Journal of Membrane Computing (2021) 3:133–148
https://doi.org/10.1007/s41965-021-00075-1

REGULAR PAPER

Spiking neural P systems: matrix representation and formal verification

Marian Gheorghe[1] · Raluca Lefticaru[1] · Savas Konur[1] · Ionuţ Mihai Niculescu[2] · Henry N. Adorna[3]

Received: 19 November 2020 / Accepted: 24 March 2021 / Published online: 27 April 2021

Abstract
Structural and behavioural properties of models are very important in development of complex systems and applications. In this paper, we investigate such properties for some classes of SN P systems. First, a class of SN P systems associated to a set of routing problems are investigated through their matrix representation. This allows to make certain connections amongst some of these problems. Secondly, the behavioural properties of these SN P systems are formally verified through a natural and direct mapping of these models into kP systems which are equipped with adequate formal verification methods and tools. Some examples are used to prove the effectiveness of the verification approach.

Keywords Membrane computing · Spiking neural P systems · Petri nets · Kernel P systems · kPWorkbench · Formal verification

1 Introduction

Membrane computing is a research field initiated by Gh. Păun [48, 49]. This computation paradigm is inspired by the structure and functioning of the living cells. The key models are called *membrane systems* or *P systems*. The field has developed very fast and many classes of membrane systems (P systems) have been investigated. These may be classified as *cell-like*, *tissue-like* and *neural-like* P systems. A presentation of the most important such models is available in [50] and [51].

A class of neural-like P systems, called *Spiking Neural P systems* (*SN P systems*, for short), has been introduced in [30], inspired by the neurophysiological behaviour of neurons (in brain) sending electrical impulses along axons to other neurons. Significant results have been reported in the literature with respect to SN P systems, in particular, generating or recognising elements of a set (see the survey paper [47]).

Matrices have been used to represent SN P systems without delays [55] and later SN P systems with delays [15]. These matrix representations have been useful in implementing SN P systems *in silico* [8, 18, 19, 31]. A thorough presentation on these matrix representations and further research directions have been discussed in [2], including the idea of periodicity in SN P systems (without delays) started by [29]. Some Petri net-like properties of SN P systems are also provided in [2].

Another type of P systems, called *kernel P systems* (*kP systems*, for short), has been introduced [23] in order to capture in a unified approach features of various membrane computing models, making it more amenable for describing various problems, including complex applications. kP systems have an expressive formal language, allowing models to be simulated with a software framework, called kPWorkbench [12], which also includes a verification component

✉ Raluca Lefticaru
r.lefticaru@bradford.ac.uk

Marian Gheorghe
m.gheorghe@bradford.ac.uk

Savas Konur
s.konur@bradford.ac.uk

Ionuţ Mihai Niculescu
ionutmihainiculescu@gmail.com

Henry N. Adorna
hnadorna@dcs.upd.edu.ph

[1] Department of Computer Science, University of Bradford, West Yorkshire, Bradford BD7 1DP, UK

[2] Faculty of Science, University of Piteşti, Str. Târgul din Vale nr. 1, 110040 Piteşti, Romania

[3] Department of Computer Science (Algorithms and Complexity), University of Philippines Diliman, 1101 Quezon City, Philippines

[25]. A thorough presentation of this tool has been made in [42]. The modelling, simulation, verification and testing aspects of kP systems have been presented in [21, 22] and applications in synthetic biology in [24, 40, 41].

In this paper we investigate structural and behavioural properties of some classes of SN P systems. First, the overall structure and configurations of SN P systems associated to a set of routing problems are investigated through their matrix representation. This allows to make certain connections amongst some of these problems. Secondly, the behavioural properties of these SN P systems are simulated and formally verified through a natural and direct mapping of these models into kP systems, which are equipped with adequate formal verification methods and tools. Some examples are used to prove the effectiveness of the verification approach.

This paper is structured as follows: Sect. 2 presents the preliminaries regarding SN P systems and kP systems. Section 3 discusses the matrix representation introduced in [15] for SN P systems for some Petri net-like properties. Some connections between SN P systems and kP systems are presented in Sect. 4. Simulation and formal verification of the kP systems obtained from the SN P systems, by examining two examples, are discussed in Sect. 5. Finally, conclusions are presented in Sect. 6.

2 Preliminaries

This section briefly presents the notations used, then gives the basic definitions regarding *spiking neural P systems* (*SN P systems*, for short) and *kernel P systems* (*kP systems*, for short).

For a finite alphabet $A = \{a_1, ..., a_p\}$, A^* represents the set of all strings (sequences) over A. The empty string is denoted by λ and $A^+ = A^* \setminus \{\lambda\}$ denotes the set of non-empty strings.

A multiset over A is a mapping $f : A \rightarrow \mathbb{N}$, represented as a string $a_1^{f(a_1)} \cdots a_p^{f(a_p)}$, where the order is not important, and where elements which are not in the support (i.e., elements a_j, with $1 \le j \le p$, having $f(a_j) = 0$) are omitted. In the sequel, multisets will be represented by such strings.

2.1 Spiking neural P systems

The definition and other concepts related to SN P systems are from [2, 55]. In this paper, we consider only **SN P systems without delay rules**.

Definition 1 An *SN P system* of degree m, $m \ge 1$, is a tuple

$$\Pi = (O, \sigma_1, \dots, \sigma_m, \text{syn}, \text{in}, \text{out}),$$

where

- $O = \{a\}$ is a singleton alphabet (a is called spike);
- $\sigma_i, 1 \le i \le m$, are neurons, $\sigma_i = (n_i, R_i), 1 \le i \le m$, where
  - $n_i \ge 0$ is the number of spikes in σ_i;
  - R_i is a finite set of rules of the following forms:

    (*Type (1); spiking rules*)
    $E/a^c \rightarrow a^p$; where E is a regular expression over $\{a\}$, and $c \ge 1, p \ge 1$, such that $c \ge p$;
    (*Type (2); forgetting rules*)
    $a^s \rightarrow \lambda$, for $s \ge 1$, such that for each rule $E/a^c \rightarrow a^p$ of type (1) from R_i, $a^s \notin L(E)$;
- $\text{syn} = \{(i,j) | 1 \le i,j \le m, i \ne j\}$ (synapses between distinct neurons);
- in, out $\in \{1, \dots .m\}$ indicate the input and output neurons respectively.

Remark 1 The *in* neuron will not be distinguished in what follows and, when specified, *out* will be also not considered.

The SN P system Π computes by applying one rule from each neuron.

A **configuration** of SN P system Π is an m-size vector of integers

$$C = (a_1, a_2, \dots, a_m),$$

where $a_j, 1 \le j \le m$, represents the number of spikes in neuron σ_j.

A **configuration at time** k, $k \ge 0$, of an SN P system Π, as above, is a vector

$$C^{(k)} = (a_1^{(k)}, a_2^{(k)}, \dots, a_m^{(k)}),$$

where $a_j^{(k)} \in \mathbb{Z}^+ \cup \{0\}, 1 \le j \le m$, is the number of spikes present at time k in neuron σ_j.

The vector $C^{(0)} = (a_1^{(0)}, a_2^{(0)}, \dots, a_m^{(0)})$ is the **initial configuration vector** of SN P system Π, where $a_j^{(0)}, 1 \le j \le m$, represents the initial number of spikes, n_j, in neuron σ_j.

We say a rule $r_x \in R_j, 1 \le j \le m$, of neuron σ_j is **applicable** at time k, $k \ge 0$, if and only if the multiset a^n satisfies E_x (or $a^n \in L(E_x)$), where $n = a_j^{(k)}$. At some time k, we can have in neuron σ_j, $a^n \in L(E_x) \cap L(E_y)$, for some rules r_x and r_y, $x \ne y$. In this case one of the two rules will be chosen to be applied. This is how the non-determinism[1] of the system is realised, whereas several neurons with their chosen applicable rules can fire (or spike) simultaneouly at time k, demonstrating parallelism.

[1] In [53] there is a normal form for SN P systems where $L(E_x) \cap L(E_y)$ is either $L(E_x)$ or empty, hence in order to find all non-deterministic rules is sufficient to verify if their regular expressions are the same.

A rule of type 2 from R_j removes spike(s) from the neuron at some time k when applied. Such a rule could only be applied if and only if the number of spikes in σ_j is exactly the amount of spikes it needs to be applied. Formally, if there exists a rule $a^s \rightarrow \lambda \in R_j$, then there cannot exist any rule $E/a^c \rightarrow a^p$ of type 1 in R_j such that $a^s \in L(E)$.

In defining rules in SN P systems, we follow the standard convention of simply not specifying E whenever the left-hand side of the rule is equal to E.

The sequence of configurations defines a **computation** of the system. SN P systems obtain inputs from outside the system (environment) through the designated input neuron(s). We say that a computation **halts** or reaches a **halting configuration** if it reaches a configuration where no more applicable rules are available. We could represent the output of the system as the **number of steps lapse between the first two spikes** of the designated output neuron. Another output representation of SN P system is a sequence of (coded) spikes. This sequence is called a **spike train**, which is a (combinatorial) sequence of spikes and no spikes (silence) made by the systems.

