Virus Machines: To tree or not to tree

Antonio Ramírez-de-Arellano[1,2], Francis George C. Cabarle[1,2,3], David Orellana-Martín[1,2], Henry N. Adorna[3], Mario J. Pérez-Jiménez[1,2]

[1]Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[2]SCORE lab, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: aramirezdearellano@us.es, dorellana@us.es, fcabarle@us.es, marper@us.es
[3]Department of Computer Science, University of the Philippines Diliman, 1101 Quezon City, Philippines
E-mail: fccabarle@up.edu.ph, hnadorna@up.edu.ph

Summary. In the present work we further study the computing power of virus machines, or VMs in short. VMs are computing models inspired by the transmission networks of viruses. VMs consist of hosts that contain zero or more virus objects, and an instruction graph that controls the transmissions of virus objects among hosts. The present work improves the understanding of the computing power of VMs by introducing normal forms. Normal forms restrict the features or the number of such features in a given computing model. For VMs we restrict in our normal forms the features such as the number of hosts, number of instructions, and the number of virus objects in each host. After we recall some known results on the computing power of VMs we give our normal forms. For instance we show characterisations from previous inclusions regarding the computation of finite sets of numbers. We also show new characterisations and normal forms for singleton sets and finite sets. Another result using a new normal form are characterisations when the instruction graphs of VMs are (not) restricted to tree graphs. New characterisations of finite sets from VMs with tree instruction graphs are provided, with some conjectures or open problems.

Keywords: Virus machines, Computational power, Natural computing, normal forms.

1 Introduction

In the present work, we consider some normal forms for virus machines, in short, VMs. Virus machines introduced in [1] are unconventional and natural computing models inspired by networks of virus transmissions. More information on unconventional and natural computing is found in [2] and [3], respectively. From [1, 4]

it is shown that VMs are Turing complete, that is, they are algorithms capable of general purpose computations. From such works it is also shown some VMs for computing specific classes of (in)finite sets of numbers.

Virus machines consists of three *subgraphs*: a directed and weighted *host graph* with nodes and edges referred to as *hosts* and *channels*, respectively; a directed and weighted *instruction graph* where nodes are *instructions* and edge weights determine which instruction to prioritise and next activate; an *instruction-channel* graph which connects an edge between instructions and channels in the previous graphs. Hosts contain zero or more virus objects, and activating an instruction means opening a channel since channels are closed by default. Opening a channel means virus objects from one host are transferred to another host.

Briefly, the idea of a normal form for some computing model is to consider restrictions in the model while maintaining its computer power. That is, the consideration of lower bounds for ingredients of a computing model is a natural direction for investigation. For instance a well-known normal form in language theory is the Chomsky normal form, CNF in short, from [5]. Instead of having an infinite number of forms to write rules in a grammar for context-free sets, CNF shows that two forms are enough. Normal forms in unconventional and bio-inspired models include [6], with recent and optimal results in [7], a bibliography in [8], and a recent survey in [9].

The present work contributes the following to the study of virus machines and their computing power. Some normal forms for VMs are provided, such as: providing characterisations (previously were inclusions) for generating families of finite sets; showing new characterisations for finite sets of numbers using restrictions on the number of required hosts, instructions, or viruses; new characterisations are also given for singleton sets of numbers. We also consider a new restriction: limiting or not limiting the instruction graph to be a tree graph, that is, an acylcic graph. We show for instance that some VMs with a tree instruction graph and with some lower bounds on the number of hosts, instructions, and viruses can only compute finite sets. Our results on normal forms are then used to ask new questions regarding other normal forms and restrictions on VMs.

The organisation of the present work is as follows. In Section 2 we recall in a brief the features of VMs used previously in investigating their computing power. Section 2.1 recalls some known results, while Section 2.2 provides new results concerning normal forms of VMs, specifically for computing finite and singleton sets. Section 3 provides new normal forms, for instance, when the instruction graph is restricted to a tree. Lastly, Section 4 provides conclusions, conjectures, and directions for further work.

