Spiking Neural P System Models as Formal Framework Models

Ren Tristan A. de la Cruz

February 26, 2021

1 Background

Membrane computing is a field of computer science that studies models of computation known as *P systems*. P systems refer to a family of models of computation that are inspired by different biological processes. P systems use biological concepts like cells, cell membranes, neurons, tissues, etc. Rules (computing operations) in P systems are inspired by biological processes like chemical reactions in cells, ion transport between regions divided by membranes, membrane creation, division and dissolution, spiking of neurons, neurogenesis and synaptogenesis, etc. P systems are parallel and distributed models of computation. This means a P system can apply multiple rules at the same time and these rules maybe applied in different parts or components of the P system.

Most P systems use multisets of object symbols as computing elements. One can think of these symbols as abstraction of things like molecules or ions. The multisets of symbols are compartmentalized into regions that are defined by the membranes that enclose them. This is the reason why the field is known as 'membrane computing'. Regions can be connected to each other. In cell-like P systems, membranes inside the cell can be nested. For example, if there are two membranes in a cell, membrane 0 and membrane 1, and membrane 1 is inside membrane 0 then the region enclosed by membrane 1 is connected to the region enclosed by membrane 0 that is outside membrane 1. In tissue-like P systems, membranes represent cells and each cell enclosed one region. Cells (and hence regions) can be connected to each other via channels. Tissue P systems form networks of cells. In general, the *configuration* of a P system refers to the network of cells or network of membranes where each cell or membrane contains a multiset of objects. In some P system models, a cell/membrane has a state or status that is different from its multiset content. For example, in some cell-like P systems, a membrane has a state known as *polarity* or *charge* which can either be negative, positive, or neutral.

There are many diverse P system models. One reason for the diversity of P systems is the diversity of the types of rules used by the systems. There are many different types of rules and a P system model can use more than one type. A rule is an operation that transforms the configuration of a P system. One can look at a rule as having two components: (1) a set of conditions the configuration of a P system has to meet for the rule to be eligible for application, and (2) a set of actions (transformations) the rule will apply to the P system's configuration. Different types of rules have different sets of conditions and different actions since they are inspired by different biological processes. For example, one common type of rule is called an *evolution rule*. An evolution rule is associated with a region and it consumes a multiset of objects in that region then produces another multiset of objects. Evolution rules are inspired by chemical reaction occurring inside cell membranes. Another example of a rule type is the *communication rule*. In cell-like P systems, a communication rule is associated with a membrane and it facilitates a transfer or exchange of multisets of objects between the region inside the membrane and the region outside. Communication rules are inspired by the ion transfers between regions via ion channels on cell membranes. There are also types of rules that transform the network of cells/membranes itself and not just the contents of the cells/membranes. For example, there are rules that can create and delete channels between cells. There are also rules that can create and delete membranes/cells. These rules are inspired by processes like cell/membrane division, membrane dissolution, neuron creation, synapse creation and deletion, etc.

P systems are parallel models so it is possible for the systems to apply different rules at the same time and they can also apply a rule multiple times. For example, evolution rules are inspired by chemical reactions and it is possible for different chemical reactions to occur multiple times inside cell membranes. P systems have a set of criteria that specifies which combination of eligible rules are valid. This set of criteria is called the *derivation mode*. Aside from the types of rules used by P systems, different P

systems can also use different derivation modes. Derivation modes can describe the type of parallelism the P system is working on. For example, the derivation mode known as *maximal parallelism* only allows combinations of eligible rule instances that are *maximal*. In a *maximal* combination of rule instances, one can not add any additional instance of eligible rules. The maximal combination of rules will consume enough symbols such that the remaining multiset of symbols does not allow any more rules to be applied. In a P system working in maximally parallel mode, any combinations of eligibles rules that are not maximal are not used. Another common derivation mode is the *minimally parallel* mode. In P systems working in minimally parallel mode, cells/membranes with eligible rules can still apply rules in parallel but each cell/membrane can only apply at most single instance of an eligible rule. Derivation modes can also describe rule priorities. For example, in a P system with two types of rules, type A and type B, one can specify in the model that if any type A rule in a membrane is eligible then no (eligible) type B rules can be applied. In the situation where there is an eligible type A rule in a membrane, any rule combination that includes an eligible type B rule in the same membrane is not valid.

When one combines the different types of rules and the different derivation modes, one can produce a significant number of diverse P system models. In order to help make sense of the different P system models, a *formal framework* was introduced in 2007 by Freund et al. The idea behind the formal framework (for P systems) is to have a set of general concepts and constructs that can be used when analysing most, if not all, types of P systems. The first version of the formal framework can be found in [8] and it is a framework for static tissue-like P system.

A system in the formal framework is called a *network of cells*. Similar to most P systems, the configuration of a network of cells is defined by the multiset contents of the cells of the system. The framework has a single type of rule called an *interaction rule*. An *interaction rule* is a rule type that generalizes most rule types used by cell-like and tissue-like P systems. Most variants of communication rules and evolution rules are special cases of the more general interaction rule. Different types of P system rules can have different forms but can also have different syntax. By formulating P systems as network of cells and using the interaction rule form, P system rules are written as interaction rules which means they are written to have the same general form and a common syntax. The interaction rule form makes comparing rules of different P systems easier. For example, two rule types from two different P systems may superficially appear different, due to different syntax, but when they are written as interaction rules it appears that their eligibility criteria and their actions are actually very similar. In the formal framework, the effect of an interaction rule when applied to a configuration is formally defined. This is not necessarily the case for other P systems and their rules. In some P system models, the effect of a rule on the configuration is described in a less formal manner (describe in plain English) which is prone to different interpretations. If one is to write a rule type from one P system model as an interaction rule, one has to formally define the meaning of the rule. The framework helps one clarify the meaning of the rule.

Aside from a general form of a rule, the framework in [8] also lists common derivation modes used by most P systems. The meaning of those different derivation modes are formally defined. In some P systems, derivation modes are less formally defined which again can lead to multiple conflicting interpretations. The framework lists other possible derivation modes that are rarely or not yet used by existing P system models which means they can be used to formulate new P system models.

The formal framework from [8] is limited to static P system. An interaction rule from [8] can not directly represent a P system rule that changes the structure of the network of cells, a rule that can create or delete cells or the connections between cells. The second version of the formal framework [7] has a much more general form of an interaction rule. The interaction rule from [7] can be used to write rules that change the structure of the network of cells. The rule form is much more complicated than the form of the interaction rules in [8].

A third version of the formal framework can be found in [29]. This version is an extension of the first version. The main differences between the first and the third version are: (1) the interaction rule in the third version allows multiset pattern checking and (2) the third version framework introduces the input and output constructs. In the first version of the framework, the criteria for eligibility of interaction rules are combinations of the presence of certain multisets in specified regions and the absence of certain multisets in specified regions. In the third version, the criteria for eligibility of an interaction rule rely on *control languages* (the pattern). For example, if certain multisets in specified regions are in the control languages of those regions (the multisets 'fits the pattern' for those regions), then the rule is eligible. Control languages and the input and output constructs are needed to represent P systems known as *spiking neural P systems* (SNP systems). SNP systems use a type of rule that checks for patterns. Such rules need the concept of a control language in order for them to be represented in the framework. SNP

systems are also usually used as transducers so it makes sense for the new framework to have input and output constructs.

The formal frameworks can be used to understand the functioning of P system variants. This is done by translating concepts from a P system model to concepts in the framework. Since the concepts in the framework are formally defined, the process of translating P system concepts to the language of the framework forces one to clarify and formalize those concepts that may have been vaguely described in the P system model. This process is particularly helpful when analysing P system models with vague semantics. The process highlights the concepts (e.g. rule semantics) that may have different interpretations. The formal frameworks can also be used as a common language for comparing different P systems. By translating different P systems as network of cells in the formal framework, the different P systems will have the same general form and will use the same syntax. This makes the comparison of the models a significantly easier task. The formal frameworks can also be use to extend P system models with new features. Since the constructs in the formal framework can be seen as generalized versions of constructs in most P system models (i.e. the formal framework's interaction rule is a generalized P system rule), one can add features available in the formal framework constructs as new features in new P system models. Some of these applications of the formal framework can be found in [28].

This work is focuses on the use of the formal framework concepts and constructs for representing spiking neural P system models.

Neural-like P systems known as *spiking neural P systems* (SNP systems) were introduced in [12]. They are inspired by the workings of a network of spiking neuron. A biological neuron has an electric charge. If the neuron's charge changes and passes a certain threshold, the neuron will 'spike' and it will send a signal to other connected neurons. This is where the term 'spiking neuron' came from. SNP systems use one type of object called a *spike*. An SNP system's neuron stores a multiset of spikes. The multiset of spikes in an SNP system's neuron is the analogue of the electric charge in biological neurons. The spiking behavior of an SNP system's neuron is controlled by a set of rules. The rules look at the multiset of spikes in the neuron and determine if the neuron should spike or not. These rules are the analogue of the threshold mechanism in biological neurons. An SNP system is a network of such neurons. The neurons are connected to each other using *synapses*. We will refer to this model (from [12]) as the *classic* or *standard* SNP system since there are already a lot of SNP system variants that have been introduced.

The standard SNP system is based on an simplified and highly abstracted mechanisms of a spiking neuron. There are many features and mechanism in the biological neural system that are good sources of inspiration for the creation of new SNP system variants. Similar to the standard SNP systems, other SNP system variants take some features or mechanisms from the biological neural system, abstract and simplify them, and then incorporate the simplified features/mechanisms in the models. Some examples of these variants are: SNP systems with weighted synapses [31, 19], SNP systems with inhibitory rules [21], SNP systems with astrocytes [20, 18], SNP systems with neuron division, budding, and dissolution [30, 16, 36], SNP systems with dynamics synapses [3, 2], SNP systems with polarizations [32], and SNP systems with thresholds [34].

Other SNP system models can be almost exactly similar to an existing SNP system model but only having different derivations modes. For example, the standard SNP system works in minimally parallel mode while the following SNP models are almost exactly similar to standard SNP system except for working on a different derivation modes: SNP systems in [11] use exhaustive mode, SNP systems in [4] use asynchronous mode, SNP systems in [10] use sequential mode, SNP systems in [25] use asynchronous mode with local synchronization, and SNP systems in [35, 13] use a generalized derivation mode.

