A Formal Framework for Spiking Neural P Systems [⋆]

Sergey Verlan[1], Rudolf Freund[2], Artiom Alhazov[3], Linqiang Pan[4,5]

[1] Université Paris Est, LACL (EA 4219), UPEC, F-94010, Créteil, France
verlan@u-pec.fr

[2] Faculty of Informatics, TU Wien, Favoritenstr. 9–11, 1040 Vienna, Austria
rudi@emcc.at

[3] Vladimir Andrunachievici Institute of
Mathematics and Computer Science
Academiei 5, Chişinău, MD-2028, Moldova
artiom@math.md

[4] Key Laboratory of Image Information Processing and Intelligent Control of
Education Ministry of China,
School of Artificial Intelligence and Automation,
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China

[5] School of Electric and Information Engineering
Zhengzhou University of Light Industry
Zhengzhou 450002, Henan, China, lqpan@mail.hust.edu.cn

Summary. We extend the formal framework for P systems exhibited in [3] by considering the applicability of each rule to be controlled by regular conditions. The aim of this extension is to express spiking neural P systems in a formal framework. Another goal of the extension is to incorporate the notions of input and output. We also show that in the case of spiking neural P systems rules have a rather simple form and in that way spiking neural P systems correspond to vector addition systems where the application of rules is controlled by regular expressions.

1 Introduction

Based on the biological background of neurons sending electrical impulses along axons to other neurons, spiking neural P systems were introduced in [5]. In spiking neural P systems, the contents of a cell – *neuron* – consists of a number of so-called *spikes*. The rules assigned to a cell allow for sending information to other neurons

[⋆] The ideas for this paper were discussed during a stay of Artiom Alhazov and Rudolf Freund in Paris at Créteil with Sergey Verlan in summer 2018, while Rudolf Freund was a guest professor supported by Université Paris Est Créteil.

in the form of spikes corresponding to electrical impulses, which are summed up in the target cells. The application of the rules depends on the current contents of the neuron and in the general case is described by regular sets.

As inspired from biology, the cell sending out spikes may be *closed* for a specific time period corresponding to the refractory period of a neuron; during this refractory period, the neuron is closed for new input and cannot get excited – *fire* – for spiking. As already shown in [4], considering such *delay* usually is not needed to obtain the desired results, hence, we will not consider this feature in the following.

In [1] an extended variant of spiking neural P systems was presented, allowing the rules to send different numbers of spikes along the axons to different neurons, depending on the rule applied in a cell. Since then, many variants of spiking neural P systems have been considered; we only mention a few ones in the following:

In the basic model for spiking neural P systems, each cell simultaneously with the other cells applies one applicable rule. This condition may be alleviated by considering *asynchronous* – any subset of the cells applies a rule – or *sequential* – only one cell applies a rule – spiking neural P systems. In [2], the number of spikes sent along an axon was allowed to be more than one.

A first overview on results for spiking neural P systems can be found in Chapter 13 of the Handbook [8]. A rather extensive bibliography on spiking neural P systems was published in the first volume of the *Bulletin of the International Membrane Computing Society* 2016, see [6].

In [3], a formal framework for P systems was developed. In this paper we continue this line of research by extending this formal framework to capture most of the basic features of spiking neural P systems.

2 Preliminaries

The set of all finite multisets over the set V is denoted by $\langle V, \mathbb{N} \rangle$. The set of finite languages over the alphabet T is denoted by $FIN(T)$, the set of regular languages over the alphabet T by $REG(T)$. The families of finite and regular languages over arbitrary alphabets are denoted by FIN and REG, respectively. We remark that regular expressions are a way to specify regular languages. The regular expressions over an alphabet T are denoted by $REGEX(T)$. For any $E \in REGEX(T)$, $L(E)$ denotes the regular language corresponding to the regular expression E, whereas $L^\circ(E)$ denotes the corresponding regular multiset language. We remark that if $|T| = 1$, then $L^\circ(E) = L(E)$.

The set of natural numbers, i.e., the set of non-negative integers, is denoted by $\mathbb{N}$. Moreover, $\mathbb{N}_\infty := \mathbb{N} \cup \{\infty\}$.