2 VM with old ingredients

In this section, the computational power of virus machines in generating mode is discussed, from previous works related to some novel results. In this work the syntax and semantics, in addition to a simple explicit example, have been included in

the other work of this volume related to virus machines [10] For further knowledge about the power of virus machines in generating sets we refer to [11, 4, 1]. First, let us fix some notation.

Let $NVM(p, q, n)$ be the family of sets of natural numbers generated by virus machines with at most p hosts, q instructions, and n viruses in each host at any instant of the computation. For unbounded restrictions, they are replaced by a $*$.

2.1 Old results

This subsection is devoted to reviewing results prior to this work regarding the computing power of VMs with respect to certain classes or families of computable numbers.

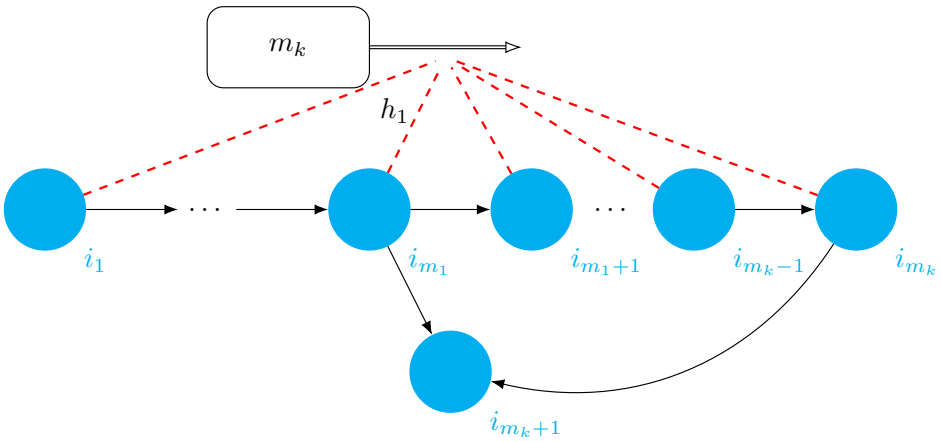

Fig. 1. A virus machine generating $NFIN$ for $NVM(1, *, *)$.

The state-of-the-art is presented in Table 1. The virus machines in the generating mode are Turing Universal; that is, they can generate recursively enumerable sets of numbers (NRE) [1] for unbounded restrictions. This power is severely reduced when the last ingredient is reduced; more precisely, a characterization of semilinear sets ($SLIN$) is proved for $NVM(*, *, 2)$ [11]. From now on, not characterizations but contentions have been proven, for finite sets ($NFIN$) they are contained in $NVM(1, *, *)$ and $NVM(*, *, 1)$ [4]. Finally, the set of power of two numbers is contained in $NVM(2, 7, *)$ [4].

An interesting and natural question is can we further restrict or provide better lower bounds, for known results about VMs? That is, provide "better" characterisations of finite sets or even other families of sets such as the singleton sets, see for instance Table 1. As we focus on finite sets later, let us see the VMs used in [4] to generate finite sets. For $NVM(1, *, *)$ the VM presented in Figure 1, and for