2.2 Kernel P systems

In the following, we will give a formal definition of kP systems; for more details, see [23].

First, we introduce a **compartment type** utilised later in defining the compartments of a kP system.

Definition 2 *T is a set of compartment types, $T = \{t_1, \dots, t_s\}$, where $t_i = (R_i, \delta_i)$, $1 \leq i \leq s$, consists of a set of rules, R_i, and an execution strategy, δ_i, defined over $\mathrm{Lab}(R_i)$, the labels of the rules of R_i.*

The definition of a kP system uses the concept of compartment type.

Definition 3 A *kP system* of degree n, $n \geq 1$, is a tuple

$$k\Pi = (A, \mu, C_1, \dots, C_n, i_0),$$

where

- A is a finite set of elements called objects;
- μ defines the membrane structure, which is a graph, (V, L), where V is a set of vertices representing compartments (or components), and L is a set of edges, i.e., links between compartments;
- $C_i = (t_i, w_{i,0})$, $1 \leq i \leq n$, is a compartment of the system consisting of a *compartment type*, t_i, from a set T, and an initial multiset, $w_{i,0}$, over A; the type $t_i = (R_i, \delta_i)$ consists of a set of evolution rules, R_i, and an execution strategy, δ_i;
- i_0 is the output compartment where the result is obtained.

The kP systems presented in this paper will only use *rewriting and communication* rules. A more general discussion regarding all types of rules of a kP system, including *structure changing* rules, i.e., membrane division, membrane dissolution, link creation and link destruction rules, can be found in [23].

A *rewriting and communication* rule of a compartment C_i, $1 \leq i \leq n$, has the form: $x \rightarrow y\{g\}$, where g represents a **guard** (this will be formally explained in Definition 5), $x \in A^+$ and $y \in (A \times T)^*$, where y is a multiset, $y = (a_1, t_1) \dots (a_h, t_h)$, where $h \geq 0$, and for each $1 \leq j \leq h$, $a_j \in A$ and t_j indicates a compartment type from T. If several compartments, $C_{j_1}, \dots C_{j_p}$, $p > 1$, linked to C_i, have the same compartment type, t_j, then one of them will be non-deterministically chosen to receive a_j.

For the next definitions, we make the following notations. For a multiset w over A and an element $a \in A$, we denote by $|w|_a$ the number of objects a occurring in w. $\mathrm{Rel} = \{<, \leq, =, \neq, \geq, >\}$ denotes the set of relational operators, $\gamma \in \mathrm{Rel}$, a relational operator. We introduce now an *abstract relational expression*.

Definition 4 If g is the *abstract relational expression* denoted by γa^n and w a multiset, then the guard g applied to w denotes the *relational expression* $|w|_a \gamma n$.

The abstract relational expression g is true for the multiset w, if $|w|_a \gamma n$ is true.

We consider now the following Boolean operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction). An *abstract Boolean expression* is defined by one of the following conditions:

- any abstract relational expression is an abstract Boolean expression;
- if g and h are abstract Boolean expressions then $\neg g$, $g \wedge h$ and $g \vee h$ are abstract Boolean expressions.

Definition 5 If g is an *abstract Boolean expression* containing g_i, $1 \leq i \leq q$, abstract relational expressions and w a multiset, then g applied to w means the *Boolean expression* obtained from g by applying g_i to w for any i, $1 \leq i \leq q$.

As in the case of an abstract relational expression, the guard g is true with respect to the multiset w, if the abstract Boolean expression g applied to w is true.

In each compartment type, apart from rules, there is an execution strategy, as mentioned by Definition 2. In this paper we only refer to one execution strategy, namely choice or alternative. For an exhaustive discussion we refer to [23].

Definition 6 For a compartment type $t = (R, \delta)$ from T and $r_1, \dots, r_s \in \text{Lab}(R)$, the *choice (alternative) execution strategy*, δ, is defined by the following notation $\{r_1, \dots, r_s\}$, with the meaning: one of the rules applicable will be non-deterministically chosen and executed; if none is applicable then nothing is executed.

Definition 7 A *configuration* of a kP system, $k\Pi$, with n compartments, is a tuple $c = (c_1, \dots, c_n)$, where $c_i \in A^*$, $1 \le i \le n$, is the multiset from compartment C_i. The *initial configuration* is $(w_1, \dots, w_n)$, where $w_i \in A^*$ is the initial multiset of the compartment $C_i, 1 \le i \le n$.

A *transition* (or *computation step*), introduced by the next definition, is the process of passing from one configuration to another.

Definition 8 Given two configurations $c = (c_1, \dots, c_n)$ and $c' = (c'_1, \dots, c'_n)$ of a kP system, $k\Pi$, with n compartments, and a multiset of rules $M_i, 1 \le i \le n$, applicable to c_i (as δ_i is a choice, M_i consists either of a rule from the compartment C_i or is empty), a *transition* or a *computation step* is the process of obtaining c' from c by using the multisets of rules $M_i, 1 \le i \le n$, denoted by $c \Longrightarrow^{(M_1,\dots,M_n)} c'$. The multiset c'_i, $1 \le i \le n$, is obtained from c_i by removing all the objects that are on the left-hand side of the rule of M_i from c_i and then adding objects a that appear as (a, t_i) on the right-hand side of each rule of $M_j, 1 \le j \le n$, and do not go to other compartments with the same type t_i as C_i.

A **computation** in a P system is a sequence of transitions (computation steps). A configuration is called **final configuration**, if no rule can be applied to it. As usual in P systems, we only consider **terminal computations**, i.e., those arriving in a final configuration. The result of a terminal computation is the **number of objects** appearing in the output compartment of a final configuration.

3 Matrix representation of spiking neural P systems and applications

We express with matrices SN P systems, as suggested in [2, 55]. Structural and behavioural properties of SN P systems considered in [13] are then investigated using matrices.

3.1 Basics on matrix representation of spiking neural P systems

We restrict our SN P systems not to have any neuron with self-loop, that is synapse directly connecting to itself when defining its matrix representation.

Definition 9 (*Spiking transition matrix*) [55] Let Π be an SN P system with the total number of rules n and m neurons. Let the rules in the systems follow some precise ordering. The **spiking transition matrix** of Π is $M_\Pi = [b_{ij}]_{n\times m}$, where

$$b_{ij} = \begin{cases} -c_i, & \text{if the left-hand side rule of } r_i \text{ in } \sigma_j \text{ is } a^{c_i} \\ & (c_i \text{ spikes are consumed}) \\ p_i, & \text{if the right-hand side of the rule } r_i \text{ in } \sigma_s \\ & (s \ne j \text{ and } (s,j) \in syn) \text{ is } a^{p_i} \\ 0, & \text{otherwise} \end{cases}$$

The matrix M_Π is (almost) a natural representation of the SN P system Π. Each row $i, 1 \le i \le n$, corresponds to a rule $r_i : E_i/a^{c_i} \to a^{p_i}$ in some neuron σ_j, with $b_{i,j}, 1 \le j \le m$, defined as above. Each column $j, 1 \le j \le m$, corresponds to a neuron σ_j.

***Example 1* An SN P system for $\mathbb{N} - \{1\}$** [55]. Let $\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \text{syn}, \text{out})$, where $\sigma_1 = (2, R_1)$, with $R_1 = \{a^2/a \to a, a^2 \to a\}$; $\sigma_2 = (1, R_2)$, with $R_2 = \{a \to a\}$; $\sigma_3 = (1, R_3)$, with $R_3 = \{a \to a, a^2 \to \lambda\}$; and $\sigma_4 = (0, \{\})$; $\text{syn} = \{(1,2),(1,3),(2,1),(2,3),(3,4)\}$; $\text{out} = \sigma_4$.

The SN P system in Example 1, generating $\mathbb{N} - \{1\}$, has $n = 5$ rules and $m = 4$ neurons and its spiking transition matrix is $M_\Pi = \begin{pmatrix} -1 & 1 & 1 & 0 \\ -2 & 1 & 1 & 0 \\ 1 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & -2 & 0 \end{pmatrix}$.

Some more details regarding the properties of this matrix, M_Π, are presented in [55]. The **initial configuration** of Π, introduced in Example 1, is $C^{(0)} = (2, 1, 1)$. The **next configuration** $C^{(k+1)}$ to $C^{(k)}$ can be computed using the following identity [55],

$$C^{(k+1)} = C^{(k)} + s^{(k)} \cdot M_\Pi, \tag{1}$$

where $s^{(k)} = (s_1^{(k)}, s_2^{(k)}, \dots, s_n^{(k)}) \in \{0,1\}^n$ is some *valid spiking vector* with respect to $C^{(k)}$ and M_Π is the spiking transition matrix of Π.

A **valid spiking vector** $s^{(k)}$ indicates the rules that could be used at time k with respect to $C^{(k)}$ and the regular expression governing the usability of rules in each neuron. $s_i^{(k)} = 1$ whenever the multiset in σ_j is in $L(E_i)$ of rule r_i in neuron σ_j, else $s_i^{(k)} = 0$. Note that $C^{(k)}$ has non-negative entries. These entries represent only the amount of multisets in each neuron in the system. Checking if the multiset in every neuron belongs to $L(E_i)$, for each i is a must to define a valid spiking vector. If there exist at least two possible rules in a neuron that can be used, then we non-deterministically choose one of them for that time instance.

