# Spiking Neural P Systems Simulation and Verification

Raluca Lefticaru
*Department of Computer Science*
*University of Bradford*
Bradford, UK
r.lefticaru@bradford.ac.uk

Marian Gheorghe
*Department of Computer Science*
*University of Bradford*
Bradford, UK
m.gheorghe@bradford.ac.uk

Savas Konur
*Department of Computer Science*
*University of Bradford*
Bradford, UK
s.konur@bradford.ac.uk

Ionuţ Mihai Niculescu
*Faculty of Science*
*University of Piteşti*
Piteşti, Romania
ionutmihainiculescu@gmail.com

Henry N. Adorna
*Department of Computer Science*
*University of Philippines Diliman*
Quezon City, Philippines
hnadorna@dcs.upd.edu.ph

***Abstract*—Spiking Neural (SN) P systems is a particular class of P systems that abstracts and applies ideas from neurobiology. Various aspects, representations and features have been studied extensively, but the tool support for modelling and analysing such systems is relatively limited. In this paper, we present a methodology that maps some classes of SN P systems to the equivalent kernel P system representations, which allows analysing SN P system dynamics using the kPWORKBENCH tool. We illustrate the applicability of our approach in some case studies, including an example system from synthetic biology.**

***Index Terms*—membrane computing; spiking neural P systems; kernel P systems; kPWorkbench; formal verification.**

## I. INTRODUCTION

Membrane computing [1], [2] is a new computational paradigm, inspired from the structure, functions and processes occurring in biological cells. *Membrane systems* (or *P systems*) are the main models in membrane computing. Since the initiation of this research field, a number of classes of P systems have been introduced and studied. In most cases, different aspects of living organisms have inspired researchers to define new models, which can be categorised as *cell-like*, *tissue-like* and *neural-like* P systems. A summary of the most important classes of these models can be found in [3], [4] and recent advances in membrane systems in [5]–[7].

*Spiking Neural P systems* (*SN P systems*, for short), introduced in [8], define one of the most studied classes of P systems. SN P systems are inspired by the neurophysiological behaviour of neurons (in brain) sending electrical impulses along axons to other neurons. Some important results have been presented with respect to SN P systems, in particular, generating or recognising elements of a set (see the survey papers [9], [10]). Also, in some variants, matrices have been used to represent SN P systems [11], [12]. We refer the reader to [13] for a comprehensive summary on matrix representations as well as Petri net-like properties of SN P systems.

Whilst there have been numerous studies on SN P systems, the modelling and analysis of these systems using computational tools have not been the focus of these studies as the tool support is very limited to some simulations [14], [15]. To overcome this limitation, *kernel P systems (kP systems)* [16] can be used as an auxiliary model to translate SN P systems into and benefit from the different computational features of kPWORKBENCH [17] tool that has been developed to support kP systems. The kP systems offer a unified approach for various P system models, making it a suitable generic model for capturing features of different classes of such systems and describing various problems.

In this paper, we present, through a number of examples and experiments, how certain types of SN P system models are analysed by translating them into the equivalent kP system ones and then analysing systems dynamics using the kPWORKBENCH tool. Here, we complement the analysis of SN P systems with simulation and formal verification tools of the kP system models. We illustrate the applicability of our approach in some case studies, including an example system from synthetic biology. The translation maps in a one-to-one manner neurons and synapses of the SN P systems into kP systems compartments and links, respectively, according to the methodology developed in [18].

The paper is structured as follows: Section II introduces SN P systems without delay rules, then kP systems and a mapping between them; Section III illustrates the concepts on a few examples, represented using both systems. Section IV presents the results obtained by simulating and verifying the kP models with kPWORKBENCH and Section V draws the conclusions and future research lines.

## II. BACKGROUND

### A. *Spiking Neural P Systems*

This section briefly gives the basic definitions regarding *spiking neural P systems*. First, some preliminary concepts and notations are introduced.

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 $\lambda$ and $A^+ = A^* \setminus \{\lambda\}$ denotes the set of non-empty strings. A *multiset* over $A$ is a mapping $f : A \to \mathbb{N}$. Considering only the elements from the support of $f$ (where $f(a_{i_j}) > 0$, for some $j$, $1 \leq j \leq p$), the multiset is represented as a string $a_{i_1}^{f(a_{i_1})} \dots a_{i_p}^{f(a_{i_p})}$, where the order is not important. In the sequel multisets will be represented by such strings.

The definition and other concepts related to SN P systems are taken from [11], [13]. In this paper, we consider only **SN P systems without delay rules**.

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

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

where

- $O = \{a\}$ is a singleton alphabet ($a$ is called spike);
- $\sigma_i, 1 \leq i \leq m$, are neurons, $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where
  - $n_i \geq 0$ is the number of spikes in $\sigma_i$;
  - $R_i$ is a finite set of rules of the following forms:
    - (*Type (1); spiking rules*) $E/a^c \to a^p$; where $E$ is a regular expression over $\{a\}$, and $c \geq 1$, $p \geq 1$, such that $c \geq p$;
    - (*Type (2); forgetting rules*) $a^s \to \lambda$, for $s \geq 1$, such that for each rule $E/a^c \to a^p$ of type (1) from $R_i$, $a^s \notin L(E)$;
- $syn = \{(i,j) | 1 \leq i,j \leq m, i \neq j\}$ (synapses between neurons);
- $in, out \in \{1, \dots .m\}$ indicate the input and output neurons, respectively.

*Remark 1:* The $in$ and $out$ neurons will not be distinguished in what follows.

The SN P system $\Pi$ computes by applying one rule from each neuron.

A *configuration* of SN P system $\Pi$ is an $m$-size vector of integers

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

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

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

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

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

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

We say a rule $r_x \in R_j$, $1 \leq j \leq m$, of neuron $\sigma_j$ is *applicable* at time $k$, $k \geq 0$, if and only if the amount of spikes in the neuron, $a_j^{(k)}$, satisfies $E_x$ (or $a_j^{(k)} \in L(E_x)$). If more than a rule is applicable then one of them is non-deterministically chosen.

In case of a spiking rule (type 1), the spike ($a$) is sent to all neurons to which a synapse exists outgoing from the neuron where the rule is applied. 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 $\sigma_j$ is exactly the amount of spikes it needs to be applied. Formally, if there exists a rule $a^s \to \lambda \in R_j$, then there cannot exist any rule $E/a^c \to a^p$ of type 1 in $R_j$ such that $a^s \in L(E)$.