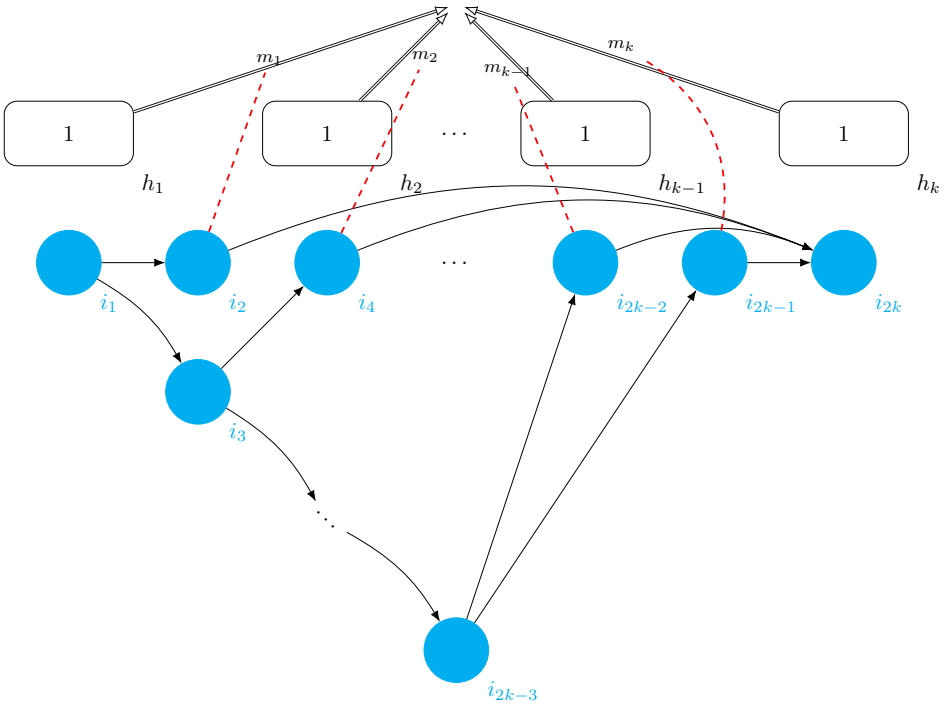

Fig. 2. A virus machine generating $NFIN$ for $NVM(*,*,1)$.

Family of sets	Relation	Hosts	Instructions	Viruses
NRE [1]	$=$	*	*	*
$SLIN$ [11]	$=$	*	*	2
$NFIN$ [4]	$\subset$	1	*	*
	$\subset$	*	*	1
$\{2^n \mid n \geq 0\}$ [4]	$\subset$	2	7	*

Table 1. Previous results: Minimum resources needed for generating family subsets of natural numbers.

$NVM(*,*,1)$ the Figure 2. The corresponding lemmas were called (viruses) and (hosts) respectively, and we follow the same notation in this work.

2.2 New results

Finite sets

Having shown the generation of finite sets by a family of virus machines in the previous section, the bounded ingredient was only one, let us see a family of virus machines with more than one bounded ingredient.

Lemma 1 (Viruses-host). *Let $F = \{m_1, \ldots, m_k\}$ a finite set of natural numbers greater than zero. Then F can be generated by a virus machine of 2 hosts, $2k + 1$ instructions, and the 2 virus in each host at most.*

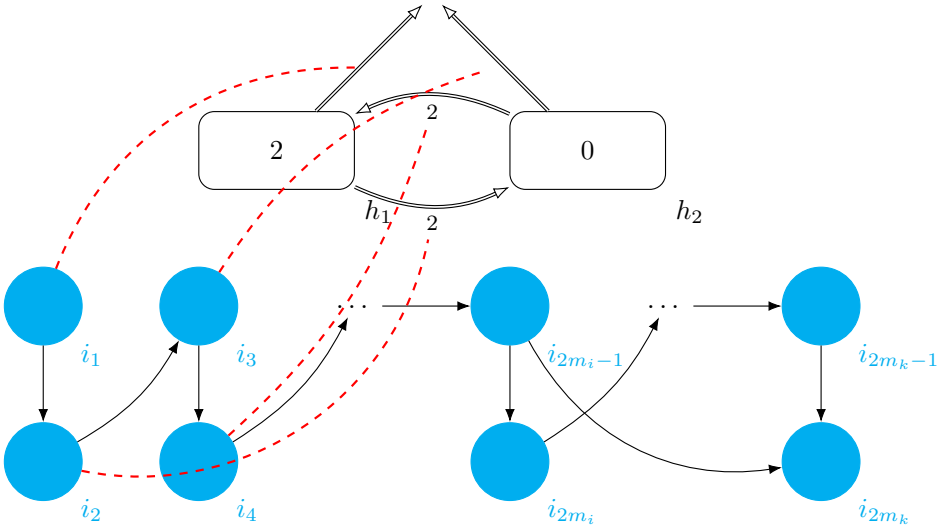

Fig. 3. Virus machine generating the finite set $F = \{m_1, \ldots, m_k\}$.