A **valid configuration** is one that is either an initial configuration or any configuration obtained from a valid configuration using a valid spiking vector or (sum of) sequence of valid spiking vectors [2]. It is not hard to see that

$$C^{(k)} = C^{(0)} + \left(\sum_{i=0}^{k-1} s^{(i)} \right) \cdot M_{\Pi}, \qquad (2)$$

where for each $i, 1 \leq i \leq k-1, s^{(i)}$ is a valid spiking vector.

Any configuration $C^{(k)}$ that satisfies Eq. (2) is called a *reachable configuration.* In particular, we define

$$R(\Pi, C^{(0)}) = \{C^{(k)} \mid C^{(k)} \text{satisfies (2)}\},$$

the **set of all reachable configurations from $C^{(0)}$ in Π**. We call $C^{(k)}$ a reachable configuration of Π from $C^{(0)}$, for each $k \geq 0$.

3.2 Properties of SN P systems

In this section, we look into some structural and behavioural properties of SN P systems [13] which resemble some properties of transition systems such as Petri nets. SN P systems transmit and route spikes in processing information. First, we consider *basic routing blocks,* namely AND-**join**, AND-**split**, OR-**join** and 2-way OR-**split.** Then we focus on *types of routing spikes* via synapses in the systems that can be identified as follows: *sequential, conditional, parallel,* and *iteration routing type.* We will show some relationships amongst some basic routing blocks and some routing types. To demonstrate these routing blocks and routing types, we have SN P systems to emulate the behaviour of spike transmission with respect to the structures of the synapses connecting neurons. In the following illustrations, neurons emit only at most 2 spikes. Boundedness and liveness of SN P systems are consider at the end of this section.

3.2.1 Basic routing blocks

One can check correctness of the matrices below, representing *basic routing blocks* by performing the matrix computation by SN P systems using equation (1) and observing the appropriate entries in the configuration vector that indicate the desired output.

1. **AND-join:** spikes are transmitted from two source neurons to a neuron, which sends the processed accumulated spikes to next neuron connected from it.

   Let the SN P system $\Pi_{\text{AND-join}}$: given by $(\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \text{syn})$, where $\sigma_1 = \sigma_2 = (1, \{a^+/a \rightarrow a\})$, $\sigma_3 = (0, \{(a^2)^+/a^2 \rightarrow a\})$, $\sigma_4 = (0, \{\})$, and syn $= \{(1, 3), (2, 3), (3, 4)\}$. The spiking transition matrix is
$$M_{\Pi_{\text{AND-join}}} = \begin{pmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{with } C^{(0)} = (1, 1, 0, 0).$$

2. **AND-split:** transmits spikes simultaneously from a source neuron to all neurons connected from it.

   Let an SN P system with three neurons given by $(\{a\}, \sigma_1, \sigma_2, \sigma_3, syn)$, where $\sigma_1 = (1, \{a^+/a \rightarrow a\})$, $\sigma_2 = \sigma_3 = (0, \{\})$ and syn $= \{(1, 2), (1, 3)\}$. This can be represented by the following matrix:
$$M_{\Pi_{\text{AND-split}}} = \begin{pmatrix} -1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{with } C^{(0)} = (1, 0, 0).$$

3. **OR-join:** spikes are transmitted from two source neurons directly to a neuron connected from these source neurons.

   Let an SN P system with three neurons given by $(\{a\}, \sigma_1, \sigma_2, \sigma_3, \text{syn})$, where $\sigma_1 = (1, \{a^+/a \rightarrow a\})$, $\sigma_2 = (1, \{a^+/a \rightarrow a\})$, $\sigma_3 = (0, \{\})$, and syn $= \{(1, 3), (2, 3)\}$. This can be represented by the following matrix:
$$M_{\Pi_{\text{OR-join}}} = \begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \text{with } C^{(0)} = (1, 1, 0).$$

4. **2-way OR-split:** the transmitting neuron must decide to which target neuron the spike must be transmitted. This can be represented by $\Pi_{\text{OR-split}} = (\{a\}, \sigma_1, \ldots, \sigma_8, \text{syn})$, where $\sigma_1 = (1, \{a^+/a \rightarrow a\})$, $\sigma_2 = \sigma_3 = (0, \{a \rightarrow a\})$, $\sigma_4 = (0, \{a^2 \rightarrow a^2, a^2 \rightarrow a\})$, $\sigma_5 = (0, \{a^2 \rightarrow \lambda, a \rightarrow a\})$, $\sigma_6 = (0, \{a^2 \rightarrow a, a \rightarrow \lambda\})$, $\sigma_7 = \sigma_8 = (0, \{\})$, and syn $= \{(1, 2), (1, 3), (2, 4), (3, 4), (4, 5), (4, 6), (5, 7), (6, 8)\}$. The spiking matrix representations is:
$$M_{\Pi_{\text{OR-split}}} = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & -2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix}, \text{ with}$$
   $C^{(0)} = (1, 0, 0, 0, 0, 0, 0, 0)$.

   Another solution, with fewer neurons than in the above mentioned SN P system, $\Pi_{\text{OR-split}}$, is obtained by removing the neurons σ_2 and σ_3 with all the synapses coming to and going out of them, and adding a synapse from σ_1 to σ_4 and an initial spike to σ_4. These will be reflected in the spiking matrix representation by removing the rows and columns 2 and 3, and introducing $C^{(0)} = (1, 1, 0, 0, 0, 0)$.

The SN P system $\Pi_{\text{OR-split}}$, as well as the others above, perform one flow of execution, starting with one spike and then moving through synapses up until no rule is executed. One can consider a slight generalisation, whereby the neuron σ_1 will have initially k spikes, $k \geq 1$, and its spiking rule will become $a^+/a \to a$. In this case the flow of spikes from σ_1 towards σ_7, σ_8 will move in k waves leading eventually to k_1 spikes in σ_7 and k_2 in σ_8, such that $k_1 + k_2 = k$. This and other properties of $\Pi_{\text{OR-split}}$ will be discussed in Sect. 5.

3.2.2 Routing type for SN P systems

Now, we demonstrate the *routing type* [13] mentioned earlier in this section. We likewise express as matrices these routing types as demonstrated by appropriate SN P systems. Similarly, correctness of these matrix representations can be checked by performing matrix operation using Eq. (1).

1. **Sequential routing:**
   Let Π_{seq} given by $(\{a\}, \sigma_1, \sigma_2, \{(1,2)\})$, where $\sigma_1 = (1, \{a \to a\}), \sigma_2 = (0, \{a \to a\})$. The spiking transition matrix is
   $$M_{\Pi_{\text{seq}}} = \begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix}, \text{ with } C^{(0)} = (1, 0).$$
   This matrix appears as a sub-matrix (top left corner) of several matrices of basic routing blocks.
2. **Conditional routing:**
   The SN P system, denoted Π_{cond} is very similar to 2-way OR-split SN P system block, but instead of collecting the spikes in two distinct neurons, σ_7 and σ_8, only one, denoted σ_7, will receive spikes from σ_5 and σ_6. In this case σ_7 will also have a rule $a \to a$. The spiking transition matrix is
   $$M_{\Pi_{\text{cond}}} = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 2 & 2 & 0 \\ 0 & 0 & 0 & -2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}, \text{ with}$$
   $C^{(0)} = (1, 0, 0, 0, 0, 0, 0)$.

   We consider the same changes as in the case of $\Pi_{\text{OR-split}}$ applied to Π_{cond} and accordingly to $M_{\Pi_{\text{cond}}}$ and the initial configuration. Then this conditional routing model will still perform as expected.
3. **Parallel routing:**
   The SN P system, Π_{par}, consists of four neurons, very similar to those defined for conditional routing; only the fourth neuron is slightly different, $\sigma_1 = (1, \{a \to a\})$, $\sigma_2 = \sigma_3 = (0, \{a \to a\})$, $\sigma_4 = (0, \{a^2 \to a\})$ and $\text{syn} = \{(1,2), (1,3), (2,4), (3,4)\}$. The spiking transition matrix is given by
   $$M_{\Pi_{\text{par}}} = \begin{pmatrix} -1 & 1 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -2 \end{pmatrix}, \text{ with } C^{(0)} = (1, 0, 0, 0).$$
   This matrix appears as a sub-matrix (top left corner) of the matrix describing 2-way OR-split SN P system block.
4. **Iteration routing:**
   The SN P system, Π_{iter}, contains three neurons $\sigma_1 = (1, \{a \to a\})$, $\sigma_2 = \sigma_3 = (0, \{a \to a\})$ and $\text{syn} = \{(1,2), (2,3), (3,1)\}$. The spiking transition matrix is
   $$M_{\Pi_{\text{iter}}} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix}, \text{ with } C^{(0)} = (1, 0, 0).$$

Remark: Notice that the non-zero sub-matrix of the matrix representing the basic routing blocks AND-join, AND-split and OR-join are sub-matrices of the matrices for conditional and parallel routings.

