, v \in (N \cup T)^*$ and u containing at least a non-terminal symbol. When applying such a rule, the string $w = w_1 u w_2$ can be rewritten as $w' = w_1 v w_2$, and we denote it by $w \Rightarrow_G w'$. The *language generated by the grammar* G is $L(G) = \{z \in T^* \mid$ there exist $z_1, \ldots, z_k$ such that $A \Rightarrow_G z_1 \Rightarrow_G \ldots z_k = z\}$. *RE* is the family of languages generated by arbitrary grammars. *NRE* is the family of length sets of languages in *RE*.

A grammar G is *right-linear* if each rule $u \to v$ has $u \in N$ and $v \in T^* \cup T^* N$. *SLIN* is the family of languages generated by right-linear grammars.

2.4. Graphs

An *undirected graph* G is a pair (V, E), where V is a finite set and E is a subset of $\{\{x, y\} \mid x \in V, y \in V, x \neq y\}$. The set V is called the vertex set of G, and its elements are called vertices. The set E is called the edge set of G, and its elements are called *edges*. If $e = \{u, v\} \in E$ is an edge of G, then we say that edge e is incident on vertices u and v. In an undirected graph, the *degree* of a vertex u is the number of edges incident on it. A *path* of length $k \geq 2$ from a vertex u to a vertex v in an undirected graph $G = (V, E)$ is a finite sequence $(w_0, w_1, \ldots, w_k)$ of vertices such that $w_0 = u$, $w_k = v$ and $\{w_j, w_{j+1}\} \in E$ for $j = 0, \ldots, k-1$. If $w_0 = w_k$ then we say that the path is a *cycle*. An undirected graph is *cyclic* if there is at least a cycle in the graph. We say that vertices u, v are connected if there exists a path from u to v. An undirected graph G is *connected* if every pair of vertices is connected by a path.

A *directed graph* G is a pair (V, E), where V is a finite set and E is a subset of $V \times V$. The set V is called the vertex set of G, and its elements are called vertices. The set E is called the arc set of G, and its elements are called *arcs*. In a directed graph, the *out-degree* of a vertex is the number of arcs leaving it, and the *in-degree* of a vertex is the number of arcs entering it. A bipartite graph G is an undirected graph (V, E) in which V can be partitioned into two sets V_1, V_2 such that $\{u, v\} \in E$ implies either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$; that is, all edges are set between the two sets V_1 and V_2 (see [5] for details).

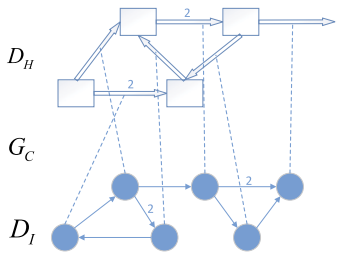

Fig. 1. Structure of a *Virus Machine*.

3. Virus machines

In what follows we formally define the **syntax** of the *Virus Machines*.

Definition 1. A Virus Machine (VM, for short) 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:
- $H = \{h_1, \ldots, h_p\}$ and $I = \{i_1, \ldots, i_q\}$ are ordered sets such that $v \notin H \cup I$ and $H \cap I = \emptyset$;
- $D_H = (H \cup \{h_{out}\}, E_H, w_H)$ is a weighted directed graph, where $E_H \subseteq H \times (H \cup \{h_{out}\})$, $(h, h) \notin E_H$ for each $h \in H$, out-degree $(h_{out}) = 0$, and w_H is a mapping from E_H onto $\mathbb{N} \setminus \{0\}$ (the set of positive integer numbers);
- $D_I = (I, E_I, w_I)$ is a weighted directed graph, where $E_I \subseteq I \times I$, w_I is a mapping from E_I onto $\mathbb{N} \setminus \{0\}$ and, for each vertex $i_j \in I$, the out-degree of i_j is less than or equal to 2;
- $G_C = (V_C, E_C)$ is an undirected bipartite graph, where $V_C = I \cup E_H$, being $\{I, E_H\}$ the partition associated with it (i.e., all edges go between the two sets I and E_H). In addition, for each vertex $i_j \in I$, the degree of i_j is less than or equal to 1;
- $n_j \in \mathbb{N}$ $(1 \leq j \leq p)$;
- $h_{out} \notin I \cup \{v\}$ and h_{out} is denoted by h_0 in the case that $h_{out} \notin H$.

A *Virus Machine* of degree (p, q)

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

can be viewed as an ordered set of p *hosts* labelled with $h_1, \ldots, h_p$ (where each host h_j, $1 \leq j \leq p$, initially contains exactly n_j *viruses* – copies of symbol v –), and an ordered set of q *control instruction units* labelled with $i_1, \ldots, i_q$. Symbol h_{out} represents the *output region* of the system (we use the term *region* to refer to host h_{out} in the case that $h_{out} \in H$ and to refer to the environment in the case that $h_{out} = h_0$). Arcs from the directed graph D_H represent *transmission channels* through which viruses can transmit from one host h_s to another $h_{s'}$ (we assume that $h_s \neq h_{s'}$). Thus, if $(h_s, h_{s'}) \in E_H$, then $h_s \neq h_{out}$ and viruses may transmit from host h_s to host $h_{s'}$ (if $s' = 0$, viruses may exit to the environment). Each channel is *closed* by default until it is opened by a control instruction (which is attached to the channel by means of an edge in graph G_C) when that instruction is *activated*. Furthermore, each channel $(h_s, h_{s'})$ is assigned with a positive natural number weight denoted by $w_{s,s'}$ or $w(h_s, h_{s'})$, which indicates the number of viruses that will be transmitted/replicated to the tail host of the channel (the virus may replicate itself while transmitting). Arcs from the directed graph D_I represent *instruction transfer paths*, being each arc $(i_j, i_{j'}) \in E_I$ assigned with a weight denoted by $w_{j,j'}$. Finally, the undirected bipartite graph G_C represents the *instruction-channel network* by which an edge $\{i_j, (h_s, h_{s'})\}$ indicates a control relationship between instruction i_j and channel $(h_s, h_{s'})$: when instruction i_j is activated, the channel $(h_s, h_{s'})$ is opened and then *one virus* can be transmitted from host h_s to $h_{s'}$, possibly being replicated $w_{s,s'}$ times in the target host. Let us note that, according to our definition, each control instruction is attached to, at most, one transmission channel.

Graphically, a virus machine of degree $(4, 6)$, with 4 hosts and 6 control instructions, can be represented as a heterogeneous network consisting of three graphs, as illustrated in Fig. 1. Each host is depicted as a rectangle and each instruction is depicted as a circle. Each arrow is either a virus transmission channel linking the hosts (or pointing to the environment), or an instruction transfer path linking the instructions; in both cases, each arrow is assigned with a positive integral weight (the weight 1 is not marked for simplicity). The control relationships between instructions and channels are represented as dotted lines.