Proof. Let $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, n_2, \ldots, n_k, i_1, h_{out})$, where:

1. $\Gamma = \{v\}$;
2. $H = \{h_1, h_2\}$;
3. $I = \{i_1, \ldots, i_{2_{m_k}}\}$;
4. $D_H = (H \cup \{h_{out}\}, \{(h_1, h_2), (h_1, h_{out}), (h_2, h_1), (h_2, h_{out})\}, w_H)$, where $w_H((h_1, h_2)) = w_H((h_2, h_1)) = 2$ and $w_H((h_1, h_{out})) = w_H((h_2, h_{out})) = 1$;
5. $D_I = (I, E_I, w_I)$, where $E_I = \{(i_a, i_{a+1}) \mid a \in \{1, \ldots, 2m_k - 1\}\} \cup \{(i_{2m_i-1}, i_{2m_k}) \mid m_i \in F\}$, $w_I((i_j, i_{j'})) = 1 \; \forall (i_j, i_{j'}) \in E_I$;
6. $G_C = (I \cup E_H, E_C)$, where $E_C = \{\{i_{2j+1}, (h_1, h_{out})\}, \{i_{2j}, (h_1, h_2)\} \mid j \in \{0, \ldots, m_k\}, j \text{ even }\} \cup \{\{i_{2j+1}, (h_2, h_{out})\}, \{i_{2j}, (h_2, h_1)\} \mid j \in \{0, \ldots, m_k\}, j \text{ odd }\}$;
7. $n_1 = 2$ and $n_2 = 0$;
8. $h_{out} = h_0$

A visual representation of this virus machine can be found in Figure 3. Let us prove that for each $m_i \in F$, there exists a computation of Π such that it produces m_i viruses in the environment in the halting configuration. Let m_i be the generated number; the following invariant holds:

$$\varphi(x) \equiv \begin{cases} C_{2x} = (2, 0, i_{2x+1}, x) & x \text{ even}, \\ c_{2x} = (0, 2, i_{2x+1}, x) & x \text{ odd}, \end{cases}$$

for each $0 \leq x \leq m_i - 1$. In particular, $\varphi(m_i - 1)$ is true, let us suppose that m_i is odd, then the following computation is verified:

$$\begin{aligned} \mathcal{C}_{2(m_i-1)} &= (2, 0, i_{2(m_i-1)}, m_i - 1), \\ \mathcal{C}_{2m_i} &= (1, 0, i_{2m_k}, m_i), \\ \mathcal{C}_{2m_i+1} &= (1, 0, \#, m_i), \end{aligned}$$

For m_i even the computation is analogous, hence the computation halts in $2m_i + 1$ steps and the number generated is m_i.

Another interesting result is that this inclusion is strict.

Proposition 1. $NFIN \subsetneq NVM(2, *, 2)$.

Proof. Inclusion is direct by the Lemma 1. Let us focus now on the inequality; for that, we construct a virus machine from [10] that generates the set of all natural numbers except the zero, which verifies the restrictions of the proposition.

Let $\Pi_{Nat} = (\Gamma, H, I, D_H, D_I, G_C, 1, 0, i_1, h_{out})$, where:

1. $\Gamma = \{v\}$;
2. $H = \{h_1, h_2\}$;
3. $I = \{i_1, \ldots, i_4\}$;
4. $D_H = (H \cup \{h_{out}\}, \{(h_1, h_2), (h_2, h_{out}), (h_2, h_1)\}, w_H)$, where $w_H((h_1, h_2)) = 2$ and $w_H((h_2, h_{out})) = w_H((h_2, h_1)) = 1$;
5. $D_I = (I, E_I, w_I)$, where $E_I = \{(i_1, i_2), (i_2, _3), (i_3, i_1), (i_3, i_4)\}$, $w_I((i_j, i_{j'})) = 1\ \forall (i_j, i_{j'}) \in E_I$;
6. $G_C = (I \cup E_H, E_C)$, where $E_C = \{\{i_1, (h_1, h_2)\}, \{i_2, (h_2, h_1)\}, \{i_3, (h_2, h_{out})\}\}$;
7. $h_{out} = h_0$;