SNP system variants are different from each other because they have different features (i.e. different types of rules), different derivation modes, or a combination of both. Aside from being inspired by neural system features and mechanisms, some SNP system variants also adopt features from other (non-SNP) P systems or get inspiration from other physical or biological phenomena. Other SNP system variants can be found in [5, 1, 6, 15, 26, 14, 33, 24, 17, 27]

As mentioned above, the third version of formal framework [29] is an extension of the first version and the modifications/extensions are made in order to represent SNP systems. In [29], after defining and describing the constructs of the framework, the authors translated the standard SNP system model (with rules without delay) as formal framework's network of cells. Standards SNP system's rules are translated as interaction rules. The authors also translated rules of other SNP systems variants [5, 19, 1, 22, 18] as interaction rules.

Section 2 Section 3 Section 4 Section 4.2 Section 4.3 Section 4.4 Section 4.5 Section 5

2 Preliminaries

This section present preliminary concepts used by SNP system models and the formal framework.

The following sets will be commonly used throughout the document: $\mathbb{N} = \{0, 1, 2, 3, ...\}$, $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$, $[1..n] = \{1, ..., n\}$, $2^{[1..n]} = \mathcal{P}([1..n])$ (power set of $[1..n]$).

Let V be a set of symbols called an *alphabet*. A *string* or *word* over V is a sequence of symbols from V. The *empty string* ϵ is a string without symbols, an empty sequence. A *string language* over V is a set of strings over V. The set of all strings over V is denoted by V^*.

A *multiset* over V is a function of the form $m : V \to \mathbb{N}_\infty$ while a *finite multiset* over V is a function of the form $m : V \to \mathbb{N}$. If m is a multiset over V and $a \in V$, $m(a)$ denotes the number of occurrences of symbol a in multiset m. If $V = \{v_1, ..., v_k\}$ and m is a finite multiset over V, m can be represented by the string ${v_1}^{m(v_1)} \cdots {v_k}^{m(v_k)}$. The size of a multiset m over V is $|m| = \sum_{v \in V} m(v)$. An *empty multiset* $\emptyset$ is any multiset of size 0. An *infinite multiset* is a multiset with an infinite size. i.e. Multiset m over V is infinite if for some $v \in V$, $m(v) = \infty$. A *multiset language* over V is a set of multisets over V. The set of all multisets over V is denoted by V°.

Let $V = \{v_1, ..., v_k\}$, m be the multiset ${v_1}^{m(v_1)} \cdots {v_k}^{m(v_k)}$ and n be the multiset ${v_1}^{n(v_1)} \cdots {v_k}^{n(v_k)}$. $m \subseteq n$ if and only if for all $v \in V$ $m(v) \leq n(v)$. $m + n$ is the multiset ${v_1}^{m(v_1)+n(v_1)} \cdots {v_k}^{m(v_k)+n(v_k)}$. If $m \subseteq n$, $n - m$ is the multiset ${v_1}^{n(v_1)-m(v_1)} \cdots {v_k}^{n(v_k)-m(v_k)}$

The set of all n-vectors whose components are finite multisets over V is denoted by $V^{\circ n}$. Let $X = (x_1, ..., x_n), Y = (y_1, ..., y_n) \in V^{\circ n}$. $X \subseteq Y$ if and only if $x_i \subseteq y_i$ for $1 \leq i \leq n$. $X + Y = (x_1 + y_1, ..., x_n + y_n)$. If $X \subseteq Y$, $Y - X = (y_1 - x_1, ..., y_n - x_n)$. An n-vector whose components are all empty multisets is denoted by $\emptyset_n$. i.e. $\emptyset_n = (\emptyset, ..., \emptyset)$. i

A *family of languages* is a set of languages. It can either be a family of string languages or a family of multiset languages. The notations $\mathscr{F}$ and $\mathscr{F}^\circ$ will be used for a family of string languages and a family of multiset languages, respectively. The notation $\mathscr{F}(V)$ will be used for a family of string languages over alphabet V while $\mathscr{F}^\circ(V)$ will be used for a family of multiset languages over alphabet V. For example, REG is the set regular string languages, $REG(V)$ is the set of regular string languages over V, REG° is the set of regular multiset languages, and $REG^\circ(V)$ is the set of regular multiset languages over V.

A *regular expression over alphabet* V is a string over the alphabet $V_{reg} = V \cup \{*, (,), \cup, |\}$ with a particular form. The regular expression defines a string language over V. Any language definable using a regular expression is called a *regular language*. The set of all regular expressions over V can be defined recursively. The empty string ϵ is a regular expression and it defines the empty language $\emptyset$. If $v \in V$, then v is a regular expression that defines the language $\{v\}$. Let E be a regular expression, $L(E)$ denotes the languages defined by regular expression E. If E_1 and E_2 are regular expressions over V, then $E_1 \cup E_2$ is a regular expression over V that defines the language $L(E_1) \cup L(E_2)$, $E_1|E_2$ or simply E_1E_2 is a regular expression that defines the language $\{w_1w_2 \mid w_1 \in L(E_1),\ w_2 \in L(E_2)\}$, (E_1) is a regular expression that similar to E_1 also defines the language $L(E_1)$, and E_1*, written as ${E_1}^*$, is a regular expression that defines the language $\{w^n \mid w \in L(E_1),\ n \in \mathbb{N}\}$. An *extended regular expression* over V is as string over $V_{exreg} = V \cup \{*, (,), \cup, |, ', \cap\}$. The set of extended regular expressions over V is defined in a similar manner to the recursive definition of the set of regular expression over V but with the following additional rules. If E_1 and E_2 are extended regular expressions over V, then $E_1 \cap E_2$ is an extended regular expression that defines the language $L(E_1) \cap L(E_2)$ and E_1' is an extended regular expression that defines the language $V^* \backslash L(E_1)$, the complement of the language $L(E_1)$. Extended regular expressions also define regular languages. A language defined by an extended regular expression can also be defined by a non-extended regular expression. The complement symbol/operator ($'$) and the intersection symbol/operator ($\cap$) are added in order to make expressions that define languages more succinct. A regular language defined by some regular expression E_1 can possibly be defined by a shorter extended regular expression E_2. For the rest of the document, the term 'regular expression' will be used for both extended and non-extended regular expressions.

Since multisets can be represented as strings, regular expressions can also be used to define *regular multiset languages*. A regular multiset language over V can be defined by a regular expression over V of a particular form. Given the alphabet $V = \{v_1, ..., v_n\}$, a regular expression that defines a multiset language has the form $E_1|E_2|E_3| \cdots |E_n$ where, for $1 \leq i \leq n$, E_i is a regular expression over $\{v_i\}$. A regular expression $E = E_1|E_2| \cdots |E_n$ defines the multiset language $\{m \mid$ string representation of m is in $L(E)\}$. $L^\circ(E)$ denotes the multiset language defined by the regular expression E.

3 Formal Framework for Spiking Neural P Systems

This section presents the constructs from the third version of the formal framework [29] and some constructs from the first version of the framework [8] that were no longer presented in [29].

Definition 1. [Configuration] An n-degree *configuration* $C = (u_1, ..., u_n)$ over alphabet V is an n-vector of multisets over V. A configuration C is called a *finite configuration* if all the components are finite multisets. A configuration is referred to as a *full configuration* to specify that the configuration can contain infinite multisets.

In the context of a *network of cells* (defined in full in Definition 6), an n-degree configuration $C = (u_1, ..., u_n)$ represents n cells and the component u_i is the multiset of objects contained in *cell* i.

Definition 2. [Interaction Rule] An n-degree *interaction rule* over alphabet V is the construct $(X \rightarrow Y; K)$ where $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$ are n-vectors of multisets over V and $K = (k_1, ..., k_n)$ is an n-vector of multiset languages.

A interaction rule acts on a configuration by consuming multisets of objects from the configuration and also producing multisets of objects in the configuration given that a certain set of conditions is met. The n-vector X contains the multisets $x_1, ..., x_n$ that the rule will 'consume' while the n-vector Y contains the multisets $y_1, ..., y_n$ that the rule will produce. The n-vector K contains multiset languages $k_1, ..., k_n$, called *control languages*, that are used to check a rule's *eligibility* (in Definition 3) with respect to some configuration.

A rule can also be written as:

$$(1, x_1) \cdots (n, x_n) \rightarrow (1, y_1) \cdots (n, y_n) \ ; \ (1, k_1) \cdots (n, k_n)$$

where any component (i, x_i), or (i, y_i), can be omitted if $x_i = \emptyset$, or $y_i = \emptyset$, and any component (i, k_i) can be omitted if $k_i = V^\circ$. To have a more compact notation, components (i, x_i), (i, y_i), (i, k_i) can be written as $[x_i]_i$, $[y_i]_i$, and $[k_i]_i$, respectively. Additionally, the control language components of the rule can be written first followed by a slash instead of a semicolon then the X and Y components. The rule can be written as:

$$[k_1]_1 \cdots [k_n]_n \ / \ [x_1]_1 \cdots [x_n]_n \rightarrow [y_1]_1 \cdots [y_n]_n$$

The rule syntax above will be used since it is closer to the syntax of most rules in different SNP system models. It is organized in such a way that all the components of a rule on the left side of the arrow, $[k_1]_1...[k_n]_n/[x_1]_1...[x_n]_n$, are the criteria for rule eligibility while all the components on the right side of the slash, $[x_1]_1 \cdots [x_n]_n \rightarrow [y_1]_1 \cdots [y_n]_n$, represent the actions that will be performed if the rule is applied. The $[x_1]_1...[x_n]_n$ components specify both eligibility criteria and actions of the rule.

Definition 3. [Rule Eligibility] Let $C = (u_1, ..., u_n)$ be an n-degree configuration over V and $r = ((x_1, ..., x_n) \rightarrow (y_1, ..., y_n); (k_1, ..., k_n))$ be an n-degree rule over V, rule r is *eligible* with respect to configuration C if the following conditions hold: (1) for all x_i, $x_i \subseteq u_i$ and (2) for all u_i, $u_i \in k_i$.

Since the rule consumes the multisets $x_1, ..., x_n$ from configuration C (x_i is consumed from cell i), the first condition $x_i \subseteq u_i$ states that (for all cells) content u_i of cell i should have enough objects in order for the rule to consume the multiset x_i of objects from the cell. The second condition simply checks if the content u_i of cell i is in the control language k_i associated with the cell.