In what follows the **semantics** associated with the computing model of virus machines is described. An *instantaneous description* or a *configuration* $\mathcal{C}_t$ at an instant t of a virus machine is described by a tuple $(a_{1,t}, \ldots, a_{p,t}, u_t, e_t)$ where $a_{1,t}, \ldots, a_{p,t}, e_t$ are natural numbers, $u_t \in I \cup \{\#\}$, where $\# \notin H \cup \{h_0\} \cup I$. The meaning of a *configuration* $\mathcal{C}_t$ is the following:

at instant t the host h_s of the system contains exactly $a_{s,t}$ viruses, the output region h_{out} contains exactly e_t viruses and, if $u_t \in I$, then the control instruction unit u_t will be activated at step $t+1$ (otherwise, if $u_t = \#$, then no instruction will be activated). The *initial configuration* of the system $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ is $\mathcal{C}_0 = (n_1, \ldots, n_p, i_1, 0)$, that is, we assume that initially no viruses appear in the output region h_{out}. A configuration $\mathcal{C}_t = (a_{1,t}, \ldots, a_{p,t}, u_t, e_t)$ is a *halting configuration* if and only if u_t is the object #. We say that a non halting configuration $\mathcal{C}_t = (a_{1,t}, \ldots, a_{p,t}, u_t, e_t)$ yields configuration $\mathcal{C}_{t+1} = (a_{1,t+1}, \ldots, a_{p,t+1}, u_{t+1}, e_{t+1})$ in one *transition step*, denoted by $\mathcal{C}_t \Rightarrow_\Pi \mathcal{C}_{t+1}$, if we can pass from $\mathcal{C}_t$ to $\mathcal{C}_{t+1}$ in the following form.

(a) First, given that $\mathcal{C}_t$ is a non halting configuration we have $u_t \in I$. Then the control instruction unit u_t is activated.
(b) Let us assume that instruction u_t is attached to a channel $(h_s, h_{s'})$. Then this channel will be opened and:
  - If $a_{s,t} \geq 1$, then a virus (only one virus) is consumed from host h_s and $w_{s,s'}$ copies of v are produced in host $h_{s'}$ (if $s' \neq out$) or in the output region h_{out}. That is, the virus is replicated $w_{s,s'}$ times during the transmission.
  - If $a_{s,t} = 0$, then there is no transmission of viruses: no virus is consumed from host h_s and no virus is produced in host $h_{s'}$ (if $s' \neq out$) or in the output region (if $s' = out$).
(c) Let us assume that instruction u_t is not attached to any channel $(h_s, h_{s'})$. Then there is no transmission of viruses.
(d) Object $u_{t+1} \in I \cup \{\#\}$ is obtained as follows:
  - Let us suppose that *out-degree*$(u_t) = 2$, that is, there are two different instructions $u_{t'}$ and $u_{t''}$ such that $(u_t, u_{t'}) \in E_I$ and $(u_t, u_{t''}) \in E_I$.
    * If instruction u_t is attached to a channel $(h_s, h_{s'})$:
      · If $a_{s,t} \geq 1$ (there are some viruses in host h_s in $\mathcal{C}_t$) then u_{t+1} is $u_{t'}$ or $u_{t''}$, the instruction corresponding to *the highest* weight path $(\max\{w_{t,t'}, w_{t,t''}\})$; if $w_{t,t'} = w_{t,t''}$, the next instruction is selected in a non-deterministic way.
      · If $a_{s,t} = 0$ (no virus exists in host h_s in $\mathcal{C}_t$), u_{t+1} is the instruction corresponding to *the lowest* weight path $(\min\{w_{t,t'}, w_{t,t''}\})$; if $w_{t,t'} = w_{t,t''}$ the next instruction is selected in a non-deterministic way.
    * If instruction u_t is not attached to a channel, then the next instruction u_{t+1} ($u_{t'}$ or $u_{t''}$) is selected in a non-deterministic way.
  - Let us suppose that *out-degree*$(u_t) = 1$. In this case the system behaves deterministically and u_{t+1} is the instruction that verifies $(u_t, u_{t+1}) \in E_I$.
  - Let us suppose that *out-degree*$(u_t) = 0$. Then u_{t+1} is the object #, and configuration $\mathcal{C}_{t+1}$ is a halting configuration.

A *computation* of a virus machine Π is a (finite or infinite) sequence of configurations such that: (a) the first term is the initial configuration $\mathcal{C}_0$ of the system; (b) for each $n \geq 2$, the n-th term of the sequence is obtained from the previous term in one transition step; and (c) if the sequence is finite (called *halting computation*) then the last term is a halting configuration.

All the computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded in the contents of the output region associated with the halting configuration. If $\mathcal{C} = \{\mathcal{C}_t\}_{t \leq l}$ of Π ($l \in \mathbb{N}$) is a halting computation, then the *length* of $\mathcal{C}$, denoted by $|\mathcal{C}|$, is l.

Let us note that in virus machines the output region (the environment in the case that $h_{out} = h_0$) plays a *passive* role, in the following sense: along any computation, the output region can receive viruses from the system but no virus can enter the system from that region.

Definition 2. A Virus Machine with input of degree (p, q, r), $p \geq 1$, $q \geq 1$, $r \geq 1$, 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:

- $(\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ is a Virus Machine of degree (p, q).
- $H_r = \{h_{i_1}, \ldots, h_{i_r}\} \subseteq H$ is the ordered set of r input hosts and $h_{out} \notin H_r$.

Given a virus machine Π with input of degree (p, q, r), for each r-tuple $(\alpha_1, \ldots, \alpha_r)$ of natural numbers, the *initial configuration* of Π *with input* $(\alpha_1, \ldots, \alpha_r)$ is

$$(n_1, \ldots, n_{i_1} + \alpha_1, \ldots, n_{i_r} + \alpha_r, \ldots, n_p, i_t, 0)$$

That is, the number α_j is added to the initial number n_{i_j} of viruses in the input host h_{i_j}, for $1 \leq j \leq r$. Therefore, in a virus machine with input we have an initial configuration associated with each input r-tuple $(\alpha_1, \ldots, \alpha_r)$ of natural numbers. A computation of a virus machine Π with input $(\alpha_1, \ldots, \alpha_r)$ starts with the initial configuration, and proceeds as stated above. The result of a halting computation of a virus machine Π with input $(\alpha_1, \ldots, \alpha_r)$ working in the *computing mode* is the total number n of viruses sent to the output region during that computation. We say that number n is *computed* by the virus machine Π with input $(\alpha_1, \ldots, \alpha_r)$, denoted by $\Pi + (\alpha_1, \ldots, \alpha_r)$.

We also denote by $N(\Pi + (\alpha_1, \ldots, \alpha_r))$ the set of all natural numbers computed by $\Pi + (\alpha_1, \ldots, \alpha_r)$. Throughout this paper, $h_{out} = h_0$, that is, the environment will be the output region.