3 The Definition of the Framework

We extend the definitions from [3] in order to be able to handle conditions given by regular expressions. We also remark that our definition will only include spiking neural P systems without delays. While it is not difficult to integrate the notion of delay in the definition, it makes all corresponding notations much more complex. This is not very constraining as it was shown in [4] for any spiking neural P system with a delay there exists an equivalent system without delay. We would like to notice that the corresponding proofs are not constructive, so in practice it might be extremely difficult to transform a system with delays to a system without delays. Another remark concerns the use of forgetting rules. They are special in the spiking framework, as normally any spiking rule should produce at least one spike. In our framework there is no such restriction, so forgetting rules are no more differentiated and correspond just to λ-rules.

3.1 Basic Structure

We start with a new definition of an interaction rule.

Definition 1. *Let V be an alphabet and $n > 0$. An* interaction *rule of dimension n is defined as*

$$(X \to Y; E)$$

where $X = (x_1, \dots, x_n)$, $Y = (y_1, \dots, y_n)$, $x_i, y_i \in \langle V, \mathbb{N} \rangle$, $1 \le i \le n$, are vectors of multisets over V and $E = (E_1, \dots, E_n)$, where each $E_i \in REGEX(V)$, $1 \le i \le n$, i.e., the E_i are regular expressions over V.

Since in most of the cases the corresponding vectors are sparse, we will also use the notation

$$(1, x_1) \dots (n, x_n) \to (1, y_1) \dots (n, y_n); (1, E_1) \dots (n, E_n)$$

for a rule $(X \to Y; E)$, and for any $1 \le i \le n$, we will omit E_i if $L^\circ(E_i)$ equals the V° as well as x_i or y_i when it is equal to the empty multiset, whenever possible.

Next we extend the definition of a network of cells from [3] in order to incorporate input and output:

Definition 2. *A* network of cells of degree $n \ge 1$ *is a construct*

$$\Pi = (n, V, w, c_{in}, c_{out}, Inf, R)$$

where

1. *n is the number of* cells*;*
2. *V is a finite alphabet;*
3. *$w = (w_1, \dots, w_n)$, $w_i \in \langle V, \mathbb{N} \rangle$, for $1 \le i \le n$, is the* finite multiset initially associated to cell *i;*
4. *$c_{in} \subseteq \{1, \dots, n\}$ is the set of* input *cells;*
5. *$c_{out} \subseteq \{1, \dots, n\}$ is the set of* output *cells;*
6. *$Inf = (Inf_1, \dots, Inf_n)$, $Inf_i \subseteq V$, for $1 \le i \le n$, is the* set of symbols occurring infinitely often in cell *i (in most of the cases, only one cell, called the* environment*, will contain symbols occurring with infinite multiplicity);*

7. R is a finite set of interaction rules *of the form given in Definition 1.*

We also recall the definition of a configuration:

Definition 3 ([3]). *Consider a network of cells $\Pi = (n, V, w, c_{in}, c_{out}, Inf, R)$. A* configuration *$C$ of Π is an n-tuple of multisets over V $(u'_1, \dots, u'_n)$ with $u'_i \in \langle V, \mathbb{N}_\infty \rangle$, $1 \le i \le n$; in the following, C will also be described by its finite part C^f only, i.e., by $(u_1, \dots, u_n)$ satisfying $u'_i = u_i \cup {Inf_i}^\infty$ and $u_i \cap Inf_i = \emptyset$, $1 \le i \le n$.*

The next definition defines the applicability of a rule. It extends the corresponding definition from [3].

Definition 4. *We say that an interaction rule $r = (X \to Y; E)$ is* eligible *for the configuration C with $C = (u_1, \dots, u_n)$ if and only if for all i, $1 \le i \le n$, we have*

- *$x_i \subseteq u_i$ (x_i is a submultiset of u_i) and*
- *$u_i \in L^\circ(E_i)$ (u_i belongs to the regular multiset language described by the expression E_i).*

Moreover, we require that $x_j \cap (V - Inf_j) \neq \emptyset$ for at least one j, $1 \le j \le n$. This last condition ensures that at least one symbol appearing only in a finite number of copies is involved in the rule. The set of all rules eligible for C is denoted by $Eligible\,(\Pi, C)$.

We remark that the only difference with the previous definition is the replacement of permitting and forbidding conditions by regular expression checks.

The application of a group of rules and the definition of the derivation modes remains the same as in [3]. Below we recall some details.