In specifying 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 $E$.

A *computation* of the system is defined by the sequence of configurations, whereby at each step, a new $C' = (a'_1, a'_2, \dots, a'_m)$ is obtained from a configuration, $C = (a_1, a_2, \dots, a_m)$, writen as $C \Longrightarrow C'$, where $a_j, a'_j \in \mathbb{Z}^+ \cup \{0\}$, $1 \leq j \leq m$, represent the the number of spikes.

## *B. Kernel P Systems*

This section summarises the basic definitions of *kernel P systems*.

*Definition 2:* A kernel P system (kP system) of degree $n$ is a tuple

$$k\Pi = (O, \mu, C_1, \dots, C_n, i_0) \quad (1)$$

where $O$ is a finite set of objects, called *alphabet*; $\mu$ defines the *membrane structure*, which is a graph, $(V, L)$, where $V$ are vertices indicating compartments, and $L$ edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a *compartment* of the system consisting of a compartment type from $T$ and an *initial multiset*, $w_i$ over $O$; $i_0$ is the *output compartment* where the result is obtained (this will not be used in the paper).

*Definition 3:* $T$ is a *set of compartment types*, $T = \{t_1, \dots, t_s\}$, where $t_i = (R_i, \sigma_i)$, $1 \leq i \leq s$, consists of a set of rules, $R_i$, and an execution strategy, $\delta_i$, defined over $Lab(R_i)$, the labels of the rules of $R_i$.

*Definition 4:* A *rewriting and communication rule,* from a set of rules, $R_i$, $1 \leq i \leq s$, used in a compartment $C_{l_i} = (t_{l_i}, w_{l_i})$, $1 \leq i \leq n$, has the form $x \to y \; \{g\}$, where $x \in O^+$ and $y$ has the form $y = (a_1, t_1) \dots (a_h, t_h)$, $h \geq 0$, $a_j \in O$ and $t_j$ indicates a compartment type from $T$ – see Definition 3 – with instance compartments linked to the current compartment, $C_{l_i}$; $t_j$ might indicate the type of the current compartment, i.e., $t_{l_i}$ – in this case it is ignored; if a link does not exist (the two compartments are not in $L$) then the rule is not applied; if a target, $t_j$, refers to a compartment type that has more than one instance connected to $C_{l_i}$, then one of them will be non-deterministically chosen.

The definition of a rule from $R_i$, $1 \leq i \leq s$, is more general than the form provided above, see [19], but in this paper we only use the current form. The guards, denoted by $g$, are Boolean conditions. The guard must be true when a rule is applied.

In each compartment of a kP system a specific *execution strategy* can be defined – for more details see [19]. In this paper, the execution strategy of any compartment will be the *choice*, i.e., given a compartment type $t$ with the set of rules, $R_t = \{r_i | 1 \leq i \leq n\}$, the execution strategy is $\delta_t = (r_1, \ldots, r_n)$, which means that at any moment one rule from $R_t$ will be selected to be executed, from those that are applicable.

### C. From SN P Systems to kP Systems

In this section we illustrate how some SN P system models are translated into equivalent kP system ones and then analysed by using the tool supporting this class of P systems.

The methodology that maps an SN P system without delay into a kP system is described in [18]. Here, we only present its key features that help us describing the translation of several SN P system examples into their equivalent kP system models.

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

1) The same symbols (spikes, objects) are used by both systems.
2) For each neuron $\sigma_i$, $1 \leq i \leq m$, of $\Pi$, a compartment $NC_i$ of type $C_i$ is considered for $k\Pi_\Pi$. In some cases a compartment $NEnv$ of type $Env$ is added to $k\Pi_\Pi$, describing the environment. (As mentioned earlier, for each neuron of $\Pi$, a unique compartment is considered for $k\Pi_\Pi$.)
3) For any synapse linking two neurons, $\sigma_i$ and $\sigma_j$, of $\Pi$, a link between the types $C_i$ and $C_j$, corresponding to components $NC_i, NC_j$ associated to the two neurons, is considered for the kP system $k\Pi_\Pi$.
4) For each set of rules $R_i$, $1 \leq i \leq m$, of $\sigma_i$ in $\Pi$, rules simulating them will be added to $C_i$ (see [18]).

## III. Examples

In this section we present several models that will show the analysis capabilities of our approach by using formal verification. For each example we define first the SN P model, then construct the equivalent kP system one. Also, each kP system model is then written in kP-Lingua, allowing these models to be simulated and analysed with the kPWorkbench tool. Due to space constraints, we show in Fig. 1 only the kP-Lingua specifications for the kernel P systems $k\Pi_{\Pi_1}, k\Pi_{\Pi_2}, k\Pi_{\Pi_3}$. However, an archive with all the specification and verification files can be downloaded from [20].

*Example 1:* Let us first consider a very simple iteration routing SN P system [18], consisting of three neurons: $\Pi_1 = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn)$, where $\sigma_1 = (1, R_1)$ with $R_1 = \{a \to a\}$ and $\sigma_2 = \sigma_3 = (0, R_1)$; $syn = \{(1,2), (2,3), (3,1)\}$. It can be easily observed that the only computation is
$(1,0,0) \Longrightarrow (0,1,0) \Longrightarrow (0,0,1) \Longrightarrow (1,0,0) \Longrightarrow \ldots$
which is an infinite loop, firing one spike moving through the neurons $\sigma_1, \sigma_2$ and $\sigma_3$.

We can translate the SN P system $\Pi_1$ to the equivalent kP system $k\Pi_{\Pi_1}$, defined as follows:

$$k\Pi_{\Pi_1} = (\{a\}, \mu, NC_1, NC_2, NC_3),$$

built in accordance with Remark 2. In this case, we have the edges (links) $L = \{\{NC_1, NC_2\}, \{NC_2, NC_3\}, \{NC_3, NC_1\}\}$, where the compartments $NC_i = (C_i, w_{i,0})$, $1 \leq i \leq 3$. The types of $NC_1$, $NC_2$ and $NC_3$ are $C_1 = (R'_1, \delta_1)$, $C_2 = (R'_2, \delta_2)$, and $C_3 = (R'_3, \delta_3)$, respectively. The initial multisets $w_{1,0} = a$, $w_{2,0} = w_{3,0} = \lambda$. The set of rules that appear in the types above are $R'_1 = \{r_{1,1} : a \to (a, C_2) \ \{= a\}\}$, $R'_2 = \{r_{2,1} : a \to (a, C_3) \ \{= a\}\}$ and $R'_3 = \{r_{3,1} : a \to (a, C_1) \ \{= a\}\}$. The execution strategies are all choice.