3.2.3 Boundedness and liveness of SN P systems

We now turn our focus on the properties of SN P systems as described by its configurations. These properties described how SN P system behaves as it performs its function.

Definition 10 We call a **rule** r **live** for an initial configuration $C^{(0)}$ if for every $C^{(k)}$, $C^{(k)} \in R(\Pi, C^{(0)})$, there exists a valid spiking sequence from $C^{(k)}$ that contains and applies r.

An SN P system Π is **live** for $C^{(0)}$ if all its rules are live for $C^{(0)}$.

This means for SN P system Π to be *live,* rules in Π must not be useless permanently during the computation. A generating or recognizing Π halts after computation. This means no more rules can be applied and therefore *"dead"* or a *"deadlock"*. We adapt the idea of the so-called *"quasi-live transition"* from Petri nets for SN P systems.

Definition 11 A rule r is **quasi-live** for an initial configuration $C^{(0)}$, if there is a valid spiking sequence from $C^{(0)}$ that contains and applies r.

We call an SN P system **quasi-live** if all its rules are quasi-live.

The idea of quasi-liveness allows SN P systems to be *"not dead"* even some of the neurons are *"useless"* or *"dead"* while the other neurons are still functional. Note that to keep SN P systems *live,* it must have a *"feedback-loop"* or an *"iteration routing"* structure where all rules involved are live.

For the following illustrations, we have for all rules $E/a^c \rightarrow a^p$ of each neuron in the SN P system, $E = a^c$. The absolute values of the entries of M_Π for each Π, indicate the values of the exponents c and p of a. $C^{(0)}$ is the corresponding initial configuration of each Π.

Example 2 Let Π_1 be an SN P system, such that, $M_{\Pi_1} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$ with $C^{(0)} = (1,0)$. It is not hard to see that Π_1 is live.

Example 3 Let Π_2 be an SN P system, such that $M_{\Pi_2} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & -1 \end{pmatrix}$ with $C^{(0)} = (1,0,0)$. It is not hard to see that Π_2 is quasi-live (without deadlock).

Example 4 Let Π_3 be an SN P system, such that $M_\Pi = \begin{pmatrix} -2 & 2 & 2 & 0 \\ -2 & 1 & 1 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$ with $C^{(0)} = (2,0,0,0)$. It is not hard to see that Π_3 is quasi-live with deadlock.

We define below a *bounded* SN P system with respect to the amount of spikes every neuron has after some computation. First, we define boundedness of a neuron.

Definition 12 A neuron σ is **bounded** for an initial configuration $C^{(0)}$ if there is a positive integer s such that, $C(\sigma) \leq s$, for every configuration $C \in R(\Pi, C^{(0)})$, where $C(\sigma)$ is the amount of spikes in σ in configuration C. We say, σ is an s-bounded neuron.

An SN P system Π is **bounded** for an initial configuration $C^{(0)}$, if all neurons are bounded for $C^{(0)}$. Π is s-bounded if all the neurons are s-bounded.

If $s = 1$, then we call the SN P system Π, **safe.**

Example 5 Let Π_4 be an SN P system, such that, $M_{\Pi_4} = \begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ with $C^{(0)} = (1,0,0)$. It is not hard to see that Π_4 is unbounded.

Note that σ_3 of Π_4 accumulates spikes unboundedly during the computation.

Example 6 The SN P system of Example 1 has neurons σ_1, σ_2 and σ_3 2-bounded and σ_4 unbounded. This is also verified in Sect. 5.

4 SN P systems and kP systems

kP systems have been conceived as a membrane computing model allowing to specify and verify problems from various areas, from specific computer science topics, such as communication and synchronisation [22], to applications in synthetic biology [40, 41]. The modelling and verification capabilities of these models have been presented in [22]. Relationships with other classes of P systems, such as P systems with active membranes and neural-like P systems [23], and membrane systems with symport/antiport rules [22].

In this section we illustrate how potential connections between SN P systems and kP systems are built. We do not intend to make a thorough investigation of these relationships, as our goal is relatively limited now, i.e., to illustrate how some specific SN P system examples are expressed as kP systems with equivalent behaviour, facilitating the use of the verification tools developed for these models. We are also aware that there are other types of SN P systems that need to be investigated in relation to kP systems as well, and these topics will be considered in a broader context.

Remark 2 For an SN P system, Π, as in Definition 1, we build a kP system, $k\Pi_\Pi$. Its components and structure are as in Definition 3.

1. The set of objects is $A = \{a\}$.
2. The set of edges (links), denoted L, of the graph giving the membrane structure μ, includes pairs $\{C_i, C_j\}$, if and only if, at least one of (i, j) or (j, i) is in syn, i.e., there is a synapse between σ_i and σ_j or vice-versa. Although the edges (links) of L are bi-directional, the objects might circulate only in one direction, as specified by the rules and in accordance with the synapse that is considered in Π.

3. For each neuron $\sigma_i, 1 \leq i \leq m$, of Π, a compartment NC_i of type C_i is considered for $k\Pi_\Pi$. One might also have a compartment *NEnv* of type *Env*.
4. For each set of rules $R_i, 1 \leq i \leq m$, of σ_i in Π, we build in C_i the following
   (a) set of rules, R'_i :
      i. for each rule of type (1), $E/a^c \rightarrow a^p \in R_i$, if $(i,j_1), \dots (i,j_h) \in syn$, $h \geq 1$, a rule $a^c \rightarrow (a^p, C_{j_1},) \dots, (a^p, C_{j_h}) \{g_E\}$ is added to R'_i, where g_E is the guard obtained from the regular expression E. Please note that as $(i,j_1), \dots (i,j_h) \in syn$, then one must have $\{i,j_1\}, \dots \{i,j_h\} \in L$. When $h = 0$, i.e., no synapse going out of σ_i, then the corresponding rule in R'_i is $a^c \rightarrow (a^p, Env) \{g_E\}$. R'_{Env}, the set of rules associated with type Env, is $\emptyset$.
      ii. for each rule of type (2), $a^c \rightarrow \lambda \in R_i$, a rule $a^c \rightarrow \lambda \{= a^c\}$ is added to R'_i.
   (b) execution strategy, δ_i : is always choice, as one single rule from those applicable must be selected; δ_{Env} is also choice.
5. The output compartment, i_0, is only considered in $k\Pi_\Pi$ when an output neuron is distinguished in Π.

Below we are focusing on translating Example 1 and 2-way OR-split SNP system, into kP system specifications.

The SN P system introduced in Example 1 is now specified using kP systems.

Example 7 The following kP system

$k\Pi_\Pi = (\{a\}, \mu, NC_1, NC_2, NC_3, NC_4, 4)$,

is built in accordance with Remark 2. In this case, we have the edges (links) $L = \{\{NC_1, NC_2\}, \{NC_1, NC_3\}, \{NC_2, NC_3\}, \{NC_3, NC_4\}\}$, where the compartments $NC_i = (C_i, w_{i,0})$, $1 \leq i \leq 4$. The types of these compartments are $C_i = (R_i, \delta_i)$, $1 \leq i \leq 4$, and the initial multisets $w_{1,0} = a^2$, $w_{2,0} = w_{3,0} = a$ and $w_{4,0} = \lambda$. The sets of rules that appear in the types above are $R_1 = \{r_{1,1} : a \rightarrow (a, C_2)(a, C_3) \{= a^2\}, r_{1,2} : a^2 \rightarrow (a, C_2)(a, C_3) \{= a^2\}\}$, $R_2 = \{r_{2,1} : a \rightarrow (a, C_1)(a, C_3) \{= a\}\}$, $R_3 = \{r_{3,1} : a \rightarrow (a, C_4) \{= a\}, r_{3,2} : a^2 \rightarrow \lambda \{= a^2\}\}$ and $R_4 = \emptyset$. The execution strategies are choice (or alternative), i.e., $\delta_1 = \{r_{1,1}, r_{1,2}\}, \delta_2 = \{r_{2,1}\}, \delta_3 = \{r_{3,1}, r_{3,2}\}$ and $\delta_4 = \{\}$. The output compartment is NC_4.

Remark 3 One can observe that the compartment NC_4 of type C_4 collects the objects from NC_3 sent out by rule $r_{3,1}$ and acting as an environment.

It is easy to observe that $k\Pi_\Pi$ and Π (Π from Example 1) have the same behaviour.

This kP system will be used in Sect. 5 for simulating and verifying certain properties of this example.

Next we present the kP system obtained from the SN P system associated with the **2-way OR-split block**. We only present the set of rules, initial multiset and execution strategies.

Example 8 Let us denote by $k\Pi_{\Pi_{\text{OR-split}}}$, the kP system built, based on Remark 2, for the SN P system, $\Pi_{\text{OR-split}}$. For each of the eight neurons of the SN P system $\Pi_{\text{OR-split}}$ a compartment, $NC_i, 1 \leq i \leq 8$, of type $C_i = (R_i, \delta_i)$ will be constructed in the kP system, $k\Pi_{\Pi_{\text{OR-split}}}$.