Definition 5. *Let C be a configuration and Π a network of cells. Let R' be a multiset of rules from $Eligible(\Pi, C)$ (i.e., a multiset of eligible rules). Let*

$$X = \sum_{(X_r \to Y_r; E_r) \in R'} X_r.$$

We say that the multiset of rules R' is applicable *to C if $X \subseteq C$.*

So, in order for a multiset of rules to be applicable, each rule should be eligible and there should be enough objects in the configuration corresponding to the sum of all left-hand sides of the rules.

The set of all multisets of rules *applicable* to C is denoted by *Applicable* (Π, C).

Definition 6 ([3]). *Consider a configuration C and a multiset of rules $R' \in Applicable\,(\Pi, C)$. We define the configuration being the result of* applying *of R' to C as*

$$Apply\,(\Pi, C, R') = \left(C - \sum_{(X_r \to Y_r; E_r) \in R'} X_r + \sum_{(X_r \to Y_r; E_r) \in R'} Y_r \right).$$

Definition 7. *A derivation mode δ is a restriction of the set of multisets of applicable rules. For a system Π and a configuration C,*

$$Applicable\,(\Pi, C, \delta) \subseteq Applicable(\Pi, C)$$

denotes the set of multisets of rules of Π applicable to configuration C according to the derivation mode δ.

3.2 Computation and Input/Output

In [3] the computation is performed as a sequence of applications of applicable rules, starting from some initial configuration, until a halting condition is met (usually the total halting – when no rule is applicable anymore). In the spiking case, also a different transducer-like strategy is used to transform an input to an output. Since the definitions from [3] cannot handle this aspect, we would like to change the notion of computation in order to incorporate these special terms.

In an informal way, the computation starts with the initial configuration, in each step $t \geq 0$ it may use the contents of the input cells, which is generated by a recursive function *Input*, and in each step it also produces a result using a recursive function *Output*.

More formally, the input function has the following signature: $Input : \mathbb{N} \times 2^V \to V^n$. Hence, the value of $Input(t, c)$ is a vector of multisets of size n that gives the input to the system at each time $t \geq 0$. The second argument c defines the set of components of the system that may receive the input. All other components of the input vector are considered to be empty. We will omit c if it is clear from the context.

Definition 8. *A* computation *of the network of cells*

$$\Pi = (n, V, w, c_{in}, c_{out}, Inf, R)$$

in derivation mode δ and using the input function $Input : \mathbb{N} \times 2^V \to V^n$ is performed as follows:

$$\begin{aligned} C_0 &= w + Input(0, c_{in}) \\ C_{i+1} &= Input(i+1, c_{in}) + Apply(\Pi, C_i, R'),\ R' \in Applicable(\Pi, C_i, \delta) \end{aligned}$$

We remark that due to the definition of the function *Input*, the defined computation is infinite, even if $Applicable(\Pi, C_i, \delta)$ is empty. However, in most cases we consider only a finite portion of the computation (basically until no more rules are applicable).

Example 1. The standard variant of the computation used in P systems starts from an initial configuration and iterates the choice and the application of some multiset of rules. In this case, the input function is defined as follows: $Input(t, c) = \bar{0}$, *i.e.* it is always the zero vector. We should call this case *empty* input.

Example 2. A *spiking train input* is an input function that satisfies the following property:

$$\exists t_1, t_2 \geq 0,\ t_1 < t_2 : \begin{cases} \mathit{Input}(t,c) \neq \bar{0} & t \in \{t_1, t_2\} \\ \mathit{Input}(t,c) = \bar{0} & t \notin \{t_1, t_2\} \end{cases}$$

We also say that the value of the input is $t_2 - t_1$.

We remark that, according to the above definition, the input cells keep the non-consumed multisets from the previous steps. It is possible to use a different strategy by emptying cells c_{in} in every step. First we define an additional function $\pi(S,X) : 2^{\mathbb{N}} \times \langle V, \mathbb{N}\rangle \to \langle V, \mathbb{N}\rangle$ that empties all components of $X = (X_1, \ldots, X_n)$ that are not indicated in S (it is similar to a projection, where instead of deleting components their value becomes empty). Here is its formal definition:

$$\forall 1 \leq i \leq n,\ \pi(S,X)_i = \begin{cases} X_i, & \text{if } i \in S \\ 0, & \text{otherwise} \end{cases}$$

We will also use the notation $\pi(\neg S, X)$ in order to define the application of π to the complement of S.

Now we can define a computation with a *transient* input.