Definition 4. [Applicability of a Multiset of Rules] Let R' be a multiset of n-degree rules over V and $C = (u_1, ..., u_n)$ be an n-degree configuration over V, R' is *applicable* to configuration C if the following conditions hold: (1) each rule $(X_j \rightarrow Y_j; K_j) \in R'$ is eligible with respect to configuration C and (2)

$$X = \left(\sum_{(X_j \rightarrow Y_j; K_j) \in R'} X_j \right) \subseteq C.$$

The applicability of a multiset of rules is simply an extension of the concept of rule eligibility to a multiset of rules. For a multiset R' of rules to be applicable, all rules in R' should be eligible (condition 1) and there should be enough objects in the configuration's cells to be consumed by all rules in R' (condition 2). Vector X represents total multisets of objects that will be consumed per cell by all the rules in R'.

Definition 5. [**Application of a Multiset of Rules**] If R' is multiset of rules applicable to configuration $C = (u_1, ..., u_n)$, *applying* R' to configuration C means producing a new configuration C' which is defined as:

$$C' = Apply(R', C) = C - \left(\sum_{(X_j \to Y_j; K_j) \in R'} X_j \right) + \left(\sum_{(X_j \to Y_j; K_j) \in R'} Y_j \right).$$

The *application function* $Apply(R', C)$, as defined above, takes an applicable multiset of rules R' and configuration C and outputs a new configuration C'. Configuration C' is the result of removing all multisets of objects consumed by all rules in R' from configuration C and then adding all multisets of objects produced by all rules in R' to configuration C.

Definition 6. [**Network of Cells**] A $\mathscr{F}$-controlled *network of cells* of degree n is the construct

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

where

- n is the number of cells;
- V is a finite alphabet;
- $W = (w_1, ..., w_n)$ is the *initial configuration* where $w_i \in V^\circ$ is the multiset associated with cell i.
- $c_{in} \subseteq [1..n]$ is the set of *input cells.*
- $c_{out} \subseteq [1..n]$ is the set of *output cells.*
- R is the set of interactive rules.

The primary components of a network of cell (system) are its initial configuration W and its set of interactive rules R. The first two components of a network of cells specify the degree and the alphabet used by the system. This means that the configuration and the rules of the system are of degree n and over alphabet V. c_{in} is the set of cell *ids or labels* that specifies the set of input cells while c_{out} is the set of cell ids or labels that specifies the set of output cells.

If system Π is $\mathscr{F}$-controlled, it means all control languages used by the rules of Π belong to the family of languages $\mathscr{F}$. More specifically, the control languages used by the system Π belong to the family $\mathscr{F}^\circ(V)$ of multiset languages over V.

During computation, a network of cells changes its configuration. If a system Π has the configuration C, $Applicable(\Pi, C)$ is the set of all multiset of rules (rules from R of Π) applicable to Π's current configuration C.

Definition 7. [**Network of Cells with Environment**] An $\mathscr{F}$-controlled *network of cells with environment* of degree n is the construct:

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

- The first component Π_{inf} is as defined in Definition 6.
- $Inf = (inf_1, ..., inf_n)$ where $inf_i \subseteq V$ is a *set of symbols occurring infinitely often in cell i.*

A network of cells with environment is an extension of the network of cells concept. A network of cells with environment allows infinite multisets. $Inf = (inf_1, ..., inf_n)$ is an n-vector of sets of symbols (symbols from V). inf_i is a set of symbols associated with cell i. All symbols in inf_i occurs infinitely often in cell i. If inf_i is not empty, then cell i contains an infinite multiset.

Definition 8. [**Derivation Mode**] A *derivation mode* δ is a restriction of the set of applicable rules. For network of cells Π and configuration C, $Applicable(\Pi, C, \delta) \subseteq Applicable(\Pi, C)$ denotes the set of multisets of rules in Π applicable to configuration C according to derivation mode δ.

Maximal parallelism or *maximally parallel* derivation mode is one of the most commonly used derivation modes. A maximally parallel network of cells only uses applicable multisets of rules whose size is already maximal. i.e. One can not add additional instances of eligble rules to a maximal multiset of rules without making the (new) multiset of rules non-applicable (there are not enough objects for the new multiset of rules to consume). Another common derivation mode is *minimal parallelism* or *minimally parallel* derivation mode. In this mode, rules of the system Π are grouped into non-intersecting partitions. Only applicable multisets of rules that have at most one instance of rule for each of the partition are used in this mode. Maximal parallelism, minimal parallelism and other derivations modes are listed in detail in Appendix A.

Definition 9. [**Network of Cells Working in δ Derivation Mode**] An $\mathscr{F}$-controlled n-degree *network of cells working in δ derivation mode* is the construct:

$$\Pi' = (\Pi, \delta) = ((n, V, W, c_{in}, c_{out}, R), \delta)$$

- The first component Π is as defined in Definition 6.
- δ is the derivation mode used.

$NC(n, V, \mathscr{F}, \delta)$ is the set of n-degree $\mathscr{F}$-controlled network of cells using alphabet V and working in δ derivation mode.

Definition 10. [**Computation of a Network of Cells**] A *computation* of a network of cell $\Pi' = ((n, V, W, c_{in}, c_{out}, R), \delta)$ is sequence a $C_0, C_1, C_2, ...$ of configurations of Π with the following properties:

- $C_0 = W$
- $C_{i+1} = Apply(R', C_i)$ where $R' \in Applicable(\Pi, C_i, \delta)$.

Definition 11. [**Input Function**] An *input function* for a system $\Pi' = ((n, v, W, c_{in}, c_{out}, R), \delta)$ is a function of the form $Input(\Pi') : \mathbb{N} \to V^{\circ n}$ and fulfills that condition that for all $i \notin c_{in}$ the i-th component of resulting input vector from $V^{\circ n}$ is an empty multiset.

The input function takes a number representing a time step in the computation, say t, as input and produces an n-vector, say I, of multisets over V. i.e. $Input(\Pi')(t) = I$. Only the j-th components, where $j \in c_{in}$, are allowed to be non-empty multisets since c_{in} specifies the input cells that are allowed to take non-empty input multisets.

Definition 12. [**Computation with Input of a Network of Cells**] A *computation with input* of a network of cell $\Pi' = ((n, V, W, c_{in}, c_{out}, R), \delta)$ is a sequence a $C_0, C_1, C_2, ...$ of configurations of Π with the following properties:

- $C_0 = W + Input(\Pi')(0)$
- $C_{i+1} = Apply(R', C_i) + Input(\Pi')(i + 1)$ where $R' \in Applicable(\Pi, C_i, \delta)$.

If the context is clear, a computation with input will simply be called a computation.

Definition 13. [**Output Function**] An *output function* $Output(\Pi', C)$ for a system $\Pi' = ((n, V, W, c_{in}, c_{out}, R), \delta)$ and a computation $C = C_0, C_1, C_2, ...$ of system Π' is a function that takes a number representing a time step as input and produces an output derived from the finite computation $C_0, ..., C_t$.

The output of a system Π depends on how the system is being used. For example, the system Π can be used to compute a number and this number is represented as a multiset in a particular cell in Π. For this case, the output function simply returns the multiset of the specified cell for a given time t or it can return the number itself represented by the multiset. System Π can also be used to generate a string where the i-th character of the string is represented by the multiset in a particular cell in the i-th configuration C_i of the computation. In this case, the output function for system Π and computation C returns the said string.

Definition 14. [**Halting Condition**] A *halting condition* is a condition placed on a finite computation C of a system Π' that determines if the computation is a *halting computation* and that system Π' halts after computation C.

The most commonly used halting condition is the *total halting* condtion. Total halting states that a computation $C = C_0, ..., C_t$ of a system $\Pi' = (\Pi, \delta)$ is a halting computation if $Applicable(\Pi, C_t, \delta) = \emptyset$. i.e. There are no applicable multiset of rules (in δ derivation mode) for configuration C_t. Commong halting conditions are listed in detail in Appendix B.

4 Spiking Neural P System Models as Formal Framework Network of Cells

This section looks at different SNP system models as networks of cells. Concepts and constructs from the different SNP system models are translate to concepts and construct in the formal framework.

4.1 Spiking Neural P System

Definition 15. [Spiking Neural P System] A *spiking neural P system (SNP system)* [12] of degree n is the construct:

$$\Pi = (O, \sigma_1, ..., \sigma_n, syn, i_o)$$

- $O = \{a\}$ is the singleton alphabet of the system. Symbol a is called a *spike*.
- $\sigma_1, ..., \sigma_n$ are neurons of the form
  - $\sigma_i = (n_i, R_i)$ for $1 \leq i \leq n$.
  - $n_i \in \mathbb{N}$ is initial number of spikes in neuron i.
  - R_i is a finite set of rules of the following two forms:
    - *Spiking Rule:* $E/a^c \rightarrow a; d$ where E is regular expression over O, $d \in \mathbb{N}$,$c \in \mathbb{N}\backslash\{0\}$.
    - *Forgetting Rule:* $a^s \rightarrow \lambda$ where $s \in \mathbb{N}\backslash\{0\}$ and $\{a^s\} \cap L^\circ(E) = \emptyset$ for all E that are regular expressions of a spiking rules in the same neuron.
- $syn \subseteq [1..n] \times [1..n]$ is the set of *synapses* where $(i, i) \notin syn$ for all $i \in [1..n]$.
- $i_o \in [1..n]$ specifies the *input neuron* in accepting mode or the *output neuron* in generating mode.

Standard SNP systems of degree n, as networks of cells, belong to the family $NC(n, O, REG^\circ(O), min)$. They use the alphabet is $O = \{a\}$. All control languages used by the rules are from the family $REG^\circ(O)$ which is the family of regular multiset languages over O. The systems work under minimal parallelism ($\delta = min$) where rules in the system are partitioned in such a way that each neuron has its own set of rules and only combinations of rules where at most 1 rule is used per neuron are allowed.

A spiking rule $E/a^c \rightarrow a; d$ in neuron i is eligible if the multiset $m_i = a^{n_i}$ in the neuron i is in the multiset languages $L^\circ(E)$ and the number of spikes n_i is at least c. If the rule is applied, at step t, c spikes are consumed changing the spike count of the neuron to $n_i - c$, the neuron 'closes' for d time steps from time step t to time step $t + d - 1$, then the neuron will send a spike at time $t + d$ to all other neurons adjacent to neuron i i.e. Neuron i will send a spike to all neuron j such that $(i, j) \in syn$. The d compnent of the rule is called the *delay* since it delays the rule from sending out spikes. When a neuron is closed, it can neither receive spikes nor activate a rule. A forgetting rule $a^s \rightarrow \lambda$ in neuron i is eligible if the multiset m_i in the neuron is a^s. When the forgetting rule is applied, all s spikes in the neuron are consumed changing the spike count in the neuron to 0.