The set of rules are $R_1 = \{r_{1,1} : a \rightarrow (a, C_2)(a, C_3) \{\geq a\}\}$, $R_i = \{r_{i,1} : a \rightarrow (a, C_4) \{= a\}\}$, $i = 2, 3$, $R_4 = \{r_{4,1} : a^2 \rightarrow (a^2, C_5)(a^2, C_6) \{= a^2\}, r_{4,2} : a^2 \rightarrow (a, C_5)(a, C_6) \{= a^2\}\}$, $R_5 = \{r_{5,1} : a^2 \rightarrow (\lambda, C_7) \{= a^2\}, r_{5,2} : a \rightarrow a \{= a\}\}$, $R_6 = \{r_{6,1} : a^2 \rightarrow (a, C_8) \{= a^2\}, r_{6,2} : a \rightarrow \lambda \{= a\}\}$, $R_7 = R_8 = \emptyset$.

The initial multisets are $w_{1,0} = a^k$, $k \geq 1$, and $w_{i,0} = \lambda$, $2 \leq i \leq 8$, and the execution strategies are all choice.

Both, $\Pi_{\text{OR-split}}$ and $k\Pi_{\Pi_{\text{OR-split}}}$, finally obtain in neurons σ_7, σ_8 and compartments NC_7, NC_8, respectively, the expected results.

Remark 4 The mapping of Example 1 and 2-way OR-split SNP system into kP system specifications, by using Remark 2, shows how natural is the process of getting the components $NC_i, 1 \leq i \leq n$, of the kP system from the neurons σ_i of the SN P system, where $n = 4$, in the case of Example 1, and $n = 8$, for the 2-way OR-split system.

A simpler solution to the problem of splitting the amount of objects present in the initial compartment, by using a kP system, $k\Pi'_{\Pi_{\text{OR-split}}}$, with only three compartments, NC_1, NC_2 and NC_3, can be constructed. The k objects from NC_1 will be distributed non-deterministically to the other two.

NC_1 is of type $C = (R, \delta)$ and NC_2, NC_3 of type $C' = (\emptyset, \delta)$. R contains the rule, $r : a \rightarrow (a, C') \{\geq a\}$, and δ might be choice (of a rule, r) or sequence. This solution exploits the fact that the rule r sends non-deterministically an object a to one of the two compartments, NC_2, NC_3, of the same type C'.

5 Simulation and verification using kPWorkbench

To provide a tool support for kernel P systems, an integrated software suite, the kPWorkbench [42] platform, has been developed (available and downloadable from its website [43]). The tool includes two simulators, a native one, which allows the execution of the entire system or a step-wise approach, and the Flame simulator [17], a general purpose large scale agent based simulation environment. The later is meant to allow a kP system to be expressed as a set of communicating X-machines [27] and then executed on high-performance hardware platform [11].

The verification component of kPWorkbench [26] checks the correctness of kP system models. Verification process works by exhaustively analysing all possible execution paths. Verification checks if system in question meets requirements, expressed in a formal logic [32, 33, 35]. Verification, in particular model checking, has been widely applied to the analysis of various systems, e.g. safety-critical systems [34, 36], concurrent systems [6], distributed systems [54], network protocols [37], systems and synthetic biology [10, 39], multi-agent systems [1] and pervasive systems [9, 38], as well as some engineering applications [14, 44, 45].

Any kP system model can be expressed using kPWorkbench 's kP-Lingua language—see [23], for details, including the language syntax. A kP-lingua model can be executed, using some of the above mentioned simulators, or formally verified, through the verification component of kPWorkbench .

To assist users in verification process, which is a very cumbersome process for non-experts, the platform also features a user friendly property language, *kP-Queries*, based on *natural language* statements, which makes the property specification a much easier task. The query language comprises a list of property patterns written as natural language statements. This is very useful for non-experts as they do not need to know the syntax of such query languages, relying on existing pre-defined patterns. The properties expressed in *kP-Queries* are verified using the Spin [28] and NuSMV [16] model checkers after being translated into corresponding *Linear Temporal Logic (LTL)* and *Computation Tree Logic (CTL)* syntax.

We now use kPWorkbench to model and analyse the behaviour of the SN P system described in Example 1 in Sect. 3. The corresponding kP system model $k\Pi_\Pi$ has been expressed in kPWorkbench 's kP-Lingua language as presented in Fig. 1a. In kP-Lingua notation, the usual multiset notation is replaced by exposing the multplicity of a symbol in front of it, i.e., a^3 will become `3a`.

The model in Fig. 1a has four compartment types, `C1`, `C2`, `C3` and `C4` with corresponding instances `NC1`, `NC2`, `NC3` and `NC4`, respectively. The compartment `NC1` starts with initial multiset `2a`, `NC2` and `NC3` with `a`; `NC4` is initially empty. Only one of the two rules of `C1` is selected non-deterministically. The first rule is executed only if its guard `=2a` is true. This rule also sends an `a` to the instance of the type `C2` and `C3`. The rules in the compartments `C2` and `C3` are executed similarly. `NC4` has no rules.

To observe the dynamic evolution of the system, we have run simulation experiments using kPWorkbench 's native simulator. Table 1 shows some simulation results. We have presented a finite halting computation (left) and the first steps from a longer computation (right).

Remark 5 Table 1 shows on the left a computation ending after 7 steps, with a^6 in the output compartment, NC4. This means that the corresponding SN P system will stop with 6 spikes in σ_4. The simulation that appears on the right, shows that steps 4 and 6 have the same configuration with respect to compartments NC1, NC2 and NC3. Hence, this computation might stop later on, after n steps ($n \geq 10$ for the configuration after 6 steps) or might continue forever.

Remark 6 Tables 2 and 4 summarise some of the properties verified for the $k\Pi_\Pi$ model and $k\Pi_{\Pi_{\text{OR-split}}}$ model, respectively. One can observe that any property that refers to the number of spikes, n, of a neuron of the SN P system is translated to the same property, but referring to a^n, in the corresponding compartment of the kP system. This observation together with Remark 4 show how natural is the verification of an SN P system, derived directly from the verification of the associated kP system.

The first line of each property in these tables expresses the property through a natural language statement. Hence, its meaning becomes obvious to the reader.

Some interesting properties that have been verified for the $k\Pi_\Pi$ model can be seen in Table 2. The verification results of most of these properties are true, for example property 1: eventually the number of objects `a` in `NC4` will be greater than 0, or properties 2, 3 that state that the number of `a` objects in compartments `NC1-NC3` will not exceed 1 or 2. This proves what has been presented in Sect. 3.2, Example 6, where the boundedness of certain neurons of the SN P system, Π, presented in Example 1. Two properties, 4 and 6, are not true. Property 4 checks if the computation will halt with an `a` in compartment `NC1` and empty compartments `NC2` and `NC3`. Many of the execution traces, those that halt, have this property; however, the NuSMV model checker could provide a counterexample with an infinite loop, showing it is not always true—the loop was considering the application of the same rules like those applied at steps 3–4, then repeated in steps 5–6, provided in Table 1 (right). Although

Fig. 1 kP-Lingua specifications for the kernel P systems $k\Pi_\Pi$ and $k\Pi_{\Pi_{\text{OR-split}}}$

```
type C1 {
  choice {
    =2a : a -> a (C2), a (C3) .
    =2a : 2a -> a (C2), a (C3) .
  }
}

type C2 {
  choice {
    =a : a -> a (C1), a (C3) .
  }
}

type C3 {
  choice {
    =a:  a -> a (Env) .
    =2a: 2a -> {} .
  }
}

type C4 {}

NC1 {2a} (C1) - NC2 {a} (C2) .
NC1 - NC3 {a} (C3) .
NC2 - NC3 .
NC3 - NC4 {} (C4) .
```

(a) KPL file for $k\Pi_\Pi$

```
type C1 {
  choice {
    >=a : a -> a (C2), a (C3) .
  }
}

type C2 {
  choice {
    =a : a -> a (C4) .
  }
}

type C3 {
  choice {
    =a : a -> a (C4) .
  }
}

type C4 {
  choice {
    =2a : 2a -> a (C5), a (C6).
    =2a : 2a -> 2a (C5), 2a (C6).
  }
}

type C5 {
  choice {
    =a  : a -> a (C7) .
    =2a : 2a -> {} .
  }
}

type C6 {
  choice {
    =2a : 2a -> a (C8) .
    =a  : a -> {} .
  }
}

type C7 {}

type C8 {}

NC1 {6a} (C1) - NC2 {} (C2) .
NC1 - NC3 {} (C3) .
NC2 - NC4 {} (C4) .
NC3 - NC4 .
NC4 - NC5 {} (C5) .
NC4 - NC6 {} (C6) .
NC5 - NC7 {} (C7).
NC6 - NC8 {} (C8).
```

(b) KPL file for $k\Pi_{\Pi_{OR-split}}$

Table 1 Simulation results for $k\Pi_\Pi$: a halting computation (left) and a longer, possible infinite one (right)