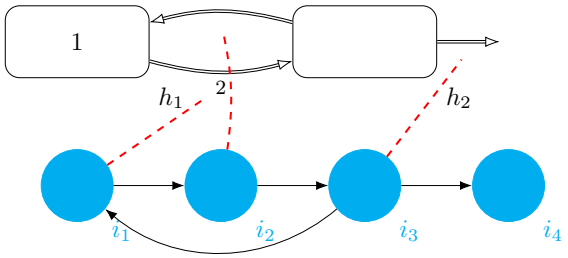

Fig. 4. Virus machine generating the set of natural numbers $\mathbb{N} \setminus \{0\}$.

A visual representation of this virus machine can be found in Fig. 4. Now, let us prove that for each $n \in \mathbb{N}$, there exists a halting computation generating the number n. For generating this number, the following invariant holds:

$$\varphi(k) \equiv \mathcal{C}_{3k} = (1, 0, i_1, k), \text{ for each } 0 \leq k \leq n-1$$

In particular, $\varphi(n-1)$ is true, then the following configuration is verified $\mathcal{C}_{3(n-1)} = (1, 0, i_1, n-1)$, from here, after the 4 transition steps the halting configuration is reached $\mathcal{C}_{3n+1} = (1, 0, \#, n)$, whose output is the natural number n.

With this proposition a new question arises: can we get not only the inclusion but the characterization of the finite sets by a family of virus machines? This is answered in the next section.

Singleton sets

Now let us move to the second family of sets, the Singleton sets, these are sets of natural numbers with only one element, in this work we include the empty set in this family.

Theorem 1. *The following sets of numbers are equivalent to singleton sets:*

1. $NVM(1, *, 1)$;
2. $NVM(*, 1, *)$.

Proof. The proof of equivalence is done by the double inclusion technique.

1. Let us start with the left side inclusion, let $\Gamma = \{v\}$ be a singleton set of natural number $v \in \mathbb{N}$, then it can be generated by the VM Π_{sing_1} of degree $(1, 1)$ depicted in Figure 5, the initial configuration is $\mathcal{C}_0 = (1, i_1, 0)$ and in the following configuration, one virus is consumed and replicated by the weight of the arc, that is v, and sent to the environment, leading to the halting configuration $\mathcal{C}_1 = (0, \#, v)$. Thus, after one transition step, the set generated is $\{v\}$.
   For the reverse inclusion, suppose any VM with only one host and one virus: the host can only be attached to the environment, and let us fix that the weight of that channel is $w \in \mathbb{N}$. Thus, the only number generated is w or none, depending on the instruction graph (if the computation halts or not). Thus, we generate a singleton set.
2. For the inclusion on the left side we can use the VM Π_{sing_1} depicted in Figure 5 as it only has one instruction and the inclusion has already been proven.
   Let us focus on the inclusion of the right side. With only one instruction, there are two possibilities in the instruction graph:
   - The node with a self-arc, which creates an infinite loop, thus a non-halting computation and generating the empty set.

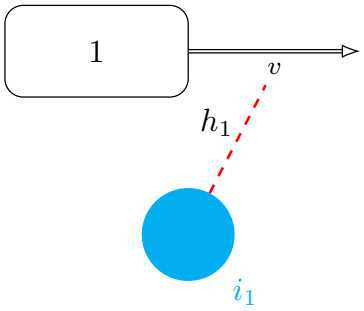

Fig. 5. The VM Π_{sing_1} generating the singleton set $\{v\}$.

- The node with no arcs, thus the machine, halts after only one transition step as there is no other possible path. In this sense, two options can be separated:
  - The instruction is attached to a channel which is attached to the environment,
  - The instruction is not attached to a channel which is attached to the environment, thus the number generated is 0.