*Example 2:* Let SN P system $\Pi_2 = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn)$, where $\sigma_1 = (6, R_1)$ with $R_1 = \{a^*/a \to a\}$, $\sigma_2 = (0, R_2)$ with $R_2 = \{a^*/a \to a\}$, and $\sigma_3 = (0, R_3)$ with $R_3 = \{a^*/a \to a\}$; $syn = \{(1,2), (2,3), (3,2)\}$. This SN P system model, starts with 6 spikes in $\sigma_1$ and 0 in $\sigma_2$ and $\sigma_3$, ending with 0 in $\sigma_1$ and 5 in $\sigma_2$ and 1 in $\sigma_3$. This final configuration will loop forever, like a steady state. This example illustrates in [18] a quasi-live SN P system, i.e., a system that has no deadlock, from each configuration there is a configuration where the system will move to. After a number of step the system will reach a steady state and will remain in that state an infinite number of steps.

We can translate the SN P system $\Pi_2$ to the equivalent kP system $k\Pi_{\Pi_2}$, defined as follows:

$$k\Pi_{\Pi_2} = (\{a\}, \mu, NC_1, NC_2, NC_3).$$

For this case, we have the edges (links) $L = \{\{NC_1, NC_2\}, \{NC_2, NC_3\}\}$, where the compartments $NC_i = (C_i, w_{i,0})$, $1 \leq i \leq 3$,. The types of these compartments are $C_i = (R'_i, \delta_i)$. The initial multisets $w_{1,0} = a^6$, $w_{2,0} = w_{3,0} = \lambda$. The set of rules that appear in the types above are $R'_1 = \{r_{1,1} : a \to (a, C_2) \ \{\geq a\}\}$, $R'_2 = \{r_{2,1} : a \to (a, C_3) \ \{\geq a\}\}$ and $R'_3 = \{r_{3,1} : a \to (a, C_2) \ \{\geq a\}\}$. The execution strategies are choice.

*Example 3:* Let SN P system $\Pi_3 = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn)$, where $\sigma_1 = (6, R_1)$ with $R_1 = \{a^*/a \to a\}$, $\sigma_2 = (0, R_2)$ with $R_2 = \{a^*/a \to a\}$, and $\sigma_3 = (0, \emptyset)$ with $R_3 = \emptyset$; $syn = \{(1,2), (2,1), (1,3)\}$. This SN P system model, starts with 6 spikes in $\sigma_1$ and 0 in $\sigma_2$ and $\sigma_3$. At step $k$, $k \geq 1$, the configuration of the system is $(5, 1, k)$. Again the system will loop forvever, keeping constant values in $\sigma_1$ and $\sigma_2$ and increasing by 1, at each step, the value in $\sigma_3$. This example illustrates in [18] an unbounded SN P system – the number of spikes in $\sigma_3$ is unbounded.

The SN P system $\Pi_3$ is translated to the equivalent kP system $k\Pi_{\Pi_3}$:

$$k\Pi_{\Pi_3} = (\{a\}, \mu, NC_1, NC_2, NC_3).$$

Here, we have the links $L = \{\{NC_1, NC_2\}, \{NC_1, NC_3\}\}$, where the compartments $NC_i = (C_i, w_{i,0})$, $1 \leq i \leq 3$,. The types of these compartments are $C_i = (R'_i, \delta_i)$. The initial multisets $w_{1,0} = a^6$, $w_{2,0} = w_{3,0} = \lambda$. The set of rules that appear in the types above are $R'_1 = \{r_{1,1} : a \to$

```
type C1 {
 choice {
  =a : a -> a (C2) .
 }
}
type C2 {
 choice {
  =a : a -> a (C3) .
 }
}
type C3 {
 choice {
  =a : a -> a (C1) .
 }
}
NC1 {a} (C1) - NC2 {} (C2) .
NC2 - NC3 {} (C3) .
NC3 - NC1 .
```

**(a)** KPL file for $k\Pi_{\Pi_1}$

```
type C1 {
 choice {
  >=a : a -> a (C2) .
 }
}
type C2 {
 choice {
  >=a : a -> a (C3) .
 }
}
type C3 {
 choice {
  >=a : a -> a (C2) .
 }
}
NC1 {6a} (C1) - NC2 {} (C2) .
NC2 - NC3 {} (C3) .
```