An n-degree SNP system's configuration is the construct $(m_1/t_1, ..., m_n/t_n)$ where m_i is the multiset of spikes in neuron i while $t_i \in \mathbb{N}$ is the *state* of neuron i. Neuron i is in an *open state* if $t_i = 0$ otherwise it is in a *closed state* if $t_i > 0$. t_i also represents the time delay before the active rule in neuron i sends a spike out.

To simplify the formal framework, the delay component of the spiking rule is not incorporated to the framework's interaction rule. This is not very restrictive, in terms of computability, since it was shown in [9] that the SNP system model that only has spiking rules without delay (i.e. all spiking rules have $d = 0$, simply written as $E/a^c \rightarrow a$) is still turing-complete. This also means that the configuration of the system is simply the n-vector $(m_1, ..., m_n)$ of the multisets of spikes of the neurons.

An **SNP system's spiking rule (without delay) $\mathbf{E/a^c \rightarrow a}$** in neuron i can be written as the formal framework interaction rule:

$$[L^\circ(E)]_i / [a^c]_i \rightarrow [a]_{j_1} \cdots [a]_{j_k}$$

where neurons $j_1, ..., j_k$ are neurons adjacent to neuron i. i.e. $(i, j) \in syn$ for $j \in \{j_1, ..., j_k\}$, as shown in Figure 1.

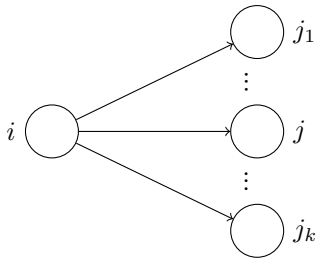

Figure 1: Neuron i and Adjacent Neurons $j_1, ...j_k$

An **SNP system's forgetting rule $\mathbf{a^s \to \lambda}$** in neuron i can be written as the interaction rule:

$$[L^\circ(E)]_i \ / \ [a^c]_i \to \emptyset_n$$

Originally, SNP systems have been used to accept or generate sets of numbers. An SNP system in *accepting mode* is used to accept a set of numbers. Neuron i_o is the specified input neuron. The input neuron will receive two spike, one spike at time t_1 and another spike at time t_2. The difference $n = t_2 - t_1$ between the arrival time of the first spike and second spike is the input to the system. Number n is accepted by the system if the system halts on the input. Total halting is the halting condition used by the SNP system model.

The function below is the input function used by the system Π' (network of cells representation of SNP system Π):

$$Input(\Pi')(t) = Z^{(t)} = (z_1^{(t)}, ..., z_{i_o}^{(t)}, ..., z_n^{(t)}) = (\emptyset, ..., z_i^{(t)}, ..., \emptyset)$$

$$z_{i_o}^{(t)} = \begin{cases} a & , \ t \in \{t_1, t_2\} \\ \emptyset & , \ t \notin \{t_1, t_2\} \end{cases}$$

An SNP system in *generating mode* is used to generate a set of numbers. Neuron i_o is the specified output neuron. The number generated by the SNP system is represented by the difference $n = t_2 - t_1$ between time t_1 when neuron i_o first spikes and time t_2 when neuron i_o spikes for the second time. The function below is the output function used by network of cells Π':

$$Output(\Pi', C)(t) = \begin{cases} n = t_2 - t_1 & \text{, if output neuron spikes at least once in } t \text{ steps} \\ \infty & \text{, if output neuron has not spike yet in } t \text{ steps} \end{cases}$$

$C = C_1, C_2, C_3, ...$ is the computation of Π'. Both t_1 and t_2 can be determined by looking at the different configurations in computation C then the output function can simply return $n = t_2 - t_1$.

The n-degree SNP system Π, as defined in Definition 15 (except all spiking rules have delay $d = 0$), can be represented by the formal framework's REG°-controlled network of cells:

$\Pi' = ((n, O, W = (w_1, ..., w_n), c_{in} = \emptyset, c_{out} = \{i_o\}, R), \delta = min)$ where $w_i = a^{n_i}$ (in generating mode)

$\Pi' = ((n, O, W = (w_1, ..., w_n), c_{in} = \{i_o\}, c_{out} = \emptyset, R), \delta = min)$ where $w_i = a^{n_i}$ (in accepting mode)

4.2 Spiking Rules as Interaction Rules

This section looks at how different spiking rules from different SNP system variants can be written as interaction rules. Different SNP system variants that are very similar to each other when one looks at them as network of cells. As networks of cells, they all have the same components (i.e. same alphabet, same configuration, same input/output functions, same derivation mode, etc). The only difference is the form of the interaction rules that they use. The different versions of spiking rules used by the different SNP system variants in this section are simply restricted versions of an interaction rule, interaction rules with specific forms. This is not always the case since there are spiking rule variants that can not be written as one interaction rule. The original spiking rule with delay is a simple example. We will only deal with non-delayed version of the different spiking rule variants in this section.

An SNP system variant in [5] has a spiking rule of the form $E/a^c \to a^p$. This rule called an *extended spiking rule*. The extended spiking rule is a generalization of the original spiking rule. In an extended

spiking rule, one can specify the number of spikes to be sent out to adjacent neurons. This is specified by the multiset a^p (p spikes are sent out). If the spiking rule is in neuron i and there are synapses from neuron i to neurons $j_1, ..., j_k$ (see Figure 1), then its corresponding interaction rule is:

$$[L^\circ(E)]_i \ / \ [a^c]_i \rightarrow [a^p]_{j_1} \cdots [a^p]_{j_k}$$

A variant called *SNP systems with weighted synapses* [19] has the additional feature having positive integer weights on its synapses. A spike sent through a synapse with weight w will arrive as w spikes at the target neuron. The spiking rule in this variant has the same form as an extended spiking rule, $E/a^c \rightarrow a^p$. If the rule is activated, the neuron will send out p spikes through its outgoing synapses. The weight of the synapse will be a multiplier and the p spikes sent through a synapse with weight w will arrive at the target neuron as $p \cdot w$ spikes. If a spiking rule $E/a^c \rightarrow a^p$ is in neuron i and neuron i is connected to neurons $j_1, ..., j_k$ where the weight of synapse (i, j_q) is w_q $(1 \leq q \leq k)$, see Figure 3, then its corresponding interaction rule is:

$$[L^\circ(E)]_i \ / \ [a^c]_i \rightarrow [a^{p \cdot w_1}]_{j_1} \cdots [a^{p \cdot w_q}]_{j_q} \cdots [a^{p \cdot w_k}]_{j_k}$$

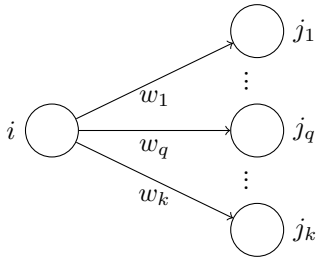

Figure 2: Neuron i and Adjacent Neurons $j_1, ...j_k$

A variant called *SNP systems with multiple channels* [22] adds the idea of *synapse labels* to the original SNP system. This SNP system variants has a set of labels and each synapse in the system is labeled. The spiking rule in this variant has the form $E/a^c \rightarrow a : (l)$ where the l is a synapse label. If the spiking rule is in neuron i and neuron i is connected to neurons $j_1, ..., j_k$, when the rule $E/a^c \rightarrow a^p(l)$ is activated it will consume c spikes in neuron i and will send p spikes through synapses labeled l. No spikes will be transmitted through out-going synapses with label that is not labeled l. Let $j'_1, ..., j'_q$ be a subset of $j_1, ..., j_k$ such that neurons $j'_1, ..., j'_q$ are connected to neuron i via synapses labeled l (i.e. synapses $(i, j'_1), ..., (i, j'_q)$ are all labeled as l). Then the spiking rule's corresponding interaction rule is:

$$[L^\circ(E)]_i \ / \ [a^c]_i \rightarrow [a^p]_{j'_1} ... [a^p]_{j'_q}$$

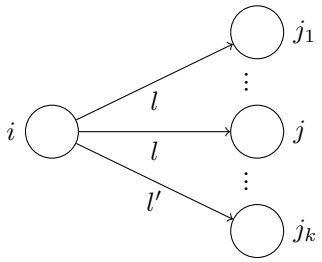

Figure 3: Neuron i and Adjacent Neurons $j_1, ...j_k$

4.3 Spiking Rules with Delays

An n-degree SNP system whose spiking rules have no delay ($d = 0$) can be represented by a network of cell from the family $NC(n, \{a\}, REG^\circ, min)$. This section shows that n-degree SNP systems that use spiking rules with delay ($d > 0$) can simulated by network of cells from the family $NC(n, V, REG^\circ, min)$

where $V = \{a\} \cup \{x, r_1, ..., r_f\}$. With the help of the additional symbols from $\{x, r_1, ..., r_f\}$, the behavior of a spiking rule with delay can be simulated by a set of interaction rules.

Given a spiking rule $E/a^c \to a^p; d$, the following behavior should be simulated:

- *Delayed Spiking*: When the rule is applied at time t, the neuron should only send spikes to adjacent neurons at time $t + d$.
- *Prevention of Rule Application*: When a rule is active in a neuron, no other rules in the neuron should be applied.
- *Closing of Neuron*: When a neuron is closed, it should not receive spikes from other neurons.

If a spiking rule with delay d is applied at time t, it will be active for $d+1$ steps, from time t to time $t+d$. The spiking rule is active $d+1$ steps but the neuron that applied the rule is only closed for d steps, from time t to time $t+d-1$. At time $t+d$, the neuron opens and it sends out spikes to adjacent neurons. Time $t + d$ is the only time when the rule is still active but the neuron is already open. The scenario at time $t + d$ is the reason why the *closing of neuron* behavior is differentiated from the *prevention of rule application* behavior.