3 To tree or not to tree

Until now, the computational power of virus machines was studied by bounding/unbounding the three main ingredients: hosts, instructions, and the number of viruses at any time of computation. Nevertheless, virus machines are heterogeneous networks divided by three graphs, thus more restrictions can be studied, for example, the kinds of graphs that form the instruction graph.

In this section, a novel and interesting scope is proposed to discuss the computational power of virus machines: the *instruction graph properties*.

For this study, several notation and clarifications must first be presented.

Definition 1. *A path in a directed graph $G = (V, E)$ is a sequence of edges $(e_1, \ldots, e_{n-1})$, for which there is a sequence of vertices $(v_1, \ldots, v_n)$, such that $e_i = (v_i, v_{i+1})$, for each $i = 1, \ldots, n-1$, and $v_i \neq v_j$, for all $i, j = 1, \ldots, n$. Under the same conditions, if $(v_n, v_1) \in E$, then the path is called a cycle. A graph without cycles is called a tree. The depth of a tree is the longest path of the tree.*

We say that v_1 is connected to v_n if there is a path $w = (e_1, \ldots, e_{n-1})$, whose sequence of vertices is $(v_1, \ldots, v_n)$. We denote by $V(v_i) \subseteq V$ the subset of vertices that are connected by a path from v_i.

A graph $G = (V, E)$ is connected if there are paths that contain each pair of vertices. A connected component of the graph G is a subgraph graph $G' = (V', E')$, such that $V' \subseteq V$, $E' \subseteq E$ where $(v_i, v_j) \in E'$ if and only if $(v_i, v_j) \in E$, and $v_i, v_j \in V'$ are connected.

Proposition 2 (Invariance). *If the instruction graph D_I of a virus machine Π of degree (p, q), with $p, q \geq 1$,*

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

is not connected, then there exists another virus machine Π' of degree (p, q'), with $q' \leq q$, which has the same computation.

Proof. Let Π be the virus machine fixed in the statement, setting the instruction graph to $D_I = (I, E_I, w_I)$, as it is not connected; then $I(i_1) \neq I$. Let Π' be the virus machine of degree $(p, q') = |I(i_1)|)$, defined as Π but with a new instruction graph $D_{I(i_1)} = (I(i_1), E_{I(i_1)}, w_{I(i_1)})$.

Due to the semantics associated with virus machines, any instruction that can be activated must be connected by a path from the initial instruction; thus, the set of instructions of Π that can be activated at some instant of the computation is contained in $I(i_1)$, therefore Π' has the same computation.

Using this result, from now on, all the virus machines defined are supposed to have a connected instruction graph, with the connected component $I(i_1)$, being i_1 the initial instruction. In addition, the same notation of the components of a virus machine Π is used for the following results.

3.1 Instruction graph as a tree

Proposition 3. *If the instruction graph is a tree, then all computations halt. In addition, the number of transition steps is bounded by the depth of the tree.*

Having this restriction on the instruction graph is a limitation of the power of these devices. More precisely, we lose Turing universality; let us see a characterization of finite sets of natural numbers with this restriction. First, let us fix some notation:

Let $NVM_{tree}(p, q, n)$ be the family of sets of natural numbers that can be generated by a virus machine with a tree instruction graph, at most p hosts, q instructions and n virus in each host at any instant of computation. In case there is no restriction, it is written as $*$. Let $NFIN$ be the family of finite sets of natural numbers.

The following results hold:

Corollary 1. $NVM_{tree}(*, *, *) \subseteq NFIN$

Proof. The inclusion $NVM_{tree}(*, *, *) \subseteq NFIN$ is direct by the Proposition 3, as all computations halt, then every virus machine halts in a finite number of steps, thus the set of numbers that can be generated is finite.

Theorem 2. $NVM_{tree}(p, *, *) = NFIN$, *for each* $p \geq 1$.