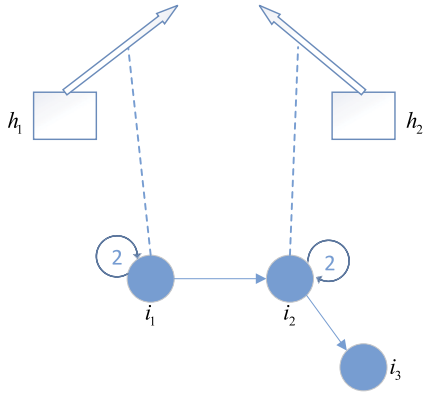

Fig. 2. A *VM Adder*.

4. An example of virus machine

In this Section, we give an example to further illustrate the way virus machines work. Let us consider the virus machine with input of degree (2,3,2)

$$VM_{Sum} = (\Gamma, H, H_2, I, D_H, D_I, G_C, 0, 0, i_1, h_{out})$$

working in the computing mode, defined as follows:

- $\Gamma = \{v\}$, $H = H_2 = \{h_1, h_2\}$,
  $I = \{i_1, i_2, i_3\}$, $h_{out} = h_0$.
- $D_H = (\{h_0, h_1, h_2\}, E_H, w_H)$, where
  $E_H = \{(h_1, h_0), (h_2, h_0)\}$ and $w_H(h_1, h_0) = w_H(h_2, h_0) = 1$.
- $D_I = (I, E_I, w_I)$, where
  $E_I = \{(i_1, i_1), (i_1, i_2), (i_2, i_2), (i_2, i_3)\}$ and
  $w_I(i_1, i_1) = w_I(i_2, i_2) = 2$, $w_I(i_1, i_2) = w_I(i_2, i_3) = 1$.
- $G_C = (I \cup E_H, E_C)$, where
  $E_C = \{\{i_1, (h_1, h_0)\}, \{i_2, (h_2, h_0)\}\}$.

The Virus machine VM_{Sum} computing the sum of two natural numbers is depicted in Fig. 2.

For each $n_1, n_2 \in \mathbb{N}$, the initial configuration of the system VM_{Sum} with input (n_1, n_2) is $\mathcal{C}_0 = (n_1, n_2, i_1, 0)$, and it computes as follows.

- In the first step, the instruction i_1 is activated, thus making the channel $h_1 \to h_0$ open to transmit exactly one virus from h_1 to the environment. Note that the virus does not replicate itself while transmitting because the weight of the channel is 1. Then, the instruction activation signal transfers along the higher weight path $i_1 \xrightarrow{2} i_1$ (i.e., the instruction i_1 will be activated again in the next step). Thus, $\mathcal{C}_1 = (n_1 - 1, n_2, i_1, 1)$.
- Then, the instruction i_1 will be activated repeatedly n_1 times, making all n_1 viruses in h_1 be sent to the environment via the channel $h_1 \to h_0$. Thus, the configuration of the system becomes $\mathcal{C}_{n_1} = (0, n_2, i_1, n_1)$.
- In the step $n_1 + 1$, the instruction i_1 is activated, but the host h_1 is empty now, which means that no transmission occurs in the channel $h_1 \to h_0$, making the instruction activation signal transfer along the lower weight path $i_1 \to i_2$ (i.e., the instruction i_2 will be activated in the next step). Thus, $\mathcal{C}_{n_1+1} = (0, n_2, i_2, n_1)$.
- In the step $n_1 + 2$, the instruction i_2 is activated, making the channel $h_2 \to h_0$ open to transmit exactly one virus from the host h_2 to the environment. Thus, $\mathcal{C}_{n_1+2} = (0, n_2 - 1, i_2, n_1 + 1)$.
- Similarly, the instruction i_2 will be activated repeatedly n_2 times, sending all n_2 viruses in h_2 to the environment via the channel $h_2 \to h_0$, hence the configuration of the system becomes $\mathcal{C}_{n_1+n_2+1} = (0, 0, i_2, n_1 + n_2)$.
- In the step $n_1 + n_2 + 2$, the instruction i_2 is activated. Because no transmission occurs in the channel $h_2 \to h_0$, the instruction activation signal transfers along the lower weight path $i_2 \to i_3$, making the instruction i_3 activate in the next step. Thus, $\mathcal{C}_{n_1+n_2+2} = (0, 0, i_3, n_1 + n_2)$.
- In the step $n_1 + n_2 + 3$, the instruction i_3 is activated. In this step, no transmission occurs because no channel is attached to i_3. Given that *out-degree*$(i_3) = 0$, no next instruction exists receiving activation signal, so $\mathcal{C}_{n_1+n_2+3} = (0, 0, \#, n_1 + n_2)$, which is a halting configuration. Hence, the output of the system VM_{Sum} with input (n_1, n_2) is $n_1 + n_2$ (i.e., the number of viruses sent to the environment along the computation is $n_1 + n_2$).

Table 1
The computation in *VM Adder*.

Step t	Ins.	$a_{1,t}$	$a_{2,t}$	u_t	e_t	Config.
		n_1	n_2	i_1	0	$\mathcal{C}_0$
1	i_1	n_1-1	n_2	i_1	1	$\mathcal{C}_1$
...	...	...	...	...	...	...
n_1	i_1	0	n_2	i_1	n_1	$\mathcal{C}_{n_1}$
n_1+1	i_1	0	n_2	i_2	n_1	$\mathcal{C}_{n_1+1}$
n_1+2	i_2	0	n_2-1	i_2	n_1+1	$\mathcal{C}_{n_1+2}$
...	...	...	...	...	...	...
n_1+n_2+1	i_2	0	0	i_2	n_1+n_2	$\mathcal{C}_{n_1+n_2+1}$
n_1+n_2+2	i_2	0	0	i_3	n_1+n_2	$\mathcal{C}_{n_1+n_2+2}$
n_1+n_2+3	i_3	0	0	#	n_1+n_2	$\mathcal{C}_{n_1+n_2+3}$

Table 1 presents the computation in the *VM Adder* mentioned above, where e_t denotes the number of viruses sent to the environment up to instant t.

5. The universality of virus machines

In this section, we examine the computational power of *VMs* working in the computing mode. For each $p, q, n \geq 1$, we denote by $NVM(p,q,n)$ the family of all subsets of $\mathbb{N}$ computed by Virus machines with at most p hosts, q instructions, and each host having at most n viruses in any instant of each computation; that is, $NVM(p,q,n) \subseteq \mathcal{P}(\mathbb{N})$. If one of the parameters p, q, n is not bounded, then it is replaced with $*$. In particular, $NVM(*,*,*)$ denotes the family of all subsets of natural numbers computed by virus machines. A *non-restricted Virus Machine* is a virus machine such that there is no restriction on the number of hosts, the number of instructions and the number of viruses contained in any host along any computation.

First, we prove that *non-restricted* virus machines working in the computing mode, are computationally complete; that is, they are able to compute all subsets of natural numbers which are Turing computable. In the proof of this result, we will simulate *register machines* working in the non-deterministic and generative form by means of virus machines with input working in computing mode.