Additional symbols will be used in order to simulate the behavior of spiking rules. The *delay* symbol x is used to represent delay and the closed state of a neuron. If there are x symbols in a neuron, then that neuron is closed. The *rule* symbols $r_1, ..., r_f$ are used to represent rules being active in a neuron. If symbol $r_i \in \{r_1, ..., r_f\}$ is in a neuron, then the spiking rule represented by r_i is active in that neuron. If neuron i contains the most number of spiking rules (with delay), then f is the number of spiking rules (with delay) in neuron i. For example, if neuron A contains the most number of spiking rules, having 10 spiking rules with delay, then $f = 10$ and set of rule symbols that will be used is $\{r_1, ..., r_{10}\}$. Neuron A will use symbols $r_1, ..., r_{10}$ but another neuron in the same system, say neuron B, that only has two spiking rules with delay, will only use symbols r_1 and r_2. The alphabet of the network of cell will be $V = \{a, x, r_1, ..., r_f\}$.

Let the spiking rule R be in neuron i and neurons $j_1, ..., j_k$ are adjacent to neuron i (see Figure 1). The behavior of a spiking rule $R : E/a^c \to a^p; d$ will be simulated by interaction rules with the following forms:

- $R_1 : [L^\circ(E_1)]_i/[a^c]_i \to [x^{d-1}r]_i$
- $R_2 : [L^\circ(E_2)]_i/[x]_i \to \emptyset_n$
- $R_3 : [L^\circ(E_3)]_i/[r]_i \to [a^p]_{j_1} \cdots [a^p]_{j_k}$
- $E_1 = E$
- $E_2 = a^*|x^+|r_1^*| \cdots |r_f^*$
- $E_3 = a^*|r$

Interaction rule R_1 can only be applied if the neuron i contains the multiset a^n such that $a^n \in L^\circ(E_1)$ where $E_1 = E$. This is almost the exact scenario when spiking rule R can be applied. i.e. $a^n \in L(E)$. The difference is that the regular expression E_1 of interaction rule R_1 is over the alphabet $V = \{a, x, r_1, ..., r_f\}$ and has a specific form specified in Section 2 while the regular expression E of spiking rule R is over the alphabet $O = \{a\}$. The regular expression E_1 specifies which spike counts can make interaction rule R_1 eligible but it also specifies that there should not be any delay symbol x nor rule symbols $r_1, ..., r_f$ for interaction rule to be eligible. For example, if an interaction rule uses the regular expression $(a^2)^+$ and control language $L^\circ((a^2)^+) = \{a^2, a^4, a^6, ...\}$, then the rule is only eligible if there is an even number of spikes and there are no $x, r_1, ..., r_p$ symbols in cell/neuron i. It is not sufficient to check only the number of spikes. The lack of delay and rule symbols means that the neuron/cell is open and there are no active rules. When interaction rule R_1 is applied at time t, it will consume the multiset a^c in neuron i and it will produce multiset $x^{d-1}r$ in the same neuron at time t. Symbol $r \in \{r_1, ..., r_f\}$ is the rule symbol associated with spiking rule R so it is one of the symbols produced by interaction rule R_1. The presence of delay symbols x means neuron i is now closed while the presence of rule symbol r means there is an active rule in neuron i.

Interaction rule R_2 simulates the delay countdown. Rule R_2 has a single eligibility condition, the presence of at least one delay symbol x in neuron i. It does not care about the spike count nor the presence or absence of rule symbols $r_1, ..., r_f$. This single condition is specified by the rule's regular expression $E_2 = a^*|x^+|r_1^*| \cdots |r_f^*$. When applied, interaction rule R_2 consumes a single delay symbol x from neuron i but does not produce any new symbols.

If interaction rule R_1 is applied at time t and neuron i has the multiset a^n, after rule application neuron i will now have the multiset $a^{n-c}x^{d-1}r$. From time $t + 1$ to $t + d - 1$, there will be at least one delay symbol x in neuron i which means interaction rule R_2 will be used in that time period. At time

$t+d$, the multiset in neuron i is $a^{n-c}r$. There are no more delay symbols x. There are only spike symbols and the single rule symbol r. Interaction rule R_3 that is associated with spiking rule R is applied at time $t+d$. Regular expression $E_3 = a^*|r$ specifies that rule R_3 is eligible if the rule symbol r is in neuron i and there are no other rule symbols nor delay symbols in the neuron. It does not care about the spike count. When rule R_3 is applied at time t, rule symbol r is consumed from neuron i and p spikes are produced in each neuron $j \in \{j_1, ..., j_k\}$ adjacent to neuron i.

Figure 4 shows the rules used by neuron i in time t to $t+d$ and the multiset in the neuron before and after the rule is applied.

Time	**Multiset (Before Rule Application)**	**Rule**	**Multiset (After Rule Application)**
$t+0$	a^n	R_1	$a^{n-c}x^{d-1}r$
$t+1$	$a^{n-c}x^{d-1}r$	R_2	$a^{n-c}x^{d-1}r$
...	...	R_2	...
$t+d-2$	$a^{n-c}x^2r$	R_2	$a^{n-c}x^1r$
$t+d-1$	$a^{n-c}xr$	R_2	$a^{n-c}r$
$t+d$	$a^{n-c}r$	R_3	a^{n-c}

Figure 4: Multiset in Neuron i Before and After an Interaction Rule is Applied (From Time t to $t+d$)

For each spiking rule R, there are corresponding interaction rules R_1 and R_3. For example, if there are two spiking rules with delay in a neuron, rules R_u and R_v, then spiking rule R_u will have the corresponding interaction rule $R_{u,1}$ that has the form of interaction rule R_1 and interaction rule $R_{u,3}$ that has the form of interaction rule R_3 while spiking rule R_v will have interaction rules $R_{v,1}$ and $R_{v,3}$. The neuron will only have a single interaction rule R_2 since this rule is not specific to a single spiking rule. To simulate the behavior of the spiking rule R_u, interaction rules $R_{u,1}$, R_2, and $R_{u,3}$ are used and to simulate the behavior of spiking rule R_v, interaction rules $R_{v,1}$, R_2, and $R_{v,3}$ are used. The delay of the spiking rule is encoded in interaction rule R_1 while the action (number of spikes produces) is encoded in interaction rule R_3. Interaction rule R_2 is used to implement the delay countdown mechanism used by all spiking rules with delay.

The *prevention of rule application* behavior is implemented using interaction rule R_1 (is not eligible if there are rule symbols in the neuron) while the *spiking delay* behavior is implemented using all three interaction rules R_1 (produces delay objects), R_2 (countdown mechanism), R_3 (actual spike production). Interaction rule R_1 produces delay symbols in the neuron that indicate that the neuron is closed but there are no mechanisms preventing other neurons from sending spikes. For example, if there is a synapse from neuron h to neuron i, R_3 interaction rules in neuron h should be modified such that they will not produce spikes in neuron i if neuron i is closed. Let $R_3 : [L^\circ(E_3)]_h/[r]_h \rightarrow \cdots [a^p]_i \cdots$ be an interaction rule in neuron h. The rule produces the multiset a^p in neuron i as specified by $[a^p]_i$ on the right of the arrow. Interaction rule R_3 has to be modified in such a way as to not send the multiset a^p to neuron i if neuron i is closed or is about to close. Neuron i is closed if there is at least one delay symbol x in the neuron and neuron i is about to close if there are no delay symbols in the neuron but there is at least one rule with delay in neuron i that is eligible. Interaction rule R_3 will have to check the multiset in neuron i to see if there are any delay symbols in the neuron or to see if the multiset in neuron i makes rules with delay in neuron i eligible. For simplicity, we will call a neuron *closed* if it is actually closed (neuron with delay symbols) or if it is about to close (with eligible rules with delay). Interaction rule R_3 will be modified to include a control language that is associated with neuron i. i.e. Interaction rule R_3 will have the form $[L^\circ(E_3)]_h[L^\circ(E_i)]_i/[r]_h \rightarrow \cdots [a^p]_i \cdots$. The control language $L^\circ(E_i)$ contains all multisets that signify that neuron i is open. The regular expression E_i specifies multisets that do not contain delay symbol x and do not contains spike counts that will make a rule with delay in neuron i to be eligible.

We look at an SNP subsystem in Figure 5 for a specific example. For now, our concern is translating the spiking rule $r_u : a^7/a^7 \rightarrow a^3 : 6$ in neuron h to a set of interaction rules. At most, there are two rules with delay in a neuron so the alphabet for the corresponding network of cells is $V = \{a, x, r_1, r_2\}$. We will use the rule symbol r_1 for spiking rule r_u in neuron h.

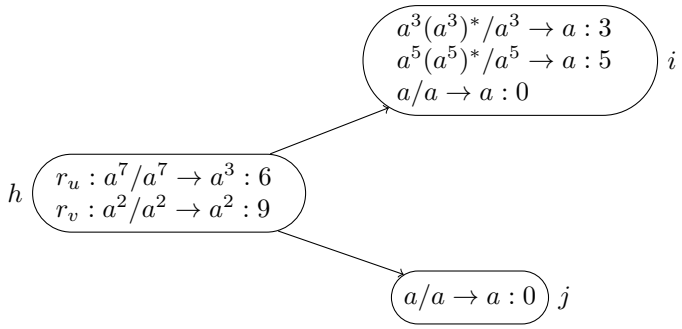

Figure 5: Example SNP System

Initially, spiking rule $r_u : a^7/a^7 \to a^3 : 6$ will have the corresponding interaction rules:

- $R_{u,1}:\ [L^\circ(a^7)]_h/[a^7]_h \to [x^5 r_1]_h$
- $R_{u,2}:\ [L^\circ(a^*|x^+|r_1^*|r_2^*)]_h/[x]_h \to \emptyset_n$
- $R_{u,3}:\ [L^\circ(a^*|r_1)]_h/[r_1]_h \to [a^3]_i[a^3]_j$

Interaction rule $R_{u,1}$ only activates if there is exactly 7 spikes in neuron h and there are no delay or rule symbols. When applied, say at time t, interaction rule $R_{u,1}$, consumes 7 spikes and produces the multiset $x^5 r_1$ in neuron h. Multiset x^5 represents 5 additional time steps (time $t+1$ to $t+5$) when neuron h is closed (excluding the current time step t). The rule symbol r_1 represents spiking rule r_u being active. Interaction rule $R_{u,2}$ will be applied as long as there are still delay symbols in neuron h. Interaction rule $R_{u,2}$ only consumes a single delay symbol x per step and it will be used for 5 steps (at time $t+1$ to $t+5$). In total, neuron h is closed for 6 steps at time t to $t+5$. At time $t+6$, there will be no more delay symbols but the rule symbol r_1 is still in neuron h which means interaction rule $R_{u,3}$ will be applied consuming rule symbol r_1 in neuron h and producing the multiset a^3 in both neuron i and neuron j.