Proof. The left inclusion $NVM_{tree}(*,*,*) \subseteq NFIN$ is direct from Corollary 1.

The right inclusion $NFIN \subseteq NVM_{tree}(p,*,*)$, for each $p \geq 1$, is proved in Lemma 1 (host) of the work [4], where the virus machine presented was Figure 1, which has the instruction graph as a tree.

Corollary 2. $NVM_{tree}(*,*,*) = NFIN$.

Proof. Direct from Corollary 1 and Theorem 2.

Theorem 3. $NVM_{tree}(*,*,n) = NFIN$, *for each* $n \geq 1$.

Proof. Again, the left inclusion $NVM_{tree}(*,*,*) \subseteq NFIN$ is directly related to the Corollary 1.

The right inclusion $NFIN \subseteq NVM_{tree}(*,*,n)$, for each $n \geq 1$ is proved in Lemma 2 (viruses) of the work [4], where a virus machine with an instruction graph as a tree was presented, generating finite number sets.

Another interesting result occurs if we also bound the amount of hosts:

Theorem 4. $NVM_{tree}(p,*,n) = NFIN$, *for each* $p \geq 2$, *and* $n \geq 2$.

Proof. The right-hand inclusion is by applying again the Corollary 1. For the left side, we can use the Lemma 1 where the VM presented in Figure 3 has the instruction graph as a tree, therefore, $NVM_{tree}(p,*,n) \subseteq NFIN$.

Family of sets	Symbol	Hosts	Instructions	Viruses
$NFIN$ (Corollary 2)	=	*	*	*
(Theorem 2)	=	1	*	*
(Theorem 3)	=	*	*	1
(Theorem 4)	=	2	*	2

Table 2. Minimum resources needed for generating/characterizing family subsets of natural numbers with the *instruction graph as a tree*.

We summarise our main results so far in Table 3.

4 Conjectures and conclusions

In the present work we considered normal forms for VMs: first, by summarising known results for some families of number sets (see Table 1); next, by providing some new results and showing strict characterisations from previous inclusions (see Table 2). We summarise our results and ask new questions for our sequel works regarding normal forms of VMs in Table 3. The rows marked with "?" in Table 3 are open questions, such as if $NVM_{tree}(*,2,*)$ is a strict superset of $NFIN$. It

is also interesting to consider "new" normal forms, such as the graph properties of the host graph and the instruction-channel graph. For instance, considering the weights of the host graph, or if a bijection exists between instructions and channels.

One reason for the interest in normal forms is the consideration of "jumps" in computing power from one family of computable sets to another. For instance in Table 1 we know that VMs with unbounded number of hosts, instructions, and viruses are Turing machines, that is, they are general purpose computers. In Table 3 we see that if we are allowed only one host and one virus in a VM, with an arbitrary number of instructions (the graph is not a tree) then the computing power has a significant jump down to singleton sets. Besides realising such jumps, it is also interesting to realise frontiers or thresholds of the ingredients between families of sets.

Another interesting direction is to consider "small" VMs in the sense of [12, 13]. That is, give lower bounds to the number of instructions or hosts required to maintain a certain computing power, in fact, authors in [14] constructed a small universal VM using 9 hosts and 33 instructions. It would be interesting to study also the lower bound number to compute $NFIN, SLIN$. From Table 1 for instance it is interesting to give better lower bounds for characterisations of NRE and $SLIN$. At least for VMs with instruction graph as a tree, Table 2 provides better lower bounds.

In [15] and its sequel [10] a matrix representation is given for VMs. It is interesting to consider properties in the present work, such as lowering the values for ingredients or restriction to a tree graph, using such a representation. Another direction is to consider the results and conjectures in the present work for VMs in the accepting and function computing modes as in [4], or with parallel VMs as in [16, 17].