Let us recall that a *register machine* is a tuple $M = (m, H, l_0, l_h, I)$, where $m \geq 1$ is the number of registers, H is a finite set of labels, $l_0 \in H$ is the starting label, $l_h \in H$ is the halting label (assigned to instruction `HALT`), I is the set of instructions *bijectively* labelled by elements of H, so each element from H labels only one instruction from I, thus precisely identifying it. The instructions of M can be of the following forms:

- $l_1 : (\texttt{ADD}(r), l_2, l_3)$, with $l_1 \in H \setminus \{l_h\}$, $l_2, l_3 \in H$, $1 \leq r \leq m$ (add 1 to register r and non-deterministically jump to one of the instructions l_2, l_3).
- $l_1 : (\texttt{SUB}(r), l_2, l_3)$, with $l_1 \in H \setminus \{l_h\}$, $l_2, l_3 \in H$, $1 \leq r \leq m$ (if register r is non-empty, then decrease the value of register r by one and jump to instruction l_2; otherwise go to instruction l_3).
- $l_h : \texttt{HALT}$, with $l_h \in H$ (the halt instruction).

A *configuration* $\mathcal{C}_t$ of a register machine at an instant t is the following $(m+1)$-tuple: $(a_{1,t}, \ldots, a_{m,t}, l_t)$, where $a_{r,t}$, $1 \leq r \leq m$, are natural numbers and $l_t \in H$. The meaning is the following: $a_{r,t}$ is the value of register r at instant t and l_t is the current label, which indicates the next instruction to be executed. All the computations start from the *initial configuration* $\mathcal{C}_0 = (0, \ldots, 0, l_0)$ (each register is empty – i.e., stores the number zero – and the first instruction to be executed is l_0) and end up when reaching the halt instruction (a halting configuration is characterized by the following condition: the last component is l_h). Then, the number n stored in the first register at a halting computation is said to be computed by M. It is known (see, e.g., [13]) that register machines generate/compute all sets of Turing computable numbers, hence they characterize the family *NRE* of Turing computable sets of natural numbers.

We have the following result.

Theorem 1. $NVM(*,*,*) = NRE$.

Proof. It is obvious that $NVM(*,*,*) \subseteq NRE$ due to the universality of Turing machines and given that we assume the Turing-Church thesis. So it suffices to prove $NRE \subseteq NVM(*,*,*)$. For this purpose, let $M = (m, H, l_0, l_h, I)$ be a register machine. To simulate M, we associate with it a virus machine VM_M whose output region is the environment, in such a way that a natural number n is computed by M if and only if it is computed by VM_M.

Each register r, $1 \leq r \leq m$, in M, will be represented as a host h_r in VM_M. Particularly, the first host h_1 in VM_M represents the first register in M. Hence, simulating M means simulating each `ADD` instruction by a module of type $VM_{\texttt{ADD}}$, each `SUB` instruction by a module of type $VM_{\texttt{SUB}}$ and then *assembling* the modules associated with the instructions in M. In addition, a module of type $VM_{\texttt{OUTPUT}}$ will be used to send all viruses in h_1 to the environment.

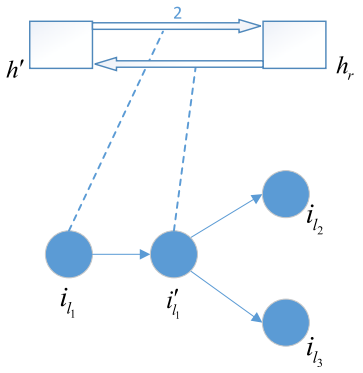

Fig. 3. Module $VM_{\mathrm{ADD}_{l_1}}$ simulating $l_1 : (\mathrm{ADD}(r), l_2, l_3)$.

Module $VM_{\mathrm{ADD}_{l_1}}$ simulating the ADD instruction $l_1 : (\mathrm{ADD}(r), l_2, l_3)$.

Let $VM_{\mathrm{ADD}_{l_1}} = (\Gamma, H, H_1, I, D_H, D_I, G_C, 0, 1, i_{l_1}, h_{out})$ be the virus machine with input of degree $(2, 4, 1)$ if $l_2 \neq l_3$, and of the degree $(2, 3, 1)$ if $l_2 = l_3$, working in computing mode, defined as follows:

- $\Gamma = \{v\}$, $H = \{h_r, h'\}$, $H_1 = \{h_r\}$,
  $I = \{i_{l_1}, i_{l_2}, i_{l_3}, i'_{l_1}\}$, $h_{out} = h_0$.
- $D_H = (\{h_0, h_r, h'\}, E_H, w_H)$, where
  $E_H = \{(h_r, h'), (h', h_r)\}$ and
  $w_H(h_r, h') = 1$, $w_H(h', h_r) = 2$.
- $D_I = (I, E_I, w_I)$, where
  $E_I = \{(i_{l_1}, i'_{l_1}), (i'_{l_1}, i_{l_2}), (i'_{l_1}, i_{l_3})\}$ and
  $w_I(i_{l_1}, i'_{l_1}) = w_I(i'_{l_1}, i_{l_2}) = w_I(i'_{l_1}, i_{l_3}) = 1$.
- $G_C = (I \cup E_H, E_C)$, where
  $E_C = \{\{i_{l_1}, (h', h_r)\}, \{i'_{l_1}, (h_r, h')\}\}$.

The Virus machine $VM_{\mathrm{ADD}_{l_1}}$ is depicted in Fig. 3.

Let us see how $VM_{\mathrm{ADD}_{l_1}}$ simulates the instruction $l_1 : (\mathrm{ADD}(r), l_2, l_3)$. For each $n \in \mathbb{N}$, the initial configuration of the system $VM_{\mathrm{ADD}_{l_1}}$ with input n is $\mathcal{C}_0 = (n, 1, i_{l_1}, 0)$. The module computes as follows.

Firstly, the instruction i_{l_1} is activated, making the channel $h' \xrightarrow{2} h_r$ open to transmit a virus from h' to h_r. Note that the weight of the channel is 2, which means that the virus replicates itself while transmitting, so $\mathcal{C}_1 = (n + 2, 0, i'_{l_1}, 0)$. Then, the instruction i'_{l_1} is activated, making the channel $h_r \to h'$ open to transmit exactly one virus back to h'. Therefore, after two transition steps the number of viruses in host h_r is increased by 1, while the number of viruses in host h' remains unchanged, thus simulating the action that adds 1 to register r in M. Meanwhile, note that *out-degree*$(i'_{l_1}) = 2$, so the system behaves non-deterministically, choosing between two instructions (i.e., i_{l_2} and i_{l_3}) to activate, thus simulating the action of going to one of the instructions with labels l_2, l_3 in M, hence leading to $\mathcal{C}'_2 = (n + 1, 1, i_{l_2}, 0)$ or $\mathcal{C}''_2 = (n + 1, 1, i_{l_3}, 0)$, respectively. Thus, after two computation steps, we have correctly simulated the instruction of type ADD as mentioned.

Module $VM_{\mathrm{SUB}_{l_1}}$ simulating the SUB instruction $l_1 : (\mathrm{SUB}(r), l_2, l_3)$.