Definition 9. *A* computation *of the network of cells*

$$\Pi = (n, V, w, c_{in}, c_{out}, \mathit{Inf}, R)$$

in derivation mode δ with transient input and using the input function Input is performed as follows:

$$C_0 = w + \mathit{Input}(0, c_{in})$$
$$C_{i+1} = \mathit{Input}(i+1, c_{in}) + \pi(\neg c_{in}, \mathit{Apply}(\Pi, C_i, R')),\ R' \in \mathit{Applicable}(\Pi, C_i, \delta).$$

A system with a transient input is useful for some applications, e.g., for image processing or learning, as it becomes simpler to feed the data into the system.

The result of the computation, or the *output*, is defined as the result of the corresponding output function over the time series of the values of output cells. The output function has a memory and can decide a value based on the history of the computation, which means that the output is a time series, too. Moreover, the output function needs the description of the system and the derivation mode in order to correctly compute its result.

Formally, the signature of the output function is

$$\mathit{Output} : \mathbb{N} \times NC \times (V^n)^* \times 2^V \times \Delta \to S,$$

where NC is the family of all networks of cells, Δ is the set of all possible derivation modes and S is the set containing all possible elements of the output.

Suppose that $C_0, \dots, C_k, \dots$ is the computation of Π on the sequence of inputs $Input(t)$. Let us also denote $C(t) = C_0, \dots, C_t$. Then the result of the computation is given by the sequence

$$R(t) = Output(t, \Pi, C(t), c_{out}, \delta),\ t \geq 0.$$

The defined computation is infinite. We can restrict ourselves to finite computations by using an additional function transforming an infinite computation to a single result. A more elegant solution is to use the following convention – we expect the output function to produce a finite number of non-empty results: for any computation there exists a $t_h \geq 0$ such that $R(t)$ is empty for all $t > t_h$. Then the result is just the union of the corresponding values obtained from the first t_h computation steps.

Example 3. The traditional output in P systems yields the result of the output cell when a halting configuration is reached. This can be obtained using the following output function:

$$Output(t, \Pi, C(t), c_{out}, \delta) = \begin{cases} \pi(c_{out}, C_t) & \text{if } Applicable(\Pi, C_t, \delta) = \emptyset \\ \emptyset & otherwise \end{cases}$$

We remark that the result set in this case is the vector of multisets over V of dimension $|c_{out}|$. In the simplest case, when c_{out} is a single cell, the result is just the multiset that is contained in this cell.

We can further simplify the definition of the output function by observing that for finite computations we need to check for a halting condition, and if it is satisfied then we need to collect the result (in most of the cases yielding the contents of the output cells).

Example 4. According to the remark above, total halting can be also seen as an output function checking that there are no more rules applicable for the current configuration, and if it is the case, then it yields as the result the contents of the output cell(s).

Example 5. One of the traditional output strategies in spiking neural P systems is counting the time difference between two consecutive spikes in the output neuron. It can be obtained using the output function that for each time $t \geq 0$ checks if in the output cell there is a non-empty contents at time t and at some time $t' < t$, but the output cell is empty for all $0 \leq t'' < t, t'' \neq t'$. In this case it yields the value $t - t'$ as a result. A similar observation holds for the input case.

4 Spiking Neural P Systems

In the case of spiking neural P systems the definitions given in the preceding section can be simplified using the observation that the alphabet of the system contains a single object a, also called a spike. In this case, the contents of each cell is just a number and the whole configuration is a vector of numbers. Moreover, any regular expression over a one-letter alphabet corresponds to a semi-linear set of numbers. Hence, we can use the following form for spiking rules (we suppose to have k cells):

$$E/X, \text{ where } X \in \mathbb{Z}^k,\ E \in (2^{\mathbb{N}})^k. \tag{1}$$

Example 6. Consider the spiking rule $(a^2+a^3)^*/a^2 \to a$ in cell 1 connected to cells 2 and 3 (and the system having a total of 4 cells). Taking into account the above remark we can represent it as $(S, \mathbb{N}, \mathbb{N}, \mathbb{N})/(-2, 1, 1, 0)$, where $S = \{2n+3k \mid n, k \geq 0, n+k > 0\}$.

Such kind of rules can be used if the corresponding system does not have self-loops. In the converse case it is sufficient to transform the rules into normal form.