**(b)** KPL file for $k\Pi_{\Pi_2}$

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

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

NC1{6a}(C1)-NC2{}(C2) .
NC1-NC3{}(C3).
```

**(c)** KPL file for $k\Pi_{\Pi_3}$

Fig. 1. kP-Lingua specifications for the kernel P systems $k\Pi_{\Pi_1}$, $k\Pi_{\Pi_2}$ and $k\Pi_{\Pi_3}$

| Step | $NC_1$ | $NC_2$ | $NC_3$ |
|---|---|---|---|
| 0 | $a$ | $\lambda$ | $\lambda$ |
| 1 | $\lambda$ | $a$ | $\lambda$ |
| 2 | $\lambda$ | $\lambda$ | $a$ |
| 3 | $a$ | $\lambda$ | $\lambda$ |
| 4 | $\lambda$ | $a$ | $\lambda$ |
| 5 | $\lambda$ | $\lambda$ | $a$ |
| 6 | $a$ | $\lambda$ | $\lambda$ |
| 7 | $\lambda$ | $a$ | $\lambda$ |

**(a)** $k\Pi_{\Pi_1}$

| Step | $NC_1$ | $NC_2$ | $NC_3$ |
|---|---|---|---|
| 0 | $a^6$ | $\lambda$ | $\lambda$ |
| 1 | $a^5$ | $a$ | $\lambda$ |
| 2 | $a^4$ | $a$ | $a$ |
| 3 | $a^3$ | $a^2$ | $a$ |
| 4 | $a^2$ | $a^3$ | $a$ |
| 5 | $a$ | $a^4$ | $a$ |
| 6 | $\lambda$ | $a^5$ | $a$ |
| 7 | $\lambda$ | $a^5$ | $a$ |

**(b)** $k\Pi_{\Pi_2}$

| Step | $NC_1$ | $NC_2$ | $NC_3$ |
|---|---|---|---|
| 0 | $a^6$ | $\lambda$ | $\lambda$ |
| 1 | $a^5$ | $a$ | $a$ |
| 2 | $a^5$ | $a$ | $a^2$ |
| 3 | $a^5$ | $a$ | $a^3$ |
| 4 | $a^5$ | $a$ | $a^4$ |
| 5 | $a^5$ | $a$ | $a^5$ |
| 6 | $a^5$ | $a$ | $a^6$ |
| 7 | $a^5$ | $a$ | $a^7$ |

**(c)** $k\Pi_{\Pi_3}$

Fig. 2. Execution traces for kernel P systems $k\Pi_{\Pi_1}$, $k\Pi_{\Pi_2}$ and $k\Pi_{\Pi_3}$

| Step | $NC_1$ | $NC_2$ | $NC_3$ | $NC_4$ | $NC_5$ | $NC_6$ | $NC_7$ | $NC_8$ | $NC_9$ | $NC_{10}$ | $NC_{11}$ | $NC_{12}$ | $NEnv$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $a^3$ | $a^3$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ | $\lambda$ |
| 3 | $a^3$ | $X$ | $X$ | $X$ | $X$ | $X$ | $X$ | $\lambda$ | $\lambda$ | $a$ | $\lambda$ | $\lambda$ | $\lambda$ |
| 4 | $a^3$ | $X$ | $X$ | $X$ | $X$ | $X$ | $X$ | $\lambda$ | $a$ | $a$ | $\lambda$ | $\lambda$ | $\lambda$ |
| 6 | $a^3$ | $X$ | $X$ | $X$ | $X$ | $X$ | $X$ | $a$ | $a$ | $a^2$ | $\lambda$ | $\lambda$ | $\lambda$ |
| 25 | $a^3$ | $X$ | $X$ | $X$ | $X$ | $X$ | $X$ | $a^4$ | $a^5$ | $a^4$ | $a^2$ | $a^4$ | $a^4$ |
| 25' | $a^3$ | $X$ | $X$ | $X$ | $X$ | $X$ | $X$ | $a^4$ | $a^5$ | $a^4$ | $a^3$ | $a^3$ | $a^4$ |
| 25" | $a^3$ | $X$ | $X$ | $X$ | $X$ | $X$ | $X$ | $a^4$ | $a^4$ | $a^5$ | $a^3$ | $a^4$ | $a^3$ |

Fig. 3. Execution traces for kernel P system $k\Pi_{\Pi_4}$

$(a, C_2)(a, C_3)$ $\{\geq a\}\}$, $R'_2 = \{r_{2,1} : a \rightarrow (a, C_1)$ $\{\geq a\}\}$, $R'_3 = \emptyset$. The execution strategies are choice.

*Example 4:* In this example, we consider a simple *pulse generator system* [21], from synthetic biology. The system is composed of geometrically organised cells. A signalling molecule is produced, propagating then through the neighbouring cells. The pulse generator consists of two different bacterial strains, *sender cells* and *pulsing cells* (see Fig. 4). In our simple scenario there is one single sender cell, denoted $S1$ and 5 pulsing cells, three of them in the close vicinity of $S1$, denoted $P8$, $P9$ and $P10$, and two more, namely $P11$ and $P12$. The pulsing cells on the right, $P11$, $P9$ and $P12$, can also send the signalling molecule out to the environment. The sender cell contains the gene *luxI* from Vibrio fischeri. This gene codifies the enzyme *LuxI* responsible for the synthesis of the signalling molecule *3OC6-HSL* (*AHL*). In the SN P system model below, the signalling molecule is represented by the symbol $a$. Using only one symbol, $a$, it requires some

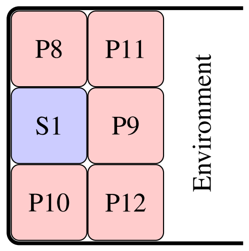

Fig. 4. Pulse generator system

codifications with powers of $a$ in order to denote entities of the model. In this respect we also need several neurons. The sender cell is defined by the neuron $\sigma_1$, the pulsing cells presented above are denoted by the neurons $\sigma_8$, $\sigma_9$, $\sigma_{10}$, $\sigma_{11}$ and $\sigma_{12}$. The rest of the neurons are used to simulate the production of *AHL* by the sender ($\sigma_2$), the distribution of *AHL* to the neighbouring cells ($\sigma_5$, $\sigma_6$, $\sigma_7$).

The SN P system, $\Pi_4$, describing this pulse generator is formally defined below

$\sigma_1 = \sigma_2 = (3, \{a^3 \to a^3\})$;
$\sigma_3 = (0, \{a^*/a^3 \to a^3, a^*/a^3 \to a^2, a^*/a^3 \to a\})$;
$\sigma_4 = (0, \{a^*/a^3 \to a^3\})$;
$\sigma_5 = (0, \{a^3 \to a, a^2 \to \lambda, a \to \lambda\})$;
$\sigma_6 = (0, \{a^3 \to \lambda, a^2 \to a, a \to \lambda\})$;
$\sigma_7 = (0, \{a^3 \to \lambda, a^2 \to \lambda, a \to a\})$;
$\sigma_8 = \sigma_9 = \sigma_{10} = \sigma_{11} = \sigma_{12} = (0, \{a^5a^*/a \to a\})$;
$\sigma_E = (0, \{\})$.

and synapses $syn = \{(1,2), (1,3), (2,1), (3,4), (3,5), (3,6), (3,7), (4,3), (5,8), (6,9), (7,10), (8,11), (10,12)$.

The SN P system $\Pi_4$ is translated to the equivalent kP system $k\Pi_{\Pi_4}$. Here, we only present the set of rules, initial multiset and execution strategies. For each neuron, $\sigma_i$, $1 \le i \le 12$, of the SN P system $\Pi_4$ a compartment, $NC_i$ of type $C_i = (R'_i, \delta_i)$ will be constructed in the kP system, $k\Pi_{\Pi_4}$.

For this example, we illustrate the compartment links as follows:

$$\begin{tikzcd}
 & & NC_4 & NC_5 \arrow[r, dash] & NC_8 \arrow[r, dash] & NC_{11} \arrow[dr, dash] & \\
NC_2 \arrow[r, dash] & NC_1 \arrow[r, dash] & NC_3 \arrow[u, dash] \arrow[ur, dash] \arrow[r, dash] \arrow[dr, dash] & NC_6 \arrow[r, dash] & NC_9 \arrow[rr, dash] & & NEnv \\