Let $VM_{\mathrm{SUB}_{l_1}} = (\Gamma, H, H_1, I, D_H, D_I, G_C, 0, 0, i_{l_1}, h_{out})$ be the virus machine with input of degree $(2, 3, 1)$ if $l_2 \neq l_3$, and of the degree $(2, 2, 1)$ if $l_2 = l_3$, working in computing mode, defined as follows:

- $\Gamma = \{v\}$, $H = \{h_r, h''\}$, $H_1 = \{h_r\}$,
  $I = \{i_{l_1}, i_{l_2}, i_{l_3}\}$, $h_{out} = h_0$.
- $D_H = (\{h_o, h_r, h''\}, E_H, w_H)$, where
  $E_H = \{(h_r, h'')\}$ and $w_H(h_r, h'') = 1$.
- $D_I = (I, E_I, w_I)$, where
  $E_I = \{(i_{l_1}, i_{l_2}), (i_{l_1}, i_{l_3})\}$ and
  $w_I(i_{l_1}, i_{l_2}) = 2$, $w_I(i_{l_1}, i_{l_3}) = 1$.
- $G_C = (I \cup E_H, E_C)$, where
  $E_C = \{\{i_{l_1}, (h_r, h'')\}\}$.

The Virus machine $VM_{\mathrm{SUB}_{l_1}}$ is depicted in Fig. 4.

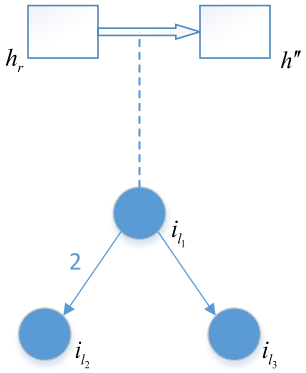

Fig. 4. Module $VM_{\mathrm{SUB}_{l_1}}$ simulating $l_1 : (\mathrm{SUB}(r), l_2, l_3)$.

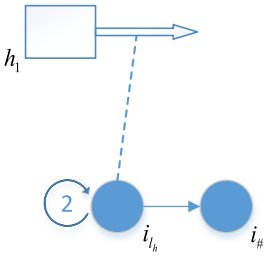

Fig. 5. Module VM_{OUTPUT}.

Let us see how $VM_{\mathrm{SUB}_{l_1}}$ simulates the instruction $l_1 : (\mathrm{SUB}(r), l_2, l_3)$. For each $n \in \mathbb{N}$, the initial configuration of the system $VM_{\mathrm{SUB}_{l_1}}$ with input n is $\mathcal{C}_0 = (n, 0, i_1, 0)$. The module computes as follows.

- If the host h_r is nonempty (i.e., $n \neq 0$), a virus will be transmitted between h_r and h'' via the channel $h_r \to h''$, so that the number of viruses in host h_r will be decreased by 1. Then, the instruction activation signal will transfer along the higher weight path $i_{l_1} \xrightarrow{2} i_{l_2}$, which means that the instruction i_{l_2} will be activated in the next step. Thus, $\mathcal{C}_1 = (n-1, 1, i_{l_2}, 0)$.
- If the host h_r is empty (i.e., $n = 0$), then no virus transmission occurs, so the instruction activation signal will transfer along the lower weight path $i_{l_1} \to i_{l_3}$, making the instruction i_{l_3} activate in the next step. Thus, $\mathcal{C}_1 = (0, 0, i_{l_3}, 0)$.

In this way, after one computation step, we have correctly simulated the instruction of type SUB as mentioned.

Module VM_{OUTPUT} simulating the OUTPUT of the system.

Let $VM_{\mathrm{OUTPUT}} = (\Gamma, H, H_1, I, D_H, D_I, G_C, 0, i_{l_h}, h_{out})$ be the virus machine with input of degree $(1, 2, 1)$, working in computing mode, defined as follows:

- $\Gamma = \{v\}$, $H = \{h_1\}$, $H_1 = \{h_1\}$,
  $I = \{i_{l_h}, i_{\#}\}$, $h_{out} = h_0$.
- $D_H = (\{h_0, h_1\}, E_H, w_H)$, where $E_H = \{(h_1, h_0)\}$ and $w_H(h_1, h_0) = 1$.
- $D_I = (\{i_{l_h}, i_{\#}\}, E_I, w_I)$, where
  $E_I = \{(i_{l_h}, i_{l_h}), (i_{l_h}, i_{\#})\}$
  and $w_I(i_{l_h}, i_{l_h}) = 2$, $w_I(i_{l_h}, i_{\#}) = 1$.
- $G_C = (I \cup E_H, E_C)$, where
  $E_C = \{\{i_{l_h}, (h_1, h_0)\}\}$.

The Virus machine VM_{OUTPUT} is depicted in Fig. 5.

Let us see how the VM_{OUTPUT} module performs its computations. For each $n \in \mathbb{N}$, the initial configuration of the system VM_{OUTPUT} with input n is $\mathcal{C}_0 = (n, i_{l_h}, 0)$. The module computes as follows.

- Firstly, the instruction i_{l_h} is activated. Then,
  - If the host h_1 is nonempty (i.e., $n \neq 0$), a virus will be transmitted between h_1 and the environment via the channel $h_1 \to h_0$, so that the number of viruses in h_1 will be decreased by 1. Then, the instruction activation signal will

transfer along the higher weight path $i_{l_h} \xrightarrow{2} i_{l_h}$, which means that the instruction i_{l_h} will be activated in the next step.
  - If the host h_1 is empty (i.e., $n = 0$), then no virus transmission occurs and the instruction activation signal will transfer along the lower weight path $i_{l_h} \to i_{\#}$, which means that the instruction $i_{\#}$ will be activated in the next step. Given that $out\text{-}degree(i_{\#}) = 0$ the second component of the next configuration will be # and the system VM_{OUTPUT} halts in the next step.
- The instruction i_{l_h} is activated repeatedly until host h_1 is empty. Once it is empty, the next instruction will be $i_{\#}$, so that $\mathcal{C}_{n+1} = (0, i_{\#}, n)$. Then, the system halts with all n viruses having been transferred from host h_1 to the environment. Therefore, the halting configuration will be $\mathcal{C}_{n+2} = (0, \#, n)$.

In this way, the output of the system has been correctly produced.

Note that the same auxiliary host h' can be used for all VM_{ADD} modules, and the same auxiliary host h'' can be used for all VM_{SUB} modules. Even more, just one auxiliary node can be used in the whole system, with no interferences due to the fact that at most one communication channel can be opened during each computation step.

Finally, the virus machine VM_M associated to the register machine M is the union of all modules of types VM_{ADD_l}, $VM_{\mathrm{SUB}_{l'}}$ and VM_{OUTPUT}, for each ADD instruction l and for each SUB instruction l'. The output region of this VM_M is the environment. An important remark is needed: in the module associated with the instruction labelled by l_0, the instruction i_{l_0} must be equal to i_1; that is, instruction i_{l_0} will be the first instruction activated in the virus machine VM_M. At the end of any computation in the virus machine VM_M simulating a register machine M, the number of viruses present in host h_1 will be equal to the number stored in the first register in M. These viruses are then sent by the module VM_{OUTPUT} to the environment, which is the output region of VM_M. Therefore, the theorem is proved. □

6. Bounded virus machines

In this Section, we try to consider a more "realistic" case of virus machines in which the number of viruses present in each host during any computation is bounded. This way, we obtain a bounded system, called *bounded virus machine*. Not very surprisingly, the computational power of such virus machines falls drastically.