Definition 10. *A spiking neural P system is said to be in a normal form if the following conditions hold:*

- *for any rule $E/a^k \to a$ and any $x \in L^\circ(E)$, $|x| \geq k$,*
- *for any two rules $E_1/a^k \to a$ and $E_2/a^m \to a$ in the same neuron either $E_1 = E_2$ or $L^\circ(E_1) \cap L^\circ(E_2) = \emptyset$.*

The next theorem shows that any spiking neural P system can be transformed into this normal form.

Theorem 1. *For any spiking P system Π there exists a spiking neural P system Π' in normal form such that $L(\Pi) = L(\Pi')$.*

Proof. The first condition is easy to be satisfied as the language $\{x \in L^\circ(E) \mid |x| \geq k\}$ is regular and the applicability condition requires the cell to contain at least k spikes. For the second condition it is enough to observe that the languages $A = L^\circ(E_1) \cap L^\circ(E_2)$, $B = L^\circ(E_1) \setminus L^\circ(E_2)$, and $C = L^\circ(E_2) \setminus L^\circ(E_1)$ are regular. Then the corresponding rules can be replaced by $A/a^k \to a$, $A/a^m \to a$, $B/a^k \to a$ and $C/a^m \to a$.

Corollary 1. *Any spiking neural P system can be written using the notation given in Equation* (1)*.*

Example 7. Consider the spiking neural P system depicted on Fig. 1. It has an input neuron labeled by 1 and no output neurons. The system uses a spiking train input and a boolean output function based on the halting condition: with the first spike arriving in the input cell in the first step, the system starts spiking in a cycle of three with different configurations; it halts and thus outputs *true* if the difference between the two spikes of the input spiking train arrived with a time difference of 3 steps and thus forces the system to halt, whereas otherwise the system does not reach a halting configuration and thus we consider, as it is commonly done in the literature, that the corresponding output is *false*. This means that we here consider a recognizing spiking neural P system.

The rules of the system can be written as follows:

$1 : (\{1\}, \mathbb{N}, \mathbb{N}, \mathbb{N})/(-1, 1, 0, 1)$	
$2.1 : (\mathbb{N}, \{1\}, \mathbb{N}, \mathbb{N})/(0, -1, 1, 0)$	$2.2 : (\mathbb{N}, \{3\}, \mathbb{N}, \mathbb{N})/(0, -3, 0, 0)$
$3.1 : (\mathbb{N}, \mathbb{N}, \{1\}, \mathbb{N})/(0, 0, -1, 1)$	$3.2 : (\mathbb{N}, \mathbb{N}, \{3\}, \mathbb{N})/(0, 0, -3, 0)$
$4.1 : (\mathbb{N}, \mathbb{N}, \mathbb{N}, \{1\})/(1, 0, 0, -1)$	$4.2 : (\mathbb{N}, \mathbb{N}, \mathbb{N}, \{2\})/(1, 0, 0, -2)$

We observe that the system recognizes input sequences of the form $0^*1(000)^k10^\omega$, $k \geq 1$, i.e., the time between the two input spikes represents a number divisible by 3:

As long as the input is 0, the 4-vector describing the configuration is $(0, 0, 2, 0)$. With the first spike arriving in neuron 1, we get the following sequence of configurations using the spiking rules in the four neurons, assuming $3k$ time steps, $k \geq 1$, until the second spike arrives in neuron 1:
$(1, 0, 2, 0)$
$(0, 1, 2, 1)$
$(0, 1, 3, 0)$
$(0, 0, 1, 0)$
Then the system loops in neurons 2, 3, and 4 as follows:
$(0, 0, 0, 1)$
$(0, 1, 0, 0)$
$(0, 0, 1, 0)$
Finally, when the second spike arrives in neuron 1 at the right time, we end up with the following sequence, which finally yields a halting configuration:
$(1, 0, 0, 1)$
$(0, 2, 0, 1)$
$(0, 3, 0, 0)$
$(0, 0, 0, 0)$

The interested reader may verify that in all other cases where the time between the two spikes arriving in the input neuron 1 is not divisible by 3 yields a non-halting computation.

5 Extensions

In this section we briefly discuss some extensions of the basic model.