Step	NC1	NC2	NC3	NC4	Step	NC1	NC2	NC3	NC4
0	a^2	a	a		0	a^2	a	a	
1	a	a	a^2	a	1	a^2	a	a^2	a
2	a^2		a	a	2	a	a	a^2	a
3	a	a	a	a^2	3	a^2		a	a
4	a^2		a	a^3	4	a	a	a	a^2
5		a	a	a^4	5	a^2		a	a^3
6	a		a	a^5	6	a	a	a	a^4
7	a			a^6	7		…		

Table 2 Property patterns used in the verification experiments for $k\Pi_{\Pi}$

Prop.	kP-Queries and corresponding LTL translations	Result

1	`eventually (NC4.a >0)`
	`F (NC4.a > 0 & pInS)`
		true

2	`always (NC1.a <= 2 and (NC2.a <= 1 and NC3.a <= 2))`
	`G ((NC1.a <= 2 & NC2.a <= 1 & NC3.a <= 2) | !pInS)`
		true

3	`never (NC3.a > 2)`
	`! (F (NC3.a > 2 & pInS))`
		true

4	`eventually (NC1.a = 1 and (NC2.a = 0 and NC3.a = 0))`
	`F (NC1.a = 1 & NC2.a = 0 & NC3.a = 0 & pInS)`
		false

5	`always ((NC1.a = 1 and (NC2.a = 1 and NC3.a = 1)) implies (next (NC1.a = 2 and (NC2.a = 0 and NC3.a = 1))))`
	`G ( ((NC1.a = 1 & NC2.a = 1 & NC3.a = 1 & pInS) -> X (!pInS U (NC1.a = 2 & NC2.a = 0 & NC3.a = 1 & pInS)) ) | !pInS)`
		true

6	`always ((NC1.a = 2 and (NC2.a = 0 and NC3.a = 1)) implies (next (NC1.a = 1 and (NC2.a = 1 and NC3.a = 1))))`
	`G ( ((NC1.a = 2 & NC2.a = 0 & NC3.a = 1 & pInS) -> X (!pInS U (NC1.a = 1 & NC2.a = 1 & NC3.a = 1 & pInS)) ) | !pInS) &`
		false

7	`always (NC1.a = 0) implies (eventually (NC1.a = 1 and (NC2.a = 0 and NC3.a = 0)))`
	`G (((NC1.a = 0 & pInS) -> F (((NC1.a = 1 & NC2.a = 0) & NC3.a = 0) & pInS)) | !pInS)`
		true

Table 3 Simulation results: computation examples for $k\Pi_{\Pi_{\text{OR-split}}}$

Step	NC1	NC2	NC3	NC4	NC5	NC6	NC7	NC8	Step	NC1	NC2	NC3	NC4	NC5	NC6	NC7	NC8
0	a^6								0	a^6							
1	a^5	a	a						1	a^5	a	a					
2	a^4	a	a	a^2					2	a^4	a	a	a^2				
3	a^3	a	a	a^2	a	a			3	a^3	a	a	a^2	a	a		
4	a^2	a	a	a^2	a	a	a		4	a^2	a	a	a^2	a^2	a^2	a	
5	a	a	a	a^2	a	a	a^2		5	a	a	a	a^2	a^2	a^2	a	a
6		a	a	a^2	a	a	a^3		6		a	a	a^2	a	a	a	a^2
7				a^2	a	a	a^4		7				a^2	a	a	a^2	a^2
8					a	a	a^5		8					a^2	a^2	a^3	a^2
9							a^6		9							a^3	a^3

property 4 is false, property 7 states the context in which the computation will halt, with a particular final configuration: if in a step there are 0 objects `a` in the first compartment, then the computation will eventually halt, with one `a` in the first compartment and empty compartments `NC2` and `NC3`.

Our second example refers to the SN P system associated with the **2-way OR-split block**, $\Pi_{\text{OR-split}}$ and translated into the kP system $k\Pi_{\Pi_{\text{OR-split}}}$. This is presented in Fig. 1b. The model has eight compartment types, `C1`, ..., `C8`, with corresponding instances `NC1`, ..., `NC8`, respectively. The compartment `NC1` starts with the initial multiset `6a`. All other compartments are initially empty. `C4`, `C5` and `C6` have two rules, which are selected non-deterministically only one at a time and the selected rule is executed if the guard holds. For example, in `C5`, the first rule is executed only if its guard `=a` is true. This rule also sends an `a` to the instance of the type `C7`. The rules in the compartments `C4` and `C6` are executed similarly. `C1`, `C2`, and `C3` have only one rule, which is executed if the guard is true. `C7` and `C8` have no rules.

Table 3 presents two computations for $k\Pi_{\Pi_{\text{OR-split}}}$ with the same number of steps, but arriving to different configurations given the non-determinism. For the same kP system model we have verified some properties of interest and the results are presented in Table 4. Properties 1–6 check the number of objects in different compartments, at any time. These are summarised in Property 7, which shows that the number of objects in `NC2`, `NC3` is maximum 1 and in

Table 4 Property patterns used in the verification experiments for $k\Pi_{\Pi_{\text{OR-split}}}$

Prop.	kP-Queries and corresponding LTL translations	Result

1	`always (NC1.a >= 0 and NC1.a <= 6)`
	`G ((NC1.a >= 0 & NC1.a <= 6) | !pInS)`
		true

2	`always (NC2.a >= 0 and NC2.a <= 1)`
	`G ((NC2.a >= 0 & NC2.a <= 1) | !pInS)`
		true

3	`always (NC3.a >= 0 and NC3.a <= 1)`
	`G ((NC3.a >= 0 & NC3.a <= 1) | !pInS)`
		true

4	`always (NC4.a >= 0 and NC4.a <= 2)`
	`G ((NC4.a >= 0 & NC4.a <= 2) | !pInS)`
		true

5	`always (NC5.a >= 0 and NC5.a <= 2)`
	`G ((NC5.a >= 0 & NC5.a <= 2) | !pInS)`
		true

6	`always (NC6.a >= 0 and NC6.a <= 2)`
	`G ((NC6.a >= 0 & NC6.a <= 2) | !pInS)`
		true

7	`always ((NC2.a <= 1 and NC3.a <= 1) and (NC4.a <= 2 and (NC5.a <= 2 and NC6.a <= 2)))`
	`G ((((NC2.a <= 1 & NC3.a <= 1) & NC4.a <= 2) & NC5.a <= 2) & NC6.a <= 2) | !pInS)`
		true

8	`always (NC1.a = 3 implies (next (NC1.a = 2)))`
	`G (((NC1.a = 3 & pInS) -> X (!pInS U (NC1.a = 2 & pInS))) | !pInS)`
		true

9	`always (NC7.a >= 0 and NC7.a <= 6)`
	`G ((NC7.a >= 0 & NC7.a <= 6) | !pInS)`
		true

10	`always (NC8.a >= 0 and NC8.a <= 6)`
	`G ((NC8.a >= 0 & NC8.a <= 6) | !pInS)`
		true

11	`eventually ((NC7.a + NC8.a = 6) and (NC1.a + (NC2.a + (NC3.a + (NC4.a + (NC5.a + NC6.a)))) = 0))`
	`F ((NC7.a + NC8.a = 6 & ((((NC1.a + NC2.a) + NC3.a) + NC4.a) + NC5.a) + NC6.a = 0) & pInS)`
		true

12	`always ((NC7.a + NC8.a = 3) implies (next (NC7.a + NC8.a = 4)))`
	`G (((NC7.a + NC8.a = 3 & pInS) -> X (!pInS U (NC7.a + NC8.a = 4 & pInS))) | !pInS)`
		true

13	`always ((NC7.a + NC8.a = 6) implies (next (NC7.a + NC8.a = 6)))`
	`G (((NC7.a + NC8.a = 6 & pInS) -> X (!pInS U (NC7.a + NC8.a = 6 & pInS))) | !pInS)`
		true

14	`always (NC1.a + (NC2.a + (NC3.a + (NC4.a + (NC5.a + (NC6.a + (NC7.a + NC8.a)))))) = 6)`
	`G ((((((NC1.a + NC2.a) + NC3.a) + NC4.a) + NC5.a) + NC6.a) + NC7.a) + NC8.a = 6 | !pInS)`
		false

NC4–NC6 is maximum 2. Property 8 checks that the number of objects a in first compartment is decreasing. Properties 9 and 10 show that the number of objects in NC7 and NC8, respectively, is bounded by the initial number of objects (6 in NC1). Property 11 proves that eventually the sum of objects in NC7 and NC8 is the same as the initial number of objects in NC1, which verifies the statement made for the SN P system $\Pi_{\text{OR-split}}$ as a a model of the 2-way OR-split block presented in Sect. 3.2; it is also shown that the other compartments, NC1–NC6, are empty.

In this section, we have shown that formal verification of kP systems can be translated in a natural manner into the verification of SN P systems. Both, the modelling language, kP-lingua, and the verification mechanism provide direct mapping of the neurons into compartments and spikes into powers of the object a, respectively.