Let us recall that *SLIN* is the family of semi-linear sets of natural numbers, which is a subset of $\mathcal{P}(\mathbb{N})$. In what follows, the fact that bounded virus machines working in computing mode characterize *SLIN* is proved.

Theorem 2. *$NVM(*, *, n) = SLIN$, for all $n \geq 2$.*

Proof. The proof of Theorem 2 is a consequence of the following three lemmas presented for auxiliary purposes:

Lemma 1. *$NVM(*, *, n) \subseteq SLIN$.*

Lemma 2. *For each $a \geq 0, d \geq 1$, the arithmetic progression with first term a and common difference d, $\{a + k \cdot d \mid k \geq 1\}$, is in $NVM(3, 6, 2)$.*

Lemma 3. *Let $p, q \geq 1$ and $n \geq 2$, such that $A_1, A_2 \in NVM(p, q, n)$. Then, $A_1 \cup A_2 \in NVM(p, q + 1, n)$.*

Indeed, by Lemmas 2, 3, we can conclude that $SLIN \subseteq NVM(*, *, n)$. Combining with Lemma 1, Theorem 2 is proved. □

Proof of Lemma 1. Let Π be a virus machine with a bound n on the number of viruses present in each host during any computation. Let p be the number of hosts, and q the number of instructions. For each configuration $\mathcal{C}_t = (a_{1,t}, \ldots, a_{p,t}, u_t, e_t)$, we denote by $\mathcal{C}_t^\star$ the tuple $(a_{1,t}, \ldots, a_{p,t}, u_t)$. Given that $a_{1,t}, \ldots, a_{p,t} \leq n$ and the number of instructions u_t is less than or equal to q, we deduce that the number of possible tuples $\mathcal{C}_t^\star$ associated with configurations $\mathcal{C}_t$ reachable by Π from an initial configuration $\mathcal{C}_0$ is finite. We denote by $Reach(\mathcal{C}_0)$ the finite set $\{\mathcal{C}_t^\star \mid \mathcal{C}_t$ is a configuration that is reachable from an initial configuration $\mathcal{C}_0$ in $\Pi\}$, and denote by W_{out} the set of weights of the channels which point to the environment (the output region).

We associate with each initial configuration $\mathcal{C}_0$ of Π the right-linear grammar $G = (N, T, A, \mathcal{R})$ defined as follows:

- $N = Reach(\mathcal{C}_0)$.
- $T = \{v\}$.
- $A = \mathcal{C}_0$.
- $\mathcal{R}$ is the finite set of rules of the form:
  - $\mathcal{C}_t^\star \to \mathcal{C}_{t+1}^\star$, for $\mathcal{C}_t^\star, \mathcal{C}_{t+1}^\star \in Reach(\mathcal{C}_0)$ such that there is a transition $\mathcal{C}_t \Rightarrow \mathcal{C}_{t+1}$ in Π during which no virus is sent to the environment.

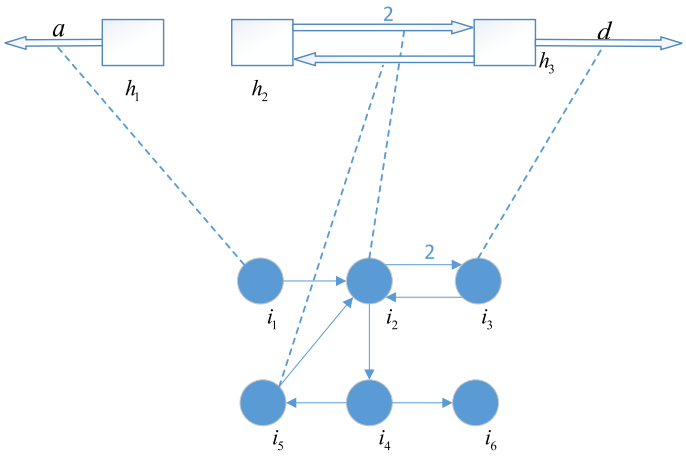

Fig. 6. A *VM* generating an arithmetical progression.

- $\mathcal{C}_t^\star \to v^k\ \mathcal{C}_{t+1}^\star$, for $k \in W_{out}$ and $\mathcal{C}_t^\star, \mathcal{C}_{t+1}^\star \in Reach(\mathcal{C}_0)$ such that there is a transition $\mathcal{C}_t \Rightarrow \mathcal{C}_{t+1}$ in Π during which k viruses are sent to the environment.
- $\mathcal{C}_t^\star \to v^k$, for $k \in W_{out}$ and $\mathcal{C}_t^\star \in Reach(\mathcal{C}_0)$ such that $\mathcal{C}_t$ is a halting configuration and there is a transition $\mathcal{C} \Rightarrow \mathcal{C}_t$ in Π during which k viruses are sent to the environment.
- $\mathcal{C}_t^\star \to \lambda$, for $\mathcal{C}_t^\star \in Reach(\mathcal{C}_0)$ such $\mathcal{C}_t$ is a halting configuration and there is a transition $\mathcal{C} \Rightarrow \mathcal{C}_t$ in Π during which no virus is sent to the environment.

The way of controlling the derivation guarantees that the set $N(\Pi)$ of natural numbers computed by the virus machine Π is the length set of the regular language $L(G)$. Hence, the set $N(\Pi)$ is semi-linear. Therefore, we conclude that $NVM(*,*,n) \subseteq SLIN$. □

Proof of Lemma 2. Let $\Pi(a,d) = (\Gamma, H, I, D_H, D_I, G_C, 1, 1, 0, i_1, h_{out})$ be the virus machine of degree $(3,6)$, working in computing mode, defined as follows:

- $\Gamma = \{v\}$, $H = \{h_1, h_2, h_3\}$,
  $I = \{i_1, i_2, i_3, i_4, i_5, i_6\}$, $h_{out} = h_0$.
- $D_H = (\{h_1, h_2, h_3, h_0\}, E_H, w_H)$, where
  $E_H = \{(h_1, h_0), (h_2, h_3), (h_3, h_2), (h_3, h_0)\}$,
  $w_H(h_1, h_0) = a$, $w_H(h_3, h_0) = d$, $w_H(h_2, h_3) = 2$, $w_H(h_3, h_2) = 1$.
- $D_I = (I, E_I, w_I)$, where
  $E_I = \{(i_1, i_2), (i_2, i_3), (i_3, i_2), (i_5, i_2), (i_2, i_4), (i_4, i_5), (i_4, i_6)\}$,
  $w_I(i_1, i_2) = w_I(i_3, i_2) = w_I(i_5, i_2) = w_I(i_2, i_4) = w_I(i_4, i_5) = w_I(i_4, i_6) = 1$, $w_I(i_2, i_3) = 2$.
- $G_C = (V_C, E_C)$, where $V_C = I \cup E_H$ and the set of edges is:
  $E_C = \{\{i_1, (h_1, h_0)\}, \{i_2, (h_2, h_3)\}, \{i_3, (h_3, h_0)\}, \{i_5, (h_3, h_2)\}\}$.