5.1 Extended Rules

A natural extension of spiking neural P systems is the mode that uses extended rules. There are two basic types of such models:

The first one, see [2], considers rules of form $E/a^m \to a^n$. When applied, each of the connected cells will receive a^n spikes in the next step. It is easy to see that

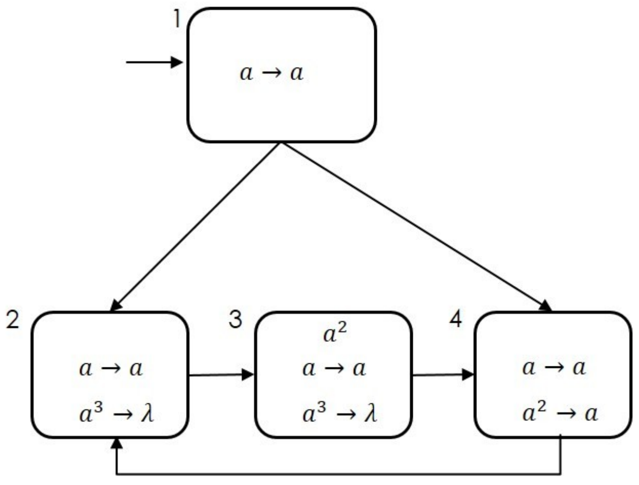

Fig. 1. A spiking neural P system recognizing numbers divisible by 3.

this can be simulated using general rules as follows (we suppose that there are 2 other cells connected to the cell where the rules are applied):

$$(E, \mathbb{N}, \mathbb{N})/(-m, n, n), \quad \text{see Fig. 2 (a).}$$

The second basic type of extended rules considers weights attached to synapses. Then each cell will receive the number of spikes corresponding to the indicated weight. Using general rules it can be simulated as follows:

$$(E, \mathbb{N}, \mathbb{N})/(-m, n, k), \quad \text{see Fig. 2 (b).}$$

Using general rules as introduced in [1], each rule may cause different numbers of spikes sent along the neurons to the connected cells. A similar simpler concept is considered in [7] as multiple channels spiking neural P systems. A simple example of this third type of extensions is depicted in Fig. 2 (c).

5.2 Non-natural Numbers of Spikes

It is obvious that the vectors of natural numbers used to represent rules and the configurations can be replaced by integer, real or complex vectors. This allows for using quite powerful transformations. In this case, the regular expressions/semi-linear sets of numbers cannot be used anymore to check for rule applicability. Hence, predicates over corresponding domains have to be used.

There are several models where the number of spikes is not a natural number. The best known one is the model using anti-spikes, for example see one recent paper [9]. It can be directly simulated with general rules working with integer

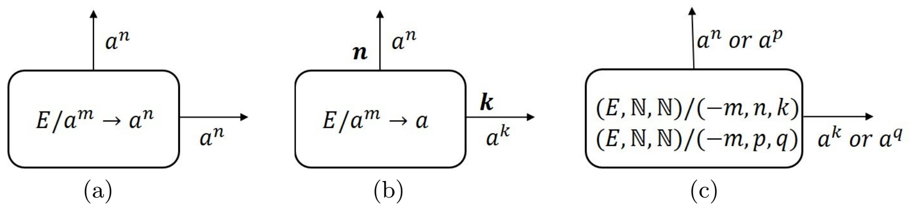

Fig. 2. Different extended spiking rules: (a) extended; (b) weights on synapses; (c) generalized rules from formal framework.

vectors. In fact, since the annihilation rule has priority over all other rules, in a neuron there can be only spikes or only anti-spikes, which corresponds to having positive or negative integers.

Another model [10] uses a variant of extended rules with real weights on synapses. This can be simulated using real vectors and equality predicates on real numbers instead of regular expressions.

5.3 Astrocytes

In spiking neural P systems with astrocytes two networks are interleaving – a normal spiking neural P system and a second network of cells interacting with the axons of the first one. An astrocyte senses several axons. Depending on the number of spikes sent through these axons, the number of spikes then allowed to pass through each of these axons is determined, For example, given an upper bound k for the sum of spikes to pass through these axons, either all spikes may pass, whereas otherwise they are removed and no spike will reach the target cells. There are many variants: (a) upper bound or (b) lower bound for the spikes to let them pass, and more general, (c) a function of the number of spikes wanting to pass and the number then allowed to pass, which can be done in an enforcing way or in an inhibitory behavior way.

This behavior can be directly modeled using general rules. The simulation is based on the observation that in each step only one rule per neuron can be applied (and there is a finite number of rules per neuron), so it is possible to know in advance how many spikes are generated by chosen rules.