 & & & NC_7 \arrow[r, dash] & NC_{10} \arrow[r, dash] & NC_{12} \arrow[ur, dash] &
\end{tikzcd}$$

where the compartments $NC_i = (C_i, w_{i,0})$, $1 \le i \le 12$ and $NEnv = (Env, w_{Env,0})$. The types of these compartments are $C_i = (R'_i, \delta_i)$ and $Env = (R'_{Env}, \delta_{Env})$, respectively. The initial multisets $w_{1,0} = a^3$, $w_{2,0} = a^3$; $w_{i,0} = \lambda$ for $3 \le i \le 12$; and $w_{Env} = \lambda$. The set of rules that appear in the types above are as follows:

$R'_1 = \{r_{1,1} : a^3 \to (a^3, C_2)(a^3, C_3)\{= a^3\}\}$;
$R'_2 = \{r_{2,1} : a^3 \to (a^3, C_1)\{= a^3\}\}$;
$R'_3 = \{r_{3,1} : a^3 \to (a^3, C_4)(a^3, C_5)(a^3, C_6)(a^3, C_7)\{\ge a^3\}$,
$\quad r_{3,2} : a^3 \to (a^2, C_4)(a^2, C_5)(a^2, C_6)(a^2, C_7)\{\ge a^3\}$,
$\quad r_{3,3} : a \to (a, C_4)(a, C_5)(a, C_6)(a, C_7)\{\ge a^3\}\}$;
$R'_4 = \{r_{4,1} : a^3 \to (a^3, C_3)\{\ge a^3\}\}$;
$R'_5 = \{r_{5,1} : a^3 \to (a, C_8)\{= a^3\}$,
$\quad r_{5,2} : a^2 \to \lambda\{= a^2\}$,
$\quad r_{5,3} : a \to \lambda\{= a\}\}$;
$R'_6 = \{r_{6,1} : a^3 \to \lambda\{= a^3\}$,
$\quad r_{6,2} : a^2 \to (a, C_9)\{= a^2\}$,
$\quad r_{6,3} : a \to \lambda\{= a\}\}$;
$R'_7 = \{r_{7,1} : a^3 \to \lambda\{= a^3\}$,
$\quad r_{7,2} : a^2 \to \lambda\{= a^2\}$,
$\quad r_{7,3} : a \to (a, C_{10})\{= a\}\}$;
$R'_8 = \{r_{8,1} : a \to (a, C_{11})\{> a^4\}\}$;
$R'_9 = \{r_{9,1} : a \to (a, C_{Env})\{> a^4\}\}$;
$R'_{10} = \{r_{10,1} : a \to (a, C_{12})\{> a^4\}\}$;
$R'_{11} = \{r_{11,1} : a \to (a, C_{Env})\{> a^4\}\}$;
$R'_{12} = \{r_{12,1} : a \to (a, C_{Env})\{> a^4\}\}$.

## IV. Experiments using kPWorkbench

This section presents the kPWorkbench software suite that was used for the simulation and verification of the kP models, and then the results obtained.

### *A.* kPWorkbench

The kPWorkbench software platform [17] has been developed to provide a tool support for kP systems (downloadable from [22]). The platform allows modelling membrane systems in a Domain Specific Language, called kP–Lingua, an expressive formal language, simulation using a native CPU simulator [23] and a GPU simulator on high-performance hardware platform [24] and verification [25] using the NuSMV [26] and Spin [27] model checkers.

The verification component of kPWorkbench [25] checks the correctness of kP system models. Verification process works by exhaustively analysing all possible execution paths, checking if the system in question meets requirements, by expressing them in a formal logic [28]–[30]. Verification, in particular model checking, has been widely applied to the analysis of various systems, e.g. safety-critical systems [31], [32], concurrent systems [33], distributed systems [34], network protocols [35], systems and synthetic biology [36], [37], multi-agent systems [38] and pervasive systems [39], [40], as well as some engineering applications [41]–[43].

To assist users in verification, a challenging process for non-experts, kPWorkbench also features a natural language like property language, *kP-Queries*, which massively simplifies property specification. The query language comprises a list of pre-defined property patterns that frequently appear in the literature. This is very useful for non-experts as they do not need to know the syntax of such query languages. The properties expressed in *kP-Queries* are verified using the Spin and NuSMV tools after being translated into corresponding *Linear Temporal Logic (LTL)* and *Computation Tree Logic (CTL)* syntax.

The modelling, simulation, verification and testing aspects of kP systems have been presented in [44], [45] and applications in synthetic biology in [46]–[48].

## B. Simulation Results

Fig. 2 and Fig. 3 present some execution traces for the models presented in Examples 1 – 4, given by the kP systems $k\Pi_{\Pi_1} - k\Pi_{\Pi_4}$.

Fig. 2 presents the first 7 computation steps of the kP systems $k\Pi_{\Pi_1} - k\Pi_{\Pi_3}$. They are all deterministic systems. These simulations confirm the informal descriptions provided for each of them. Indeed, the simulation of $k\Pi_{\Pi_1}$ shows that the kP system has 3 distinct configurations and each of them is repeated after 3 steps. The second kP system, $k\Pi_{\Pi_2}$, will move to a steady state, reached after 6 steps, and will remain there, as a quasi-live system. The third example, $k\Pi_{\Pi_3}$, presents an unbounded system – the component $NC_3$ has an unbounded number of spikes.

Fig. 3 presents some traces of execution of 3 runs, each of 25 steps, of the $k\Pi_{\Pi_4}$ system describing a simplified pulse generator. Only compartments corresponding to the 6 cells and the environment are presented. For the rest the content is not listed (an $X$ appears in each of them). The kP system is non-deterministic and different runs might lead to distinct results. The steps 3, 4, 6 and 25 of the first run show the signal molecule leaving the pulsing cell (compartment $NC_1$), then first reaching one of the neighbouring cells (either of $NC_8$, $NC_9$ or $NC_{10}$) and then one of the other two ($NC_{11}$ or $NC_{12}$). The last two lines of the table present the final steps of the other two runs.

These observations based on the execution traces are strengthened or complemented by the verification properties described in the next section.

## C. Verification Results

In the following we present some of the results obtained verifying the systems described in Section III. When modelling a system, one can be interested in various properties such as:

- **Reachability:** is a certain configuration reachable? In the non-deterministic context it is interesting to find out if there is at least one path (computation) reaching a certain configuration or all the computations will reach it.
- **Boundedness:** in an SN P system, we say a neuron is *bounded* if, in any configuration starting from the initial one, the number of spikes is less than some positive integer.
- **Liveness:** for a rule it means there exists a number of spikes that can trigger it; for an SN P system, all the rules are *live*.
- **Deadlock-free**: progress can be made (the computation will not halt).
- **Steady-state**: the configurations are unchanging in time.
- **Traceability**: identification of computation paths. The idea of not being *traceable* could potentially signal *unreachability* or *deadlock*.

In order to illustrate these, the models presented in Section III have been selected such as to exhibit different properties, for example $\Pi_1, \Pi_2$ are bounded, while $\Pi_3, \Pi_4$ have some unbounded neurons. For example, $\Pi_3$ has $\sigma_3$ unbounded, while $\Pi_4$ has the $\sigma_3$ and the environment unbounded. These properties can be easily verified using their kP counterparts and looking into the number of $a$'s (spikes) in each compartment.

For these experiments the kP-Lingua files were automatically converted into SMV specifications and the NuSMV model checker was used to verify the properties presented in the Table I. For example, the boundedness of the models was checked by imposing certain limits on the number of spikes for different neurons, in any possible configuration (that corresponds to the number of $a$ objects in the compartments of the equivalent kP system). The first column shows the kP system model for which the property was verified, the second gives a property number, the third shows the property (as kP query and also in Linear Temporal Logic), and the last column the verification result.

The *reachability* of a certain configuration can be expressed using the *eventually* pattern. For example, $\Pi_2$ will reach the configuration $(0, 5, 1)$ and, as shown in Table I the corresponding property $k\Pi_{\Pi_2} - 2$ is true. At verification the model checker will return *true* or *false*, with a *counterexample*. The *traceability* can be easily tackled this way: one should write a formula negating that a certain configuration can be reached, and, if this is false, the model checker will return a counterexample, representing a computation path, that reaches the configuration.

For $\Pi_4$, Property 1 shows that the signalling molecules $a$ eventually reach components corresponding to pulsing cells at the right end of the pulse generator system (Fig. 4), i.e., $NC_9$, $NC_{10}$ and $NC_{11}$. Property 2 shows that the excess molecules propagate to $NEnv$ with some steady-state concentration of $a$. Property 3 shows that the number of signalling molecules in the components corresponding to pulsing cells ($NC_8$, $NC_9$, $NC_{10}$, $NC_{11}$, $NC_{12}$) is always bounded by the given threshold. Property 4 shows that the component $NC_1$, corresponding to sender cell, keeps producing signalling molecules.

## D. Extending kP Systems for Verification

**Unboundedness.** In order to show that a system is unbounded, one could try to express a relation between the number of objects in a compartment (corresponding to spikes in a neuron) and each computation step. For example, in $k\Pi_{\Pi_3}$ there is an equality between the objects in compartment $NC_3$ and the current step number.

One could add to a kP system a new rule counting the current step `s -> s, step`. Extending for example $k\Pi_{\Pi_3}$ with such a rule, added in a new compartment $NC_4$, would be useful for verification purposes. It is obvious that starting with an $s$ object in the initial multiset of $NC_4$, the *step* objects are accumulated (one per computation step), and the property `always NC3.a = NC4.step` could be verified.

**Traceability.** Apart from using a counterexample trace to determine the intermediate configurations leading to a certain desired configuration, a kP system model could be extended to record the rules applied, for example `{>=a} a-> a(C2), a(C3), r11` . The objects representing rule labels would accumulate and show at each step which rules are executed.

TABLE I
PROPERTY PATTERNS USED IN THE VERIFICATION EXPERIMENTS FOR $k\Pi_{\Pi_1} - k\Pi_{\Pi_4}$

| Model | Prop. | kP-Queries and corresponding LTL translations | Result |
|---|---|---|---|
| $k\Pi_{\Pi_1}$ | 1 | `never NC1.a > 1` | true |
| | | `!( F (NC1.a > 1 & pInS))` | |
| $k\Pi_{\Pi_1}$ | 2 | `eventually (NC1.a = 0 and NC2.a = 0) and NC3.a = 1` | true |
| | | `F (((NC1.a = 0 & NC2.a = 0) & NC3.a = 1) & pInS)` | |
| $k\Pi_{\Pi_1}$ | 3 | `always (NC1.a <= 1 and NC2.a <= 1) and NC3.a <= 1` | true |
| | | `G (((NC1.a <= 1 & NC2.a <= 1) & NC3.a <= 1) \| !pInS)` | |
| $k\Pi_{\Pi_1}$ | 4 | `always (NC1.a + NC2.a) + NC3.a = 1` | true |
| | | `G (((NC1.a + NC2.a) + NC3.a) = 1 \| !pInS)` | |
| $k\Pi_{\Pi_2}$ | 1 | `never NC1.a > 6` | true |
| | | `!(F (NC1.a > 6 & pInS))` | |
| $k\Pi_{\Pi_2}$ | 2 | `eventually (NC1.a = 0 and NC2.a = 5) and NC3.a = 1` | true |
| | | `F (((NC1.a = 0 & NC2.a = 5) & NC3.a = 1) & pInS)` | |
| $k\Pi_{\Pi_2}$ | 3 | `steady-state ( NC1.a = 0 and NC2.a = 5) and NC3.a = 1` | true |
| | | `F ( G (((NC1.a = 0 & NC2.a = 5) & NC3.a = 1) \| !pInS) & pInS)` | |
| $k\Pi_{\Pi_2}$ | 4 | `always (NC1.a <= 6 and NC2.a <= 5 ) and NC3.a <= 1` | true |