6 Conclusions

This paper has presented an approach based on matrix representation of SN P systems and a way of mapping such a system to a kP system. The matrix representation allows to express in a succinct and uniform way various structural and behavioural properties of such systems. The basic model of SN P systems has been studied in connection with kP systems and two examples of SN P system models have been translated into equivalent kP system ones. These two examples have been formally verified, by using an integrated software suite, kPWorkbench .

Regarding the first line of research, investigating structural and behavioural properties of the SN P systems using the matrix representation, we recall the fact that these properties are based on the the investigation considered in [13]. In [52], similar constructs are presented in the context of generalized communicating P systems. For instance, separation and joining blocks [52] are similar to AND-split and OR-join, respectively.

A number of new research avenues remain to be investigated, in the context of this work: the relationship between various classes of SN P systems and kP systems; the investigation of the matrix representation for other problems related to SN P systems - one of interest being the reverse computation; a more systematic investigation of various properties that are expressed by using kPWorkbench .

Finally, some comments regarding the second research direction mentioned above are presented. Reverse computation has been studied for various types of P systems [3–5, 7, 46]. A first question is related to how a reverse computation is defined for SN P systems. Reversing the rules of an SN P system might involve the use of concepts introduced for networks of cells [20] or generalized communicating P systems [52], whereby spikes are collected from various neurons and a number of spikes returned to one single neuron. The regular expressions associated with such rules must be then verified a posteriori, similar to a post-condition that must be true after executing a statement. The second aspect is related to expressing reverse computation with spiking translation matrices and the relationship with the matrix representing the initial SN P system.

Acknowledgements This work of SK is supported by EPSRC research grant EP/R043787/1. HNA is supported by DOST-ERDT research grants; Semirara Mining Corp. Professorial Chair for Computer Science of College of Engineering, UPDiliman; RLC grant from UPD-OVCRD. The authors would like to thank the anonymous reviewers for their efforts in carefully reading the paper and making valuable and constructive comments that allowed us to significantly improve the quality of the paper.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no confict of interest.

References

1. Abbink, H., et al. (2004). Automated support for adaptive incident management. In: *Proc. of the 1st Int. workshop on information systems for crisis response and management, ISCRAM'04*. Brussels (pp. 153–170).
2. Adorna, H. N. (2019). Matrix representations of spiking neural P systems: Revisited. In: G. Păun (Ed.) *Proceedings of the 20th Int. conference on membrane computing, CMC20, August 5–8, 2019, Curtea de Argeş, Romania, Editura BIBLIOSTAR, Râmnicu Vâlcea, 2019* (pp. 227–247).
3. Agrigoroaiei, O., & Ciobanu, G. (2009). Dual P systems. In: Corne, D. et al (Ed.) *9th Worskhop on membrane computing, LNCS 5391* (pp. 955–107).
4. Agrigoroaiei, O., & Ciobanu, G. (2010). Reversing computation in membrane systems. *Journal of Logic and Algebraic Programming*, *79*(3–5), 278–288.
5. Alhazov, A., & Miorita, K. (2010). On reversibility and determinism in P systems. In: G. Păun et al. (Ed.) *10th Worskhop on membrane computing, LNCS 5957* (pp. 158–158).
6. Alur, R., McMillan, K., & Peled, D. (2000). Model-checking of correctness conditions for concurrent objects. *Information and Computation*, *160*(1–2), 167–188.
7. Aman, B., & Ciobanu, G. (2017). Reversibility in parallel rewriting systems. *Journal of Universal Computer Science*, *23*(7), 692–703.
8. Martínez-del Amor, M. Á., Orellana-Martín, D., Cabarle, F. G. C., Pérez-Jiménez, M. J., & Adorna, H. N. (2017). Sparse-matrix representation of spiking neural P systems for GPU. In: *15th Brainstorming week on membrane computing) Fénix Editora*. Sevilla, Spain (pp. 161–170).
9. Arapinis, M., Calder, M., Denis, L., Fisher, M., Gray, P., & Konur, S., et al. (2009). Towards the verification of pervasive systems. *Electronic Communications of the EASST, 22*.
10. Bakir, M. E., Konur, S., Gheorghe, M., Krasnogor, N., & Stannett, M. (2018). Automatic selection of verification tools for efficient analysis of biochemical models. *Bioinformatics*, *34*(18), 3187–3195. https://doi.org/10.1093/bioinformatics/bty282.
11. Bakir, M.E., Konur, S., Gheorghe, M., Niculescu, I., & Ipate, F. (2014). High performance simulations of kernel P systems. In: *16th IEEE Int. conference on high performance computing and communications* (pp. 409–412). https://doi.org/10.1109/HPCC.2014.69
12. Bakir, M.E., Ipate, F., Konur, S., Mierlă, L., & Niculescu, I.-M. (2014). Extended simulation and verification platform for kernel

P systems. In: Gheorghe M. et al. (Ed.) *15th Int. conference on membrane computing, LNCS 8961* (pp. 158–178).

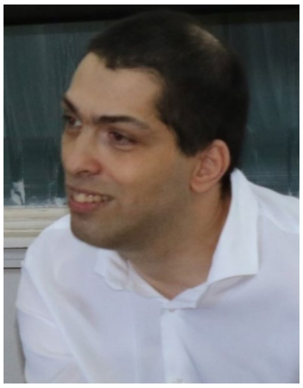
13. Cabarle, F. G. C., & Adorna, H. N. (2013). On structures and behaviors of spiking neural P systems and Petri nets. In: Csuhaj-Varjú, E. et al. (Ed.) *13th Int. conference on membrane computing, LNCS 7762* (pp. 145–160).
14. Camci, F., Eker, O. F., Baskan, S., & Konur, S. (2016). Comparison of sensors and methodologies for effective prognostics on railway turnout systems. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, *230*(1), 24–42. https://doi.org/10.1177/0954409714525145.
15. Carandang, J. P., Villaflores, J. M., Cabarle, F. G. C., Adorna, H. N., & Martínez-del Amor, M. Á. (2017). CuSNP: Spiking neural P systems simulators in CUDA. *Romanian Journal of Information Science and Technology*, *20*(1), 57–70.
16. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., & Tacchella, A. (2002). NuSMV version 2: An open source tool for symbolic model checking. In: *Proc. Int. conference on computer-aided verification (CAV 2002)*, *LNCS* (vol. 2404, pp. 359–364). Springer. https://doi.org/10.1007/3-540-45657-0_29
17. Coakley, S., Gheorghe, M., Holcombe, M., Chin, S., Worth, D., & Greenough, C. (2012). Exploitation of high performance computing in the FLAME agent-based simulation framework. In: *Proceedings of 14th IEEE Int. conference on high performance computing and communications* (pp. 538–545). https://doi.org/10.1109/HPCC.2012.79
18. Dela Cruz, R. T. A, Cailipan, D. P., Cabarle, F. G. C., Hernandez, N. H., Buño, K., Adorna, H. N., & Carandang, J. P. (2018). Matrix representation and simulation algorithm for spiking neural P systems with rules on synapses. In: P. L. Fernandez, Jr., H. N. Adorna, A. A. Sioson, J. D. L. Caro (Ed.) *Proc. 18th Philippine Computing Science Congress (PCSC2018)* (pp. 104–112).
19. Dela Cruz, R. T. A, Jimenez, Z., Cabarle, F. G. C., Adorna, H. N., Buño, K., Hernandez, N. H., & Carandang, J. P. (2018). Matrix representation of spiking neural P systems with structural plasticity. In: P. L. Fernandez, Jr., H. N. Adorna, A. A. Sioson, J. D. L. Caro (Ed.) *Proc. 18th Philippine Computing Science Congress (PCSC2018)* (pp. 152–164).
20. Freund, R., & Verlan, S. (2007). A formal framework for static (tissue) P systems. In: G. Eleftherakis, et al. (Ed.) *8th Worskhop on membrane computing, LNCS 4860* (pp. 271–284).
21. Gheorghe, M., Ceterchi, R., Ipate, F., & Konur, S. (2017). Kernel P systems modelling, testing and verification—sorting case study. In: A. Leporati, et al. (Ed.) *17th Int. Conference on Membrane Computing, LNCS 10105* (pp. 233–250). Cham.
22. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., & Lefticaru, R. (2018). Kernel P systems: From modelling to verification and testing. *Theoretical Computer Science*, *724*, 45–60.
23. Gheorghe, M., Ipate, F., Dragomir, C., Mierlă, L., Valencia-Cabrera, L., García-Quismondo, M., & Pérez-Jiménez, M.J. (2013). Kernel P Systems—Version I. In *11th Brainstorming week on membrane computing* (pp. 97–124). http://www.gcn.us.es/files/11bwmc/097_gheorghe_ipate.pdf
24. Gheorghe, M., Konur, S., & Ipate, F. (2017). Kernel P systems and stochastic P systems for modelling and formal verification of genetic logic gates. In: A. Adamatzky (Ed.) *Advances in unconventional computing, volume 1, theory* (pp. 661–675). Cham . https://doi.org/10.1007/978-3-319-33924-5_25
25. Gheorghe, M., Konur, S., Ipate, F., Mierlă, L., Bakir, M. E., & Stannett, M. (2015). An integrated model checking toolset for kernel P systems. In: G. Rozenberg, et al. (Ed.) *16th Int. conference on membrane computing, LNCS 9504* (pp. 153–170). Springer.
26. Gheorghe, M., Konur, S., Ipate, F., Mierlă, L., Bakir, M. E., & Stannett, M. (2015). An integrated model checking toolset for kernel P systems. In: G. Rozenberg, et al. (Ed.) *16th Int. conference on membrane computing, LNCS 9504* (pp. 153–170). Springer. https://doi.org/10.1007/978-3-319-28475-0_11
27. Holcombe, M. (1988). X-machines as a basis for dynamic system specification. *Software Engineering Journal*, *3*(2), 69–76. https://doi.org/10.1049/sej.1988.0009.
28. Holzmann, G. J. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, *23*(5), 275–295. https://doi.org/10.1109/32.588521.
29. Ibo, G.N, & Adorna, H.N. (2011). Periodicity as a dynamical aspect of generative spiking neural P systems. In: M. Gheorghe, et al. (Ed.) *Pre-Proc. 12th Int. conference on membrane computing (CMC12), Fontainebleau, France, 23–26 August 2011* (pp. 225–240).
30. Ionescu, M., Păun, G., & Yokomori, T. (2006). Spiking neural P systems. Fundamenta Informaticae **71**(2–3), 279–308 .
31. Jimenez, Z.B., Cabarle, F.G.C., Dela Cruz, R.T.A, Buño, K., Adorna, H.N., Hernandez, N.H., & Zeng, X. (2018). Matrix representation and simulation algorithm of spiking neural P systems with structural plasticity. In: *Pre-Proc. Asian conference membrane computing (ACMC2018)* (pp. 10–14).
32. Konur, S. (2006). A decidable temporal logic for events and states. In: *Thirteenth international symposium on temporal representation and reasoning (TIME'06)* (pp. 36–41). https://doi.org/10.1109/TIME.2006.1
33. Konur, S. (2008). An interval logic for natural language semantics. In: *Proceedings of the seventh conference on advances in modal logic, Nancy, France, 9–12 September 2008* (pp. 177–191).
34. Konur, S. (2010). Real-time and probabilistic temporal logics: An overview. CoRR **abs/1005.3200**.
35. Konur, S. (2010). A survey on temporal logics. CoRR **abs/1005.3199**.
36. Konur, S. (2014). Towards light-weight probabilistic model checking. *Journal of Applied Mathematics*, *2014*, 15. https://doi.org/10.1155/2014/814159.
37. Konur, S., & Fisher, M. (2011). Formal analysis of a VANET congestion control protocol through probabilistic verification. In: *Proceedings of the 73rd IEEE vehicular technology conference, VTC Spring 2011, 15–18 May 2011, Budapest, Hungary* (pp. 1–5). IEEE. https://doi.org/10.1109/VETECS.2011.5956327
38. Konur, S., Fisher, M., Dobson, S., & Knox, S. (2014). Formal verification of a pervasive messaging system. *Formal Aspects of Computing*, *26*(4), 677–694. https://doi.org/10.1007/s00165-013-0277-4.
39. Konur, S., & Gheorghe, M. (2015). A property-driven methodology for formal analysis of synthetic biology systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *12*, 360–371. https://doi.org/10.1109/TCBB.2014.2362531.
40. Konur, S., Gheorghe, M., Dragomir, C., Ipate, F., & Krasnogor, N. (2014). Conventional verification for unconventional computing: A genetic XOR gate example. *Fundamenta Informaticae*, *134*(1–2), 97–110.
41. Konur, S., Gheorghe, M., Dragomir, C., Mierlă, L., Ipate, F., & Krasnogor, N. (2015). Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. *ACS Synthetic Biology*, *4*(1), 83–92.
42. Konur, S., Mierlă, L., Ipate, F., & Gheorghe, M. (2020). kPWorkbench: A software suit for membrane systems. *SoftwareX*, *11*, 100407.
43. kPWorkbench website: https://github.com/kernel-p-systems/kpworkbench.
44. Lefticaru, R., Bakir, M.E., Konur, S., Stannett, M., & Ipate, F. (2018). Modelling and validating an engineering application in kernel P systems. In: M. Gheorghe (Ed.) *18th Int. conference on*