Interaction rule $R_{u,3}$always produces spikes in neuron j. This should not be the case for neuron i.Neuron i contains spiking rules with delay which means interaction rule $R_{u,3}$ should check if neuron i is closed or open and only produce spikes in neuron i if neuron i is open. Two versions of interaction rule $R_{u,3}$ should be in neuron h, a version that checks if neuron i is open and sends spikes to both neuron i and neuron j (we denote this rule as $R_{u,3}$) and a version that checks if neuron i is closed and sends spikes only to neuron j (we denote this rule as $\overline{R}_{u,3}$). Interaction rules $R_{u,3}$ and $\overline{R}_{u,3}$ are the following:

- $R_{u,3}:\ [L^\circ(a^*|r_1)]_h[L^\circ(E_i)]_i/[r_1]_h \to [a^3]_i[a^3]_j$
- $\overline{R}_{u,3}:\ [L^\circ(a^*|r_1)]_h[L^\circ(E_i')]_i/[r_1]_h \to [a^3]_j$
- where $E_i' = a^*|x^+|r_1^*|r_2^* \cup a^3(a^3)^* \cup a^5(a^5)^*$

Interaction rule $\overline{R}_{u,3}$ uses the control language $L^\circ(E_i')$. The regular expression E_i' specifies the set of multisets that signify that neuron i is closed. Regular expression E_i' is composed of three sub-expressions connected by the union operator $\cup$. The sub-expression $a^*|x^+|r_1^*|r_2^*$ represents the multisets that contain at least one delay symbol x which means neuron i is closed. The sub-expressions $a^3(a^3)^*$ and $a^5(a^5)^*$ are the regular expressions of spiking rules with delay in neuron i. If neuron i contains a multiset that is specified by either $a^3(a^3)^*$ or $a^5(a^5)^*$, then neuron i is about to close. By combining the three sub-expressions using the union operator, the resulting expression E_i' specifies all multisets that make neuron i a closed neuron. When activated, interaction rule $\overline{R}_{u,3}$ only produces the multiset a^3 in neuron j.

Interaction rule $R_{u,3}$ uses the complementary control language $L^\circ(E_i)$. If regular expression E_i' specifies multisets where neuron i is closed, regular expression E_i specifies multisets where neuron i is open. When activated, interaction rule $R_{u,3}$ produces the multiset a^3 in both neuron i and neuron j.

In general, if neuron h is connected to q neurons with spiking rules with delay, then for a spiking rule r_u in neuron h the corresponding set of interaction rules contains the rules $R_{u,1}$, $R_{u,2}$ and 2^q versions of $R_{u,3}$. For example, if neuron h is connected to two neurons ($q = 2$) both with spiking rules with delay, say neuron a and neuron b, then the spiking rule r_u will have $2^q = 2^2 = 4$ versions of interaction rule $R_{u,3}$, one for each of the following cases:

- $R_{u,3}$ for the case when both neuron a and neuron b are closed.
- $R_{u,3}$ for the case when neuron a is closed and neuron b is open.
- $R_{u,3}$ for the case when neuron a is open and neuron b is closed.
- $R_{u,3}$ for the case when both neuron a and neuron b are open.

In summary, in order to simulate the behavior (delay spiking, prevention of rule application, closing of neuron) a spiking rule r_u, additional symbols and three types of interaction rules should be created, interaction rules R_1, R_2, R_3. For spiking rule r_u, the R_1 interaction rule is denoted by $R_{u,1}$, the R_2 interaction rule is denoted by $R_{u,2}$, and the R_3 interaction rule is denoted by $R_{u,3}$. Interaction rule $R_{u,1}$ contains the information about spiking rule r_u's regular expression, delay and the number of spikes it consumes. Interaction rule $R_{u,3}$ contains the information about spiking rule r_u's action (the number of spikes it produces and the target neurons). Interaction rule $R_{u,2}$ is used for the delay countdown mechanism and does not contain information particular to spiking rule r_u which means it can be reused by other spiking rules with delay in the same neuron. i.e. If a neuron has two spiking rules with delay, r_u and r_v, their corresponding R_2 interaction rules $R_{u,2}$ and $R_{v,2}$ is the same interaction rule. If the target neurons in $R_{u,3}$ can close, then multiple versions of $R_{u,3}$ should be created for the different scenarios where some of the targets neurons are closed while some target neurons are open.

4.4 Implementable Features of SNP System Models

In Section 4.3, we showed there is a feature (spiking rule delay) that is not an inherent part of the formal framework but can be 'implemented' using constructs (additional symbols, set of interaction rules) in the framework. This section will look at different SNP system models with features that are not part of the framework but can be implemented in the framework. This section will give some ideas on how those different features can be implemented in the framework.

An SNP system variant called *extended SNP system* [1] was introduced in 2006 as an extension of the original SNP system. Its spiking rule has the form $(i, E/a^k \rightarrow P; d)$ where i specifies the neuron that contains the rule, E is a regular expression over $\{a\}$ that defines *checking set*, a^k is the multiset consumed by the rule, $d \geq 0$ is the spiking delay, and P is the set of *productions* of the from (j, w, t) where j is the target neuron, $w \in a^*$ is the multiset produced, and $t \geq 0$ is the *transmission time* (the time it will take for the multiset w to arrived at neuron j). If both spiking delay and spike transmission time are zero, then a spiking rule for extended SNP systems can be written as a single formal framework interaction rule. As shown in the previous section, the spiking delay mechanism can be implemented in the formal framework. The transmission times can be implemented by creating chains of neurons whose function is simply to delay the spike transmission. For example, the rule $(i, E/a^k \rightarrow \{(j, a^p, 7)\} : d)$ has a single production that sends multiset a^p from neuron i to neuron j with transmission time of 7 steps. A chain of 7 neurons will be created. This chain will eventually lead to neuron j (the target neuron in the production). After rule activation and the delay, the corresponding interaction rule will send the multiset a^p to the first neuron in the chain. The only rule in the neurons of the chain is an interaction rule that will forward the multiset a^p to the next neuron. Multiset a^p should arrive at neuron j after 7 steps (the transmission time).

SNP system with structural plasticity [3] is an SNP system variant that has new types of rule called *plasticity rules*. Plasticity rules are used to create or destroy synapses. A plasticity rule has the form $E/a^c \rightarrow \alpha k(N)$ where E is a regular expression over $\{a\}$ that defines the checking set, a^c is the multiset consumed by the rule, $\alpha \in \{-, +, \pm, \mp\}$ is the action (destruction, creation, creation-then-destruction, destruction-then-creation) of the rule, k is the number of synapses to be created or destroyed, and N is a set of target neurons. For example, if the rule $E/a^c \rightarrow +3(\{1, 7, 9, 10\})$ is in neuron 2 and is activated, then the rule will try to create 3 synapses that start from neuron 2 and terminate at 3 neurons from the set $\{1, 7, 9, 10\}$. This variant also has a spiking rule with the same form as the original SNP system's spiking rule but without delay. The activation of a plasticity rule at time t_0 will change the effect of the activation of a spiking rule in the same neuron at time $t > t_0$. For example, if a plasticity rule that creates synapses is activated at time t, spiking rules in the same neuron after time t will now have more target neurons. The effect of the plasticity rule changes the effect of the spiking rules in the same neuron. In the formal framework, spiking and plasticity rules can be written as interaction rules. The corresponding interaction rule of a plasticity rule will use additional symbols (aside from the spike symbol). The additional symbols will be consumed or produced in the same neuron that contains the rule. The existence (or absence) of these symbols in a neuron represents the existence (or absence) of synapses from the neuron that contains

the rule to other neurons. The symbols represent the local topology. For example, let a system have 6 neurons, say neurons $1, 2, 3, 4, 5, 6$. In neuron 1, the existence of symbols s_2, s_3, s_6 means neuron 1 is connected to neurons $2, 3, 6$. The interaction rules for the plasticity rules can consume and produce such symbols which in essence changes the topology of the system while the interaction rules for the spiking rules should check for presence/absence of such symbols in order to check the current system topology and produce the output spikes in the correct set of target (adjacent) neurons.

The variant called *SNP system with astrocytes* [18] uses a construct known as an *astrocyte*. In the system, an astrocyte 'observes' a set of synapses. Specifically, it observes the spikes being transmitted through those synapses. There is a threshold value (number of spikes) associated with an astrocyte. If an astrocyte has a threshold value of x spikes, it will only allow spikes to be transmitted through the synapses it is observing if the sum of the number of spikes 'flowing' through those synapses is not above the threshold of x spikes. If the number of spikes being transmitted is above the astrocyte threshold, then all spikes being transmitted are blocked by the astrocyte (essentially erased). The astrocyte mechanism can be implemented by modifying the interaction rule version of the system's spiking rules. The astrocyte threshold mechanism will be built as part of the interaction rules. An interaction rule's scope is the entire system. i.e. An interaction rule can have a control language (checking set) for any neuron in the system. To implement the astrocyte threshold mechanism, an interaction rule (that corresponds to a spiking rule) should observe not only the neuron that 'contains' the rule but also other neurons that is associated with the same astrocyte. For example, let there be four neurons, neurons w, x, y, z, and let there be a synapse (w, x) that connects neuron w to neuron x and a synapse (y, z) that connects neuron y to neuron z. Let there be an astrocyte that observes synapses (w, x) and (y, z). Interaction rules in neuron w will 'observe' (have a control language for) neuron y while interaction rules in neuron y will also observe neuron w. This is done so neurons w and x can know if the combination of rules that they will use will lead to the sum of the spikes going through synapses (w, x) and (y, z) exceeding the threshold or not. Neurons w and y will activate interaction rules that do not send spikes to neurons x, z if the spike count produced will be above the threshold otherwise neurons w, y will activate interaction rules that send spikes to neurons x, z. Let neuron w have spiking rules $r_1 : E_1/a^{c_1} \rightarrow a^5$ and $r_2 : E_2/a^{c_2} \rightarrow a^3$, neuron y have spiking rules $r_3 : E_3/a^{c_3} \rightarrow a^7$ and $r_4 : E_4/a^{c_4} \rightarrow a$, and the astrocyte have the threshold of 7 spikes. If rule r_1 and r_3 are used at the same time, the rules will produce a sum of 12 spikes which exceeds the threshold of 7 spikes so the astrocyte will block the 5 spikes travelling via synapse (w, x) and the 7 spikes travelling via synapse (y, z). If rule r_1 and r_4 are used at the same time, the rules will produce the sum of 6 spikes and the astrocyte will allow the 5 spikes to travel via synapse (w, x) and one spike via synapse (y, z). Multiple interaction rule versions of spiking rule r_1 should be created. One version checks the content of neuron w to see if the rule is eligible and it also checks the content of neuron y to see if rule r_3 will be used. If rule r_3 will be used, this version of interaction rule for r_1 should send 5 spikes other adjacent neurons but not to neuron x since application of rules r_1 and r_3 exceeds the threshold. Another version of the interaction rule for r_1 is the one that checks if neuron y will used rule r_4. If rule r_4 will be used, this version of the interaction rule for r_1 will send 5 spikes to all adjacent neurons including neuron x. The same thing applies for rule r_2, versions of interaction rules for r_2 should also check which rules in neuron y is eligible. On the other hand, interaction rules for rules r_3 and r_4 in neuron y should check which rules in neuron w are eligible.