| | | `G (((NC1.a <= 6 & NC2.a <= 5) & NC3.a <= 1) \| !pInS)` | |
| $k\Pi_{\Pi_2}$ | 5 | `always (NC1.a + NC2.a) + NC3.a = 6` | true |
| | | `G (((NC1.a + NC2.a) + NC3.a) = 6 \| !pInS)` | |
| $k\Pi_{\Pi_3}$ | 1 | `never NC3.a > 6` | false |
| | | `!(F (NC3.a > 6 & pInS))` | |
| $k\Pi_{\Pi_3}$ | 2 | `eventually (NC1.a = 5 and NC2.a = 1) and NC3.a >= 1` | true |
| | | `F (((NC1.a = 5 & NC2.a = 1) & NC3.a >= 1) & pInS)` | |
| $k\Pi_{\Pi_3}$ | 3 | `steady-state (NC1.a = 5 and NC2.a = 1) and NC3.a >= 1` | true |
| | | `F ( G (((NC1.a = 5 & NC2.a = 1) & NC3.a >= 1) \| !pInS) & pInS)` | |
| $k\Pi_{\Pi_3}$ | 4 | `always (NC1.a <= 6 and NC2.a <= 1) and NC3.a >= 0` | true |
| | | `G (((NC1.a <= 6 & NC2.a <= 1) & NC3.a >= 0) \| !pInS)` | |
| $k\Pi_{\Pi_3}$ | 5 | `always (NC1.a + NC2.a = 6)` | true |
| | | `G ((NC1.a + NC2.a) = 6 \| !pInS)` | |
| $k\Pi_{\Pi_4}$ | 1 | `eventually (NC9.a > 0 and NC11.a > 0) and (NC12.a > 0 and NEnv.a > 0)` | true |
| | | `F (((NC9.a > 0 & NC11.a > 0) & (NC12.a > 0 & NEnv.a > 0)) & pInS)` | |
| $k\Pi_{\Pi_4}$ | 2 | `steady-state NEnv.a > 0` | true |
| | | `F ( G (NEnv.a > 0 \| !pInS) & pInS)` | |
| $k\Pi_{\Pi_4}$ | 3 | `always ((NC8.a<=5 and NC10.a<=5) and (NC9.a<=5 and NC11.a<=5)) and NC12.a<=5` | true |
| | | `G ((((NC8.a<=5 & NC10.a<=5) & (NC9.a<=5 & NC11.a<=5)) & NC12.a<=5) \| !pInS)` | |
| $k\Pi_{\Pi_4}$ | 4 | `never NC1.a < 3 or NC1.a > 3` | true |
| | | `!( F ((NC1.a < 3 \| NC1.a > 3) & pInS))` | |

The new extended kP system has the same dynamic as the initial one (constructed using Remark 2, such as to simulate the SN P model), but is enriched with additional objects. Obviously, the SN P model has only $a$ objects (spikes) and the new extended kP system is not equivalent to it, having other types of objects. However, the restriction to $a$ objects is equivalent and the new additions, i.e. counting steps, rule labels, could be beneficial for verification.

## V. Conclusions

In this paper, we have showed how to map some classes of SN P systems to the equivalent kP systems and to analyse systems dynamics using the kPWorkbench tool, which allows simulating and formally verifying the translated kP systems. We have demonstrated the applicability of our approach in some case studies, including an example system from synthetic biology.

As future work, we will investigate in a more systematic way various properties that are expressed using kPWorkbench for more complex case studies, as well as for other classes of SN P systems, e.g. SN P systems with delays. We will also work on fully automating the translations steps.

## Acknowledgment

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.

## References

[1] Păun, Gh, "Computing with membranes," Turku Centre for Computer Science, Tech. Rep., 1998. [Online]. Available: http://tucs.fi/publications/view/?pub_id=tPaun98a

[2] ——, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[3] Păun, Gh, *Membrane Computing - An Introduction*. Springer, Verlag, 2002.

[4] Păun, Gh, G. Rozenberg, and A. Salomaa, Eds., *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2009.

[5] B. Song, X. Zeng, M. Jiang, and M. J. Pérez-Jiménez, "Monodirectional tissue P systems with promoters," *IEEE Transactions on Cybernetics*, pp. 1–13, 2020.

[6] B. Song, K. Li, D. Orellana-Martín, L. Valencia-Cabrera, and M. J. Pérez-Jiménez, "Cell-like P systems with evolutional symport/antiport rules and membrane creation," *Information and Computation*, p. 104542, 2020.

[7] B. Song, X. Zeng, and A. Rodríguez-Patón, "Monodirectional tissue P systems with channel states," *Information Sciences*, vol. 546, pp. 206 – 219, 2021.

[8] M. Ionescu, Păun, Gh, and T. Yokomori, "Spiking neural P systems," *Fundamenta Informaticae*, vol. 71, no. 2–3, pp. 279–308, 2006.

[9] H. N. Adorna, F. G. C. Cabarle, L. F. Macías-Ramos, L. Pan, M. J. Pérez-Jiménez, B. Song, T. Song, and L. Valencia-Cabrera, "Taking the pulse of SN P systems: a quick survey," in *Multidisciplinary creativity*. Spandugino, 2015, pp. 3–16.

[10] H. Rong, T. Wu, L. Pan, and G. Zhang, "Spiking neural P systems: Theoretical results and applications," in *Enjoying Natural Computing*, 2018, pp. 256–268.

[11] X. Zeng, H.N. Adorna, M.Á. Martínez-del Amor, L. Pan, and M.J. Pérez-Jiménez, "Matrix representation of spiking neural P systems," in *CMC 2010*, ser. LNCS, vol. 6501. Springer, 2009, pp. 377–391.

[12] J.P. Carandang, J.M. Villaflores, F.G.C. Cabarle, H.N. Adorna, and M.Á. Martínez-del Amor, "CuSNP: Spiking neural P systems simulators in CUDA," *Romanian Journal of Information Science and Technology*, vol. 20, no. 1, pp. 57–70, 2017.

[13] H.N. Adorna, "Matrix representations of spiking neural P systems: Revisited," in *$20^{th}$ International Conference on Membrane Computing, CMC20*, 2019, pp. 227–247.

[14] L. F. Macías-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M. J. Pérez-Jiménez, and A. Riscos-Núñez, "A P-lingua based simulator for spiking neural P systems," in *CMC 2011*, ser. LNCS, vol. 7184. Springer, 2012, pp. 257–281.

[15] M.Á. Martínez-del Amor, D. Orellana-Martín, F.G.C. Cabarle, M.J. Pérez-Jiménez, and H.N. Adorna, "Sparse-matrix representation of spiking neural P systems for GPU," in *15th Brainstorming Week on Membrane Computing (BWMC)*, 2017, pp. 161–170.

[16] M. Gheorghe, F. Ipate, C. Dragomir, L. Mierlă, L. Valencia-Cabrera, M. García-Quismondo, and M.J. Pérez-Jiménez, "Kernel P Systems - Version I," in *11th Brainstorming Week on Membrane Computing (BWMC)*, 2013, pp. 97–124.