If $a = 0$, then we assume that $(h_1, h_0) \notin E_H$ and $\{i_1, (h_1, h_0)\} \notin E_C$.

The virus machine $\Pi(a,d)$, consisting of three hosts and six instructions, is depicted in Fig. 6.

The initial configuration of the system is $\mathcal{C}_0 = (1, 1, 0, i_1, 0)$, which means that both h_1 and h_2 contain exactly one virus, and h_3 is empty. The system computes as follows:

(1) In the first step, the instruction i_1 is activated, making the channel $h_1 \xrightarrow{a} h_0$ open to transmit the only virus in h_1 to the environment. Note that the weight of the channel is a, which means that the virus will replicate itself while transmitting, thus resulting in a viruses sent to the environment. Then, the instruction activation signal transfers along the single path $i_1 \to i_2$. Thus, the configuration of the system becomes $\mathcal{C}_1 = (0, 1, 0, i_2, a)$.
(2) In the second step, the instruction i_2 is activated, making the only virus in h_2 be transmitted to h_3 via the channel $h_2 \xrightarrow{2} h_3$. Note that the weight of the channel is 2, so h_3 now contains 2 viruses and the instruction activation signal transfers to i_3 along the higher weight path $i_2 \xrightarrow{2} i_3$. Thus, $\mathcal{C}_2 = (0, 0, 2, i_3, a)$.
(3) In the third step, the instruction i_3 is activated, making a virus from h_3 be transmitted to the environment and replicate itself while transmitting, making the number of viruses in h_3 be decreased by one. Then, the instruction activation signal transfers back to i_2 along the single path $i_3 \to i_2$, making i_2 activate again. Thus, the configuration of the system becomes $\mathcal{C}_3 = (0, 0, 1, i_2, a + d)$.

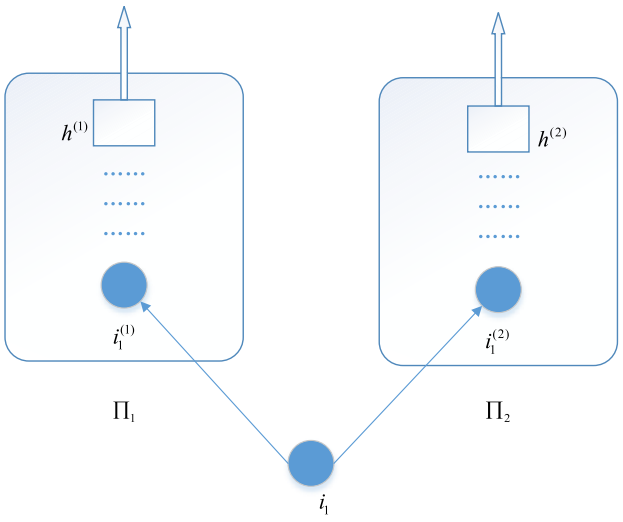

Fig. 7. The idea of the union construction.

(4) In the fourth step, the instruction i_2 is activated, but now host h_2 is empty, which means that no transmission occurs, so the instruction activation signal transfers along the lower weight path $i_2 \to i_4$. Thus, $\mathcal{C}_4 = (0, 0, 1, i_4, a + d)$.

(5) In the fifth step, the instruction i_4 is activated, and no virus transmission occurs because no channel is attached to i_4. Note that *out-degree*$(i_4) = 2$, so the system behaves non-deterministically, choosing between two instructions (i.e., i_5 and i_6) to activate in the next step.

a. If the instruction activation signal transfers along the path $i_4 \to i_5$, then $\mathcal{C}'_5 = (0, 0, 1, i_5, a + d)$. Then, in the next step i_5 is activated, making the only virus in h_3 be transmitted to h_2. Thus, $\mathcal{C}'_6 = (0, 1, 0, i_2, a + d)$. Then, the computation proceeds in a similar way was in (2):
   * $\mathcal{C}'_6 = (0, 1, 0, i_2, a + d)$.
   * $\mathcal{C}'_7 = (0, 0, 2, i_3, a + d)$.
   * $\mathcal{C}'_8 = (0, 0, 1, i_2, a + 2d)$.
   * $\mathcal{C}'_9 = (0, 0, 1, i_4, a + 2d)$.

b. If the instruction activation signal transfers along the path $i_4 \to i_6$, then $\mathcal{C}''_5 = (0, 0, 1, i_6, a + d)$. As *out-degree*$(i_6) = 0$, no next instruction exists receiving the activation signal, so the next configuration is $\mathcal{C}''_6 = (0, 0, 1, \#, a + d)$, a halting configuration. The system has computed the number $a + d$.

In general at step $5 \cdot k$, for each $k \geq 1$, we have the following:

- $\mathcal{C}'_{5k-1} = (0, 0, 1, i_4, a + k \cdot d)$.
- $\mathcal{C}'_{5k} = (0, 0, 1, i_5, a + k \cdot d)$ or
  $\mathcal{C}''_{5k} = (0, 0, 1, i_6, a + k \cdot d)$.
- $\mathcal{C}'_{5k+1} = (0, 1, 0, i_2, a + k \cdot d)$ or
  $\mathcal{C}''_{5k+1} = (0, 0, 1, \#, a + k \cdot d)$
  (halting computation, the number computed is $a + k \cdot d$).
- $\mathcal{C}'_{5k+2} = (0, 0, 2, i_3, a + k \cdot d)$.
- $\mathcal{C}'_{5k+3} = (0, 0, 1, i_2, a + (k + 1) \cdot d)$.
- $\mathcal{C}'_{5k+4} = (0, 0, 1, i_4, a + (k + 1) \cdot d)$.

Therefore, the virus machine presented in Fig. 6 computes the arithmetic progression having first term a and common difference d, $\{a + k \cdot d \mid k \geq 1\}$. Besides, the number of viruses in each host during every instant of any computation is less than or equal to 2. □

Proof of Lemma 3. Let Π_1, Π_2 be two virus machines in $NVM(p, q, n)$ generating sets of natural numbers A_1, A_2, respectively. We construct a new virus machine Π as presented in Fig. 7, where one more instruction is added for an auxiliary purpose.

The computation of the system starts with the activation of instruction i_1. In the next step, instruction $i_1^{(1)}$ or $i_1^{(2)}$ will be selected in a non-deterministic way. If instruction $i_1^{(1)}$ is selected, then the subsystem Π_1 is enabled and, when this subsystem halts, the result computed by the subsystem Π_1 will be sent to the environment. If instruction $i_1^{(2)}$ is selected, then the subsystem Π_2 is enabled and, when this subsystem halts, the result computed by the subsystem Π_2 will be sent

to the environment. Therefore, a number is computed by the system presented in Fig. 7 if and only if that number belongs to the set $A_1 \cup A_2$. □

Note that Lemma 3 holds true for arbitrary systems, not only for the bounded ones.