Family of sets	Symbol	Hosts	Instructions	Viruses	Tree Inst. Graph
Singleton (Theorem 1)	$=$	1	*	1	No
(Theorem 1)	$=$	*	*	1	No
(Theorem 1)	$=$	*	1	*	No
$NFIN$ (Theorem 2)	$=$	1	*	*	Yes
(Theorem 3)	$=$	*	*	1	Yes
(Theorem 4)	$=$	2	*	2	Yes
(Proposition 1)	$\subsetneq$	2	*	2	**No**
	$\subsetneq$?	*	2	*	Yes
	$\subseteq$?	*	3	*	Yes

Table 3. Summary of the minimum resources needed for generating/characterizing family subsets of natural numbers.

Acknowledgements

F.G.C. Cabarle is supported by the QUAL21 008 USE project, "Plan Andaluz de Investigación, Desarrollo e Innovación" (PAIDI) 2020 and "Fondo Europeo de Desarrollo Regional" (FEDER) of the European Union, 2014-2020 funds. A. Ramírez-de-Arellano is supported by the Zhejiang Lab BioBit Program (Grant No. 2022BCF05).

References

1. Valencia-Cabrera, L., Pérez-Jiménez, M.J., Chen, X., Wang, B., Zeng, X.: Basic virus machines. In: 16th International Conference on Membrane Computing (CMC16). (2015) 323–342
2. Adamatzky, A.: Handbook Of Unconventional Computing (In 2 Volumes). World Scientific (2021)
3. Bäck, T., Kok, J.N., Rozenberg, G.: Handbook of natural computing. Springer, Heidelberg (2012)
4. Ramírez-de-Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Generating, computing and recognizing with virus machines. Theoretical Computer Science **972** (07 2023) 114077
5. Chomsky, N.: On certain formal properties of grammars. Information and control **2**(2) (1959) 137–167
6. Ibarra, O.H., Păun, A., Păun, G., Rodríguez-Patón, A., Sosík, P., Woodworth, S.: Normal forms for spiking neural p systems. Theoretical Computer Science **372**(2-3) (2007) 196–217
7. Macababayao, I.C.H., Cabarle, F.G.C., de la Cruz, R.T.A., Zeng, X.: Normal forms for spiking neural p systems and some of its variants. Information Sciences **595** (2022) 344–363
8. Cabarle, F.G.C., Dela Cruz, R.T.A.: A bibliography of normal forms in spiking neural P systems and variants. In: Bulletin of the International Membrane Computing Society. Volume 12. (12 2021) 89–91
9. Cabarle, F.G.C.: Thinking about spiking neural p systems: some theories, tools, and research topics. Journal of Membrane Computing (2024) 1–20
10. Ramírez-de-Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Pérez-Jiménez, M.J., Adorna, H.N.: Virus machines and their matrix representations. 20th Brainstorming Week on Membrane Computing and First Workshop on Virus Machines, 24–26 January 2024, Sevilla, Spain (2024)
11. Chen, X., Pérez-Jiménez, M.J., Valencia-Cabrera, L., Wang, B., Zeng, X.: Computing with viruses. Theoretical Computer Science **623** (2016) 146–159
12. Cabarle, F.G.C., de la Cruz, R.T.A., Adorna, H.N., Dimaano, M.D., Peña, F.T., Zeng, X.: Small spiking neural p systems with structural plasticity. Enjoying Natural Computing: Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday (2018) 45–56
13. Păun, A., Păun, G.: Small universal spiking neural p systems. BioSystems **90**(1) (2007) 48–60
14. Ramírez-de Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Using virus machines to compute pairing functions. International Journal of Neural Systems **33**(05) (2023) 2350023

15. Ramírez-de Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Pérez-Jiménez, M.J., Adorna, H.N.: Matrix representation of virus machines. In Ferrández Vicente, J.M., Val Calvo, M., Adeli, H., eds.: Bioinspired Systems for Translational Applications: From Robotics to Social Engineering, Cham, Springer Nature Switzerland (2024) 420–429
16. Ramírez-de-Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Parallel virus machines. Submitted to Journal of Membrane Computing
17. Ramírez-de-Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Virus machines at work: Computations of workflow patterns. International Summer Conference (ISC 24) of Decision Science Alliance, 6-7 June 2024 València, Spain