In the SNP system variant called *SNP systems with rules on synapse* [26], spiking rules are placed on synapses instead of being inside neurons. Each synapse has its own set of rules. In this system, the neuron's function is primarily for storing spikes. A spiking rule placed on synapse (u, v) consumes spikes from neuron u and produces spikes in neuron v. The spiking rules in this system has a delay component. When activated, a spiking rule with delay will close a synapse instead of a neuron. The system, like other SNP system variants, work in minimally parallel mode (at most 1 rule per synapse will be used). If two rules from two different synapses that start from the same neuron are eligible, they can only be activated at the same time if both the rules consume the same number of spikes. For example, let rule $r_1 : E_1/a^{c_1} \rightarrow a$ be in synapse (x, y) and rule $r_2 : E_2/a^{c_2} \rightarrow a$ be in synapse (x, z). Both rules consume spikes from neuron x. The *strategy* used by this system is that if both r_1 and r_2 are eligible they can both be activated when they consume the same number of spikes (i.e. $c_1 = c_2$). Otherwise, only one of them can be activated. The set of eligible rules from different synapses (with the same neuron origin) that can be activated at the same time is the set that contains rules that consume the same number of spikes. The behavior of SNP systems with rules on synapse can be implemented in the formal framework. Different spiking rules on synapses (with the same origin neuron) can be combined into a set of interaction rules that will implement the behavior described above. For example, let $r_1 : a^2(a^2)^*/a^3 \rightarrow a : 1$ and $r_2 : a^3(a^3)^*/a^5 \rightarrow a : 2$ be rules on synapse (x, y) and $r_3 : a^4(a^4)^*/a^5 \rightarrow a : 3$ and $r_4 : a^5(a^5)^*/a^3 \rightarrow a : 4$

be rules on synapse (x, z). Interaction rules can be created for each of these rules. Additionally, interaction rules that combine spiking rules that can be activated together will be created. For example, rule r_1 and r_4 are both eligible when the spike count in neuron x is a multiple of 10 (r_1 is eligible when spike count is even while r_4 is eligible when the spike count is a multiple of 5) and they can be activated at the same time since they both consume 3 spikes. The interaction rules $[L(a^{10}(a^{10})^*)]_x/[a^5]_x \rightarrow [a]_y[a]_z$ corresponds to the combination of both r_1 and r_2 are activated. An interaction rule that corresponds to both r_2 and r_2 will also be added. We can use the idea from Section 4.3 to implement delay for synapses. The difference is that each synapse has its own delay. i.e. One synapse from neuron 1 is open, another synapse will be closed for 3 steps, and another synapse will be closed for 5 more steps. The state of each synapse will be encoded as existence of delay symbols associated with the said synapse. The interaction rules will also be designed to check for these synapse delay symbols. For example, the interaction rule that correspond to the combination or r_1 and r_2 will be modified to check for synapse delay symbol in order to see the state (close/open) of synapses (x, y) and (x, z).

4.5 On Other Features of SNP System Models

This section looks at different SNP system models with features that are 'impossible' to implement or at least very difficult or impractical to implement in the formal framework.

The SNP system variant known as *SNP system with white hole neurons* [23] has a spiking rule version called *white hole spiking rule* that has the form $E/a^{all} \rightarrow a^p : d$. It has very similar behavior to the extended spiking rule except that a white hole spiking rule consumes all the spikes in the neuron instead of a specified fixed number of spikes. Initially, it may appear that the difference between an extended spiking rule and a white hole spiking rule is minimal but when you take a closer look it can be argued that the white hole spiking rule is significantly more powerful (in terms of computability) than the extended spiking rule. An extended spiking rule $E/a^c \rightarrow a^p : d$ has a fixed 'action' or effect. When activated, it consumes c spikes, produce p spikes, and closes the neuron for d steps. A white hole spiking rule $E/a^{all} \rightarrow a^p : d$ has a range (infinite set) of 'actions' or effects. Its action is dependent on the number of spikes in the neuron. For example, the white hole spiking rule $a^2(a^2)^*/a^{all} \rightarrow a$ has an infinite range of possible actions since this rule can consume any even number of spikes greater that 0. The only way to implement the behavior of rule $a^2(a^2)^*/a^{all} \rightarrow a$ using only extended spiking rules is to have an infinite number of rules (one for each even number of spikes). Though the variable action of a white hole spiking rule is only concerned about the spike consumption, one can easily extended it and create a spiking rule whose action (the number of spikes produces and the delay) varies with respect to the spike count in the neuron.

The SNP system variant known as *SNP system with weights* [31] deals with real numbers, real number synapse weights and threshold functions. This variant is closer to spiking neural networks than multiset-based SNP systems (or P systems in general). Just the task of representing real numbers using only discrete formal framework constructs is already a difficult and impractical task.

An SNP system variant known as *SNP systems with anti-spikes* [15] has two types of objects, the spike a and the *anti-spike* $\overline{a}$. Inside each neuron, there is a implicit rule of the form $a\overline{a} \rightarrow \lambda$. This rule means that if there is a spike a and an anti-spike $\overline{a}$ in the neuron, they will 'annihilate' each other and both of them will be erased. This 'annihilation' rule is applied in a maximal manner. For example, if a neuron started with no objects and it receives 5 spikes and 7 anti-spikes at the current step, there will be 5 spike-anti-spike pairs that will annihilate leaving 2 anti-spikes in the neuron for the next step. This means that at the beginning of each time step, inside a neuron, there is only multiset of spikes or a multiset of anti-spikes and not a multiset with both spikes and anti-spikes. This SNP varaints has a spiking rule of the form $E/b^c \rightarrow b'$ where $b, b' \in \{a, \overline{a}\}$ and E is a regular expression that is either over alphabet $\{a\}$ or over alphabet $\{\overline{a}\}$. The spiking rule is eligible if (case 1) the regular expression E is over $\{a\}$, the neuron contains the multiset a^n, $a^n \in L(E)$, and the rule consumes $b^c = a^c \subseteq a^n$ or if (case 2) the regular expression E is over $\{\overline{a}\}$, the neuron contains the multiset $\overline{a}^n$, $\overline{a}^n \in L(E)$, and the rule consumes $b^c = \overline{a}^c \subseteq \overline{a}^n$. The rule produces the either spike ($b' = a$) or an anti-spike ($b' = \overline{a}$). The variant also has a forgetting rule of the form $b^c \rightarrow \lambda$ where $b \in \{a, \overline{a}\}$, a rule the simple consumes spikes or anti-spikes. For SNP systems with anti-spikes, any rule (spiking, forgetting, 'annihilation') can simply be written as a formal framework interaction rule. The features that significantly differentiate this variant is its *two-phase* rule application and its unusual derivation mode that combines minimal and maximal parallelism. At start of a computational step, the first phase involves each neuron checking if it has eligible rules and neurons with eligible rules should activate one eligible. The first phase works in minimally parallel (min_1) derivation mode. At most one rule is applied per neuron and a neuron is

required to apply a rule if there is an eligible rule. At the same computational step, in the second phase after the spikes have already been produced, the annihilation rule is applied in a maximal way in all the neurons. Essentially, the two-phase rule application will take two computational steps in the formal framework. Additionally, the two-phase rule application not only 'compresses' two computational steps in one computational step, it also uses two different derivation modes one for each phase. It might be possible to implement a 'hybrid' derivation mode that combines minimal and maximal parallelism (both derivation modes used during computation but only one mode is used at a given time). On the other hand, effect of the two-phase rule application might be impossible or at least very difficult to implement.

5 Discussion

In Section 4, we used the formal framework to look at different SNP system models and their features. We roughly categorize the features into three categories: (1) features that can be directly represented in the framework, (2) features that can not be directly represented but can be implemented in the framework, and (3) features that impossible or very difficult to even implement in the framework. For brevity, we can call the features in the first category as *representable* features, the features in the second category as *implementable* features, and the features in the third category as *non-implementable* features.

In Section 4.2, we look at some framework-representable features of some SNP system variants. Specifically, we look at three different versions of spiking rules from three different SNP system variants, the extended spiking rule, the spiking rule for SNP systems with (positive integer) weights, and the spiking rule for SNP systems with labeled synapses (called SNP systems with multiple channels). All non-delay versions of these spiking rules can be written as interaction rules. One spiking rule from any of those three SNP variants has a single corresponding interaction rule. This is what we mean by *representable*. A single concept/construct (feature) of an SNP system variant is represented by a single concept/construct in the framework. The spiking rules in those systems are special cases of interaction rules. i.e. interaction rules that have specific forms. The only difference between a spiking rule (from those SNP systems) and its corresponding interaction rule is the where the information about the effect of the rule is encoded. A spiking rule's effect is dependent on its location (the neuron that contains the rule) and the topology of the network. In order to know the exact effect of the rule, you have to know the rule itself, it location and information about local topology. On the other hand, the corresponding interaction rule already encodes all those necessary information. You can specify in an interaction rule which neurons to 'observe', which neuron(s) to consume multisets from, and which neurons to send multiset to. An interaction rule induces a local topology. i.e. If an interaction rule 'observes' neuron x and produces multisets in neurons a, b, and c, then it is implied that neuron x is connected to neurons a, b, and c. The interaction rules can also encode additional information like synapse weights and synapse labels.