For example, consider two axons leaving from neuron 1 and an astrocyte sensing these two axons going to neurons 2 and 3. The application of a rule

$$(E_1, \mathbb{N}, \mathbb{N})/(-m, p, q)$$

without the astrocyte controlling the axons would describe the application of a spiking rule in neuron 1 consuming m spikes, with $m \in E_1$, and sending p spikes to cell 2 and q spikes to cell 3. Using a simple astrocyte with lower bound k, i.e., only allowing the spikes to pass along the axons if their sum is at least k, this can be expressed by the rules

$$\begin{array}{ll}(E_1, \mathbb{N}, \mathbb{N})/(-m, p, q) & \text{if } \; p + q \geq k, \text{ and} \\ (E_1, \mathbb{N}, \mathbb{N})/(-m, 0, 0) & \qquad \text{otherwise.}\end{array}$$

5.4 Families of Control Languages

Instead of regular languages (given by the corresponding regular expressions) for controlling the application of rules, other classes of formal languages can be used, for example, subregular classes. In many variants of spiking neural P systems considered so far, $FIN(\{a\}) \cup \{a\}^*$ is sufficient, for example, see [4]. With $FIN(\{a\})$, usually only semi-linear sets can be obtained.

6 Conclusion

The proposed generalization of spiking allows us to describe many spiking-based models in a uniform way. As the formal framework for static P systems [3], this may open many directions for future research, including the comparison between different spiking models and the introduction of new features. As possible examples we would cite using spiking with real numbers as well as the probabilistic execution of the system.

Another conclusion that can be drawn using our formalization is that spiking neural P systems in fact can be seen as working in the sequential derivation mode, without any parallelism, which can easily be argued as follows: since the complement of $L(E)$ is regular, it is possible to construct complementary regular expressions for every rule. Then it is possible to write a series of rules each of them corresponding to the action of any combination of the initial rules (using the corresponding regular expressions). More precisely, for any combination of single rules from each neuron, it is possible to construct a single general rule that will check using the regular expressions that the chosen rules are applicable. Moreover, if in the normal form there are no identical expressions for any two rules in a cell, then the resulting system is deterministic (as it is possible for each rule to extend the regular expression with the negation of all other regular expressions and verify that any other rule is not applicable).

References

1. Alhazov, A., Freund, R., Oswald, M., Slavkovik, M.: Extended spiking neu- ral P systems. In: Hoogeboom, H.J., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers. Lecture Notes in Computer Science, vol. 4361, pp. 123-134. Springer (2006). https://doi.org/10.1007/11963516 8
2. Chen, H., Ionescu, M., Ishdorj, T.O., Păun, A., Păun, Gh., Pérez-Jiménez, M.: Spiking neural P systems with extended rules: Universality and languages. Natural Computing 7(2), 147-166 (2008)

3. Freund, R., Verlan, S.: A formal framework for static (tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 4860, pp. 271-284. Springer (2007). https://doi.org/10.1007/978-3-540-77312-2 17
4. Ibarra, O.H., Păun, A., Păun, Gh., Rodríguez-Patón, A., Sosík, P., Woodworth, S.: Normal forms for spiking neural P systems. Theor. Comput. Sci. 372(2-3), 196-217 (2007). https://doi.org/10.1016/j.tcs.2006.11.025
5. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71(2-3), 279-308 (2006), http://content.iospress.com/articles/fundamenta-informaticae/71-2-3-08
6. Pan, L., Wu, T., Zhang, Z.: A bibliography of spiking neural P systems. Bulletin of the International Membrane Computing Society 1, 63-78 (2016)
7. Peng, H., Yang, J., Wang, J. ,Wang, T., Sun, Z., Song, X., Luo, X., Huang, X.: Spiking neural P systems with multiple channels. Neural Networks 95, 66-71 (2017). https://doi.org/10.1016/j.neunet.2017.08.003
8. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England (2010)
9. Song, X., Wang, J., Peng, H., Ning, G., Sun, Z., Wang, T., Yang, F.: Spiking neural P systems with multiple channels and anti-spikes. Biosystems 169-170, 13-19 (2018). https://doi.org/10.1016/j.biosystems.2018.05.004
10. Wang, J., Hoogeboom, H.J., Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with weights. Neural Computation 22(10), 2615-2646 (2010). https://doi.org/10.1162/NECO a 00022