[17] S. Konur, L. Mierlă, F. Ipate, and M. Gheorghe, "kPWorkbench: A software suit for membrane systems," *SoftwareX*, vol. 11, p. 100407, 2020.

[18] M. Gheorghe, R. Lefticaru, S. Konur, I. Niculescu, and H. Adorna, "Spiking neural P systems - Matrix representation and formal verification," in *ICMC 2020*. Tech. Rep. of TU Wien (in preparation), 2020.

[19] C. Dragomir, F. Ipate, S. Konur, R. Lefticaru, and L. Mierlă, "Model checking kernel P systems," in *CMC 2013*, ser. LNCS, vol. 8340. Springer, 2013, pp. 151–172.

[20] "Case Studies," https://github.com/Kernel-P-Systems/kPWorkbench/wiki/Case-Studies.

[21] S. Basu, R. Mehreja, S. Thiberge, M.-T. Chen, and R. Weiss, "Spatiotemporal control of gene expression with pulse-generating networks," *PNAS*, vol. 101, no. 17, pp. 6355–6360, 2004.

[22] "kPWorkbench website:," https://github.com/Kernel-P-Systems/kPWorkbench.

[23] M.E. Bakir, F. Ipate, S. Konur, L. Mierlă, and I.-M. Niculescu, "Extended simulation and verification platform for kernel P systems," in *CMC 2014*, ser. LNCS, vol. 8961. Springer, 2014, pp. 158–178.

[24] M. E. Bakir, S. Konur, M. Gheorghe, I. Niculescu, and F. Ipate, "High performance simulations of kernel P systems," in *HPCC/CSS/ICESS 2014*, 2014, pp. 409–412.

[25] M. Gheorghe, S. Konur, F. Ipate, L. Mierla, M. E. Bakir, and M. Stannett, "An integrated model checking toolset for kernel P systems," in *CMC 2015*, ser. LNCS, vol. 9504. Springer, 2015, pp. 153–170.

[26] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV version 2: An open source tool for symbolic model checking," in *CAV 2002*, ser. LNCS, vol. 2404. Springer, 2002, pp. 359–364.

[27] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Soft. Eng.*, vol. 23, no. 5, pp. 275–295, 1997.

[28] S. Konur, "A decidable temporal logic for events and states," in *TIME'06*. IEEE, 2006, pp. 36–41.

[29] S. Konur, "An interval logic for natural language semantics," in *AiML*, 2008, pp. 177–191.

[30] ——, "A survey on temporal logics," *CoRR*, vol. abs/1005.3199, 2010.

[31] ——, "Real-time and probabilistic temporal logics: An overview," *CoRR*, vol. abs/1005.3200, 2010.

[32] ——, "Towards light-weight probabilistic model checking," *Journal of Applied Mathematics*, vol. 2014, no. 814159, pp. 1–15, 2014.

[33] R. Alur, K. McMillan, and D. Peled, "Model-checking of correctness conditions for concurrent objects," *Information and Computation*, vol. 160, no. 1-2, pp. 167 – 188, 2000.

[34] M. Yabandeh, "Model checking of distributed algorithm implementations," Ph.D. dissertation, EPFL, Lausanne, 2011.

[35] S. Konur and M. Fisher, "Formal analysis of a VANET congestion control protocol through probabilistic verification," in *VTC Spring 2011*. IEEE, 2011, pp. 1–5.

[36] S. Konur and M. Gheorghe, "A property-driven methodology for formal analysis of synthetic biology systems," in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, 2015, pp. 360–371.

[37] M. E. Bakir, S. Konur, M. Gheorghe, N. Krasnogor, and M. Stannett, "Automatic selection of verification tools for efficient analysis of biochemical models," *Bioinformatics*, vol. 34, no. 18, pp. 3187–3195, 2018.

[38] H. Abbink, R. V. Dijk, T. Dobos, M. Hoogendoorn, C. Jonker, S. Konur, P. Maanen, V. Popova, A. Sharpanskykh, P. V. Tooren, J. Treur, J. Valk, L. Xu, and P. Yolum, "Automated support for adaptive incident management," in *ISCRAM*, 2004, pp. 153–170.

[39] M. Arapinis, M. Calder, L. Denis, M. Fisher, P. Gray, S. Konur, A. Miller, E. Ritter, M. Ryan, S. Schewe, C. Unsworth, and R. Yasmin, "Towards the verification of pervasive systems," *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, vol. 22, p. 15, 2009.

[40] S. Konur, M. Fisher, S. Dobson, and S. Knox, "Formal verification of a pervasive messaging system," *Formal Aspects of Computing*, vol. 26, no. 4, pp. 677–694, 2014.

[41] F. Camci, O. F. Eker, S. Baskan, and S. Konur, "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*, vol. 230, no. 1, pp. 24–42, 2016.

[42] R. Lefticaru, S. Konur, Ü. Yildirim, A. Uddin, F. Campean, and M. Gheorghe, "Towards an integrated approach to verification and model-based testing in system engineering," in *iThings/GreenCom/CPSCom/SmartData 2017*. IEEE, 2017, pp. 131–138.

[43] R. Lefticaru, M. E. Bakir, S. Konur, M. Stannett, and F. Ipate, "Modelling and validating an engineering application in kernel P systems," in *CMC 2017*, ser. LNCS, vol. 10725. Springer, 2018, pp. 183–195.

[44] M. Gheorghe, R. Ceterchi, F. Ipate, and S. Konur, "Kernel P systems modelling, testing and verification - sorting case study," in *CMC 2016*, ser. LNCS, vol. 10105. Springer, 2017, pp. 233–250.

[45] M. Gheorghe, R. Ceterchi, F. Ipate, S. Konur, and R. Lefticaru, "Kernel P systems: from modelling to verification and testing," *Theoretical Computer Science*, vol. 724, pp. 45–60, 2018.

[46] S. Konur, M. Gheorghe, C. Dragomir, L. Mierlă, F. Ipate, and N. Krasnogor, "Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems," *ACS Synthetic Biology*, vol. 4, no. 1, pp. 83–92, 2015.

[47] S. Konur, M. Gheorghe, C. Dragomir, F. Ipate, and N. Krasnogor, "Conventional verification for unconventional computing: a genetic XOR gate example," *Fundamenta Informaticae*, vol. 134, no. 1–2, pp. 97–110, 2014.

[48] M. Gheorghe, S. Konur, and F. Ipate, "Kernel P systems and stochastic P systems for modelling and formal verification of genetic logic gates," in *Advances in Unconventional Computing: Volume 1: Theory*. Springer, 2017, pp. 661–675.