In Section 4.4, we look at some framework-implementable features of some SNP system variants. A feature is *implementable* (and not simply representable) if you have to construct a set or combination of different formal framework constructs and concepts in order to simulate the behavior of the feature. In Section 4.3, we showed how a spiking rule with delay can be implemented in the formal framework. In Section 4.4, we sketch some ideas on how to implement the following features: (1) spiking rules in extended SNP systems, (2) plasticity rules in SNP systems with structural plasticity, (3) the astrocyte mechanism in SNP systems with astrocytes, and (4) group spiking strategy and synapse closing mechanism for SNP systems with rules on synapse. The features were implemented in the formal framework by using additional symbols, creating additional interaction rules and/or combining different spiking rules into one interaction rule, and creating additional neurons/cells. Some SNP systems' configuration is defined by the multiset contents of each neuron and possibly some internal states of the neurons. For example, a configuration in the original SNP system is defined by the multisets of spikes in each neuron and a delay status (counter) that determines if a neuron is closed or open. We have shown in Section 4.3 that this delay status can be implemented using an additional symbol called the delay symbol. In this implementation, a new symbol is used to represent some internal neuron/cell state. Additional symbols can also encode local network topology. This is shown in the idea for the implementation of plasticity rules. Plasticity rules can create and destroy synapses. The presence or absence of a synapse from neuron x to neuron y is represented by the presence or absence of a new symbol that correspond for the synapse (x, y).

It is particularly helpful that an interaction rule's scope is global. i.e. An interaction rule can look at any neuron/cell and can consume and produce multisets in any cell/neuron. In the implementation of the spiking rule with delay, there is a type of interaction rule (R_3 rules in Section 4.3) that produces spikes to a set of target neurons. This type of rule needs to check the neuron it is associated with and it needs

to check the target neurons to see if they are open or closed. Since interaction rules are global, checking the multisets of any neuron is possible making the R_3 types of interaction rule feasible. In general, the global scope of the interaction rule allows us to create a single interaction rule that combines behaviors of different spiking rules that are in different partitions i.e. behaviors of spiking rules from different neurons or different synapses. You can see an example of this in the implementation of spiking rules on synapses.

If new symbols and new interaction rules are not enough. We can also create additional cells/neurons in order to implement a feature. This is done when implementing the spike transmission delay of spiking rules in the extended SNP systems. A chain of neurons are used to simulate the spike transmission delay.

In Section 4.5, we look at features that are impossible/impractical or seemingly very difficult to implement in the formal framework. Specifically, we look at the variable action (with an infinite set of possible actions) of white hole spiking rules, the two-phase rule application in SNP systems with anti-spikes, and real number weights of SNP systems with (real) weights on synapses. We can only confidently argue that the real weights feature is impossible to implement. For the two-phase rule application and rules with variable action, we can only sketch ideas on how to implement some aspects of these features.

Formal framework constructs are made up of multisets and structures involving multisets. They are discrete and finite constructs. It is really not possible to represent real numbers, e.g. real number synapse weights and real number neuron potentials/charges, without involving infinity. i.e. using infinite amount of objects/symbols to encode a real number. With this information, it seems that the framework is not applicable for the SNP systems with real weights on synapses or in general any system that deals with real numbers.

White hole spiking rule has a variable action. Its action is dependent on the content of a neuron. On the other hand, most spiking rules has a single fixed and finite set of actions. It consumes a fixed number of spikes and it produces a fixed number of spikes. A white whole spiking rule produces a fixed number of spikes but it can consume any amount of spikes in the neuron as long as the rule is activated. We can simulate the behavior of a white hole spiking rule by using an additional rule and by using a maximally parallel derivation mode. For example, let the white hole spiking rule be the rule $E/a^{all} \rightarrow a^p$. When activated the rule will consume all spikes and produce p spikes. Create an interaction rule, say R_1, that is eligible in the same scenario as the white hole spiking rule and produces p spikes but only consumes one spike. Create another interaction rule, R_2, that is eligible in the same scenario as the white hole spiking rule and let this new interaction rule consume one spike but does not produce any spikes. When a maximally parallel derivation mode is used, in the scenario that the white hole spiking rule is eligible, the two interaction rules will be eligible. Maximal parallelism and interaction rule R_2 will make sure that all spikes are consumed. We can make some modifications in order to implement a policy that R_1 will always be used. i.e. The maximally parallel multiset of rules will always include exactly one instance of R_1 and possible multiple instance of R_2. In the maximal parallel mode, interaction rules R_1 and R_2 simulates the behavior of a white hole spiking neuron.

We can use the idea of an *action function* that takes the multiset in neuron as input and returns a set of actions (multiset to consume and multiset to produce). If we look at the action function of many spiking rule variants, most of them will have action functions that are piece-wise constant function. i.e. Let f be the action function for an original spiking rule without delay and let x be the multiset in the neuron. The set of actions of the spiking rule when the neuron contains multiset x is $f(x)$. The action function is piece-wise constant since $f(x)$ will be always be empty if the rule in not eligible when the neuron contains x or $f(x)$ is always the fixed actions $\{-a^c, a\}$ (consume c spikes, produce a spike) when the rule is eligible. The white hole spiking rule $E/a^{all} \rightarrow a^p$ will have the action function $f(x) = \{-a^x, a^p\}$ which is 'linear' with respect to the multiset x.

The two-phase rule application in SNP system with anti-spikes seems to be quite difficult to implement if not impossible. The first phase runs a minimally parallel derivation mode where spiking rules are used, 1 rule at most per neuron. The second phase runs annihilation rules in a maximally parallel manner. The system is essentially doing two computational steps at a time.

6 Concluding Remarks

Appendices

A Derivation Modes

Common derivation modes from [8].

- $Applicable(\Pi, C, \delta) \subseteq Applicable(\Pi, C)$ - Set of applicable multisets of rules in δ-mode.
- $Applicable(\Pi, C, asyn) = Applicable(\Pi, C)$ - *Asynchronous* Mode ($\delta = asyn$).
- $Applicable(\Pi, C, sequ) = \{R' \in Applicable(\Pi, C) \mid |R'| = 1\}$ - *Sequential* Mode ($\delta = sequ$) - One rule instance only.
- $Applicable(\Pi, C, max) = \{R' \in Applicable(\Pi, C) \mid \nexists R'' \in Applicable(\Pi, C),\ R'i \not\subseteq R''\}$ - *Maximally Parallel* Mode ($\delta = max$) - Adding any rule to a maximally parallel R' will result in an inapplicable multiset of rules.
- $Applicable(\Pi, C, min) = \{R' \in Applicable(\Pi, C) \mid \nexists R'' \in Applicable(\Pi, C),\ R' \subseteq R'', \exists j, (R'' - R') \cap R_j \neq \emptyset, R' \cap R_j = \emptyset\}$ - *Minimally Parallel* Mode ($\delta = min$) - There is no partition $R = R_1 \cup R_2 \cup \cdots \cup R_h$. Rule set R is be partitioned. $R' \subseteq R''$. R'' 'extends' R'.
- $\delta \in \{asyn, sequ, max, min\}$ - Basic derivation modes
- $Applicable(\Pi, C, max_{rule}\delta) = \{R' \in Applicable(\Pi, C, \delta) \mid \nexists R'' \in Applicable(\Pi, C, \delta)\ |R''| > |R'|\}$ - *Maximum Rules* δ-Mode.
- $Applicable(\Pi, C, max_{set}\delta) = \{R' \in Applicable(\Pi, C, \delta) \mid \nexists R'' \in Applicable(\Pi, C, \delta)\ ||R''|| > ||R'||\}$
  - *Maximum Sets (Partitions)* δ-Mode.
- $Applicable(\Pi, C, all_{set}\delta) = \{R' \in Applicable(\Pi, C, \delta) \mid \forall j, 1 \leq j \leq h, (R_j \cap \bigcup_{X \in Applicable(\Pi, C)} X \neq \emptyset) \rightarrow (R_j \cap R' \neq \emptyset)\}$ - *All Set* δ-Mode.

B Halting Conditions

1. Total Halting: There are no more applicable multisets of rules in the entire system.
2. Adult Halting: There are still applicable multisets of rules but all of the applicable multisets lead back to the current configuration when applied to the system.
3. Partial Halting: Rules are divided into disjoint partitions. If there are no more applicable multisets of rules that contain at least on rule per partition, then the system halts.

References

[1] Artiom Alhazov, Rudolf Freund, Marion Oswald, and Marija Slavkovik. Extended spiking neural p systems. In *Membrane Computing*, pages 123–134. Springer Berlin Heidelberg, 2006.

[2] Francis George C. Cabarle, Henry N. Adorna, Min Jiang, and Xiangxiang Zeng. Spiking neural p systems with scheduled synapses. *IEEE Transactions on NanoBioscience*, 16(8):792–801, 2017.

[3] Francis George C. Cabarle, Henry N. Adorna, Mario J. Pérez-Jiménez, and Tao Song. Spiking neural p systems with structural plasticity. *Neural Computing and Applications*, 26(8):1905–1917, November 2015.

[4] Matteo Cavaliere, Oscar H. Ibarra, Gheorghe Păun, Omer Egecioglu, Mihai Ionescu, and Sara Woodworth. Asynchronous spiking neural p systems. *Theoretical Computer Science*, 410(24):2352 – 2364, 2009. Formal Languages and Applications: A Collection of Papers in Honor of Sheng Yu.

[5] Haiming Chen, Mihai Ionescu, Tseren-Onolt Ishdorj, Andrei Păun, Gheorghe Păun, and Mario J. Pérez-Jiménez. Spiking neural p systems with extended rules: Universality and languages. *Natural Computing*, 7(2):147–166, June 2008.

[6] Haiming Chen, Tseren-Onolt Ishdorj, and Gheorghe Păun. Computing along the axon. *Progress in Natural Science*, 17(4):417–423, 2007.

[7] Rudolf Freund, Ignacio Pérez-Hurtado, Agustín Riscos-Núñez, and Sergey Verlan. A formalization of membrane systems with dynamically evolving structures. *International Journal of Computer Mathematics*, 90(4):801–815, 2013.

[8] Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) p systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, pages 271–284, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[9] Oscar H. Ibarra, Andrei Păun, Gheorghe Păun, Alfonso Rodríguez-Patón, Petr Sosík, and Sara Woodworth. Normal forms for spiking neural p systems. *Theoretical Computer