7. Conclusions and discussions

In this work, a new computational model called *Virus Machine*, based on the controlled transmissions and replications of viruses, has been introduced. The dynamics of this system along time is inspired by the manner in which viruses transmit between hosts and replicate themselves. A virus machine can be represented as a heterogeneous network with three subnetworks (virus transmission, instruction transfer, and instruction-channel control networks). The computational completeness of these systems has been studied. Specifically, it has been shown that they are equivalent in power to Turing machines when no restriction is imposed over the number of hosts, instructions or viruses present in each host at any instant of the computation. Nevertheless, if the number of viruses present in each host during any computation is limited, then these systems characterize the family of semi-linear sets of natural numbers.

To our knowledge, this work is a pioneer in this area. We conclude by proposing new open problems and lines of research in this computational paradigm.

First of all, considering the usual small universal register machines [9] studied by Korec (with 8 registers and 23 instructions), one can immediately give an upper bound to the number of hosts and instructions required to compute non-semilinear sets of numbers. The question is, what is the lower bound?

Then, it is interesting to explore the ability of VMs to provide solutions, being efficient in some sense, for computationally hard problems. In particular, it would be important to be able to set frontiers of the tractability of problems in terms of syntactical ingredients of Virus Machines.

Thirdly, viruses, as we know, have various characteristics in reality. Some features were used in the VM model, while others were not, such as mutation. Taking more features of viruses into consideration may lead to new variants of VMs, and would be worth being discussed.

Last but not least, it would be interesting to incorporate the parallelism in virus machines through different possible mechanisms.

- In this work, it is assumed that each instruction is attached to at most one transmission channel. We might consider the case in which an instruction could be attached to many different channels. In that case, when the instruction was activated, there would be two possibilities: (a) only one channel might be chosen, in a non-deterministic way, to be opened; (b) all channels connected with the instruction might be opened.
- In our approach, only one instruction is enabled in each computation step. A new variant may be considered as follows: in each step, a nonempty set of control instructions could be chosen, so that all the instructions from this set would be activated simultaneously, and then many channels could be opened at that instant. Then, viruses from different hosts can be simultaneously transmitted.

Acknowledgements

The work is supported by National Natural Science Foundation of China (Grant nos. 61472333, 61370010, 61272152, 61202011, 51405408 and 71103154), Ph.D. Programs Foundation of Ministry of Education of China (20120121120039).

References

[1] Leonard M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (5187) (1994) 1021–1024.
[2] John Aycock, Computer Viruses and Malware, Springer, 2006.
[3] Yaakov Benenson, Tamar Paz-Elizur, Rivka Adar, Ehud Keinan, Zvi Livneh, Ehud Shapiro, Programmable and autonomous computing machine made of biomolecules, Nature 414 (6862) (2001) 430–434.
[4] Francesco Bernardini, Marian Gheorghe, Cell communication in tissue P systems: universality results, Soft Comput. 9 (9) (2005) 640–649.
[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, An Introduction to Algorithms, The MIT Press, Cambridge, Massachusetts, 1994.
[6] Jürgen Dassow, Gheorghe Păun, On the power of membrane computing, J. UCS 5 (2) (1999) 33–49.
[7] Nigel J. Dimmock, Andrew J. Easton, Keith Leppard, Introduction to Modern Virology, Blackwell Pub., Malden, MA (USA), 2007.
[8] Mihai Ionescu, Gheorghe Păun, Takashi Yokomori, Spiking neural P systems, Fund. Inform. 71 (2) (2006) 279–308.
[9] Ivan Korec, Small universal register machines, Theoret. Comput. Sci. 168 (2) (1996) 267–301.
[10] Lila Kari, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, Sheng Yu, DNA computing, sticker systems, and universality, Acta Inform. 35 (5) (1998) 401–420.
[11] Xiangrong Liu, Zimin Li, Juan Liu, Logan Liu, Xiangxiang Zeng, Implementation of arithmetic operations with time-free spiking neural P systems, IEEE Trans. NanoBiosci. 14 (6) (2015) 617–624.
[12] Xiangrong Liu, Juan Suo, Stephen C.H. Leung, Juan Liu, Xiangxiang Zeng, The power of time-free tissue P systems: attacking NP-complete problems, NeuroComputing 159 (2015) 151–156.
[13] Marvin L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., 1967.
[14] Yunyun Niu, Linqiang Pan, Mario J. Pérez-Jiménez, Miquel Rius Font, A tissue P systems based uniform solution to tripartite matching problem, Fund. Inform. 109 (2) (2011) 179–188.

[15] Leandro Nunes de Castro, Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications, CRC, 2006.
[16] Linqiang Pan, Carlos Martín-Vide, Solving multidimensional 0–1 knapsack problem by P systems with input and active membranes, J. Parallel Distrib. Comput. 65 (12) (2005) 1578–1584.
[17] Linqiang Pan, Mario J. Pérez-Jiménez, Computational complexity of tissue-like P systems, J. Complexity 26 (3) (2010) 296–315.
[18] Gheorghe Păun, DNA computing based on splicing: universality results, Theoret. Comput. Sci. 231 (2) (2000) 275–296.
[19] Gheorghe Păun, Computing with membranes, J. Comput. System Sci. 61 (1) (2000) 108–143.
[20] Grzegorz Rozenberg, Thomas Bäck, Joost N. Kok, Handbook of Natural Computing, Springer, 2012.
[21] Xiangxiang Zeng, Linqiang Pan, Mario J. Pérez-Jiménez, Small universal simple spiking neural P systems with weights, Sci. China Inf. Sci. 57 (9) (2014) 1–11.
[22] Xiangxiang Zeng, Lei Xu, Xiangrong Liu, Linqiang Pan, On languages generated by spiking neural P systems with weights, Inform. Sci. 278 (2014) 423–433.
[23] Xiangxiang Zeng, Xingyi Zhang, Tao Song, Linqiang Pan, Spiking neural P systems with thresholds, Neural Comput. 26 (7) (2014) 1340–1361.
[24] Xingyi Zhang, Linqiang Pan, Andrei Păun, On universality of axon P systems, IEEE Trans. Neural Netw. Learn. Syst. 26 (11) (2015) 2816–2829.
[25] Xingyi Zhang, Ye Tian, Yaochu Jin, A knee point driven evolutionary algorithm for many-objective optimization, IEEE Trans. Evol. Comput. 19 (6) (2015) 761–776.
[26] Xingyi Zhang, Ye Tian, Ran Cheng, Yaochu Jin, An efficient approach to non-dominated sorting for evolutionary multi-objective optimization, IEEE Trans. Evol. Comput. 19 (2) (2015) 201–213.
[27] Xingyi Zhang, Xiangxiang Zeng, Bin Luo, Linqiang Pan, On some classes of sequential spiking neural P systems, Neural Comput. 26 (5) (2014) 974–997.