membrane computing, LNCS 10725 (pp. 183–195). Cham. https://doi.org/10.1007/978-3-319-73359-3_12
45. Lefticaru, R., Konur, S., Yildirim, Ü., Uddin, A., Campean, F., & Gheorghe, M. (2017). Towards an integrated approach to verification and model-based testing in system engineering. In: *The international workshop on engineering data- & model-driven applications (EDMA-2017)* (pp. 131–138). https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.25
46. Leporati, A., Zandron, C., & Mauri, G. (2006). Reversible P systems to simulate Fredkin circuits. *Fundamenta Informaticae*, *74*, 529–548.
47. Pan, L., Wu, T., & Zhang, Z. (2016). A bibliography of spiking neural P systems. *Bulletin of the International Membrane Computing Society (I M C S)*, *1*(1), 63–78.
48. Păun, G. (1998). Computing with membranes. Tech. rep., Turku Centre for Computer Science . http://tucs.fi/publications/view/?pub_id=tPaun98a.
49. Păun, Gh. (2000). Computing with membranes. *Journal of Computer and System Sciences*, *61*(1), 108–143. https://doi.org/10.1006/jcss.1999.1693.
50. Păun, Gh. (2002). *Membrane computing—An introduction*. New York: Springer.
51. Păun, G., Rozenberg, G., & Salomaa, A. (Eds.). (2010). *The Oxford Handbook of Membrane Computing*. Oxford University Press.
52. Verlan, S., Bernardini, F., Gheorghe, M., & Margenstern, M. (2008). Generalized communicating P systems. *Theoretical Computer Science*, *404*, 170–184.
53. Verlan, S., Freund, R., Alhazov, A., Ivanov, S., & Pan, L. (2020). A formal framework for spiking neural P systems. *Journal of Membrane Computing*, *2*(4), 355–368. https://doi.org/10.1007/s41965-020-00050-2.
54. Yabandeh, M. (2011). Model checking of distributed algorithm implementations. Ph.D. thesis, IC.
55. Zeng, X., Adorna, H. N., Martínez-del Amor, M.Á., Pan, L., & Pérez-Jiménez, M. J. (2009). Matrix representation of spiking neural P systems. In: M. Gheorghe, et al. (Ed.) *10th Int. conference on membrane computing, LNCS 6510* (pp. 377–391).

Marian Gheorghe is currently a Professor of Computational Modelling and Software Engineering with the University of Bradford. MG obtained his PhD and BSc in Mathematics and Computer Science from the University of Bucharest. MG has investigated a large variety of computational models, such as rewriting systems – formal grammars, multiset rewriting, automata, process algebras and Petri nets. MG has also an interest in the study of new classes of unconventional computational models, especially membrane systems, and their relationships with other (unconventional) computational models, as well as the formal verification of such models. MG has published extensively in high-profile journals.

Raluca Lefticaru is a Lecturer in Computer Science with the University of Bradford and a Visiting Researcher with the University of Sheffield, UK. Previously she has been a Lecturer at the University of Bucharest, Romania, and has held several research positions in the UK. Her research interests include model-based testing, formal specification methodologies, P systems simulation, verification and testing. RL has published numerous articles in prestigious journals and conferences. She has participated in various research projects and has been on the PC of several conferences on Software Testing, Formal Methods and Membrane Computing.

Savas Konur is a Reader in the Department of Computer Science, University of Bradford. His research interests involve Formal Methods (mainly modelling, verification and analysis of complex, concurrent and stochastic systems) and design/development of software systems/tools/methods facilitating Formal Methods in various application areas, including Systems and Synthetic Biology, Ubiquitous Systems, Real-time Systems, Safety-critical Systems, Autonomous Systems and Multi-agent & Systems. He has published his results in numerous leading journals and conferences. His collaborative and interdisciplinary research programme has been funded by EPSRC, Innovate UK and EU Access Innovation.



Ionuț Mihai Niculescu holds a PhD in Computer Science from the University of Piteşti, Romania. For his thesis "Simulation of kernel P Systems using Communicating X-machines - Applications in FLAME" he was distinguished by the International Membrane Computing Society (IMCS) with the PhD Thesis Award for the year 2018. He has been an active member in two research projects, publishing papers on modelling, simulation, verification and testing. His main research interests are in membrane computing, computational modelling and parallel processing.

Henry N. Adorna is a Professor of (Theoretical) Computer Science since 2007 and Scientist at UP Diliman. He spent two years (from 2000 to 2002) at Lehrstuhl für Informatik I of RWTH Aachen under the DAAD PhD Sandwich Program to do his dissertation on abstract communication complexity of uniform computing models under Prof. Dr. Juraj Hromkovic. HNA went home to obtain his PhD in Mathematics from the UP Diliman in 2002. HNA works primarily on P systems since 2009. Other research interests of HNA include algorithmics for NP-hard problems, particularly combinatorial hard problems in computational biology, discrete mathematics, natural and unconventional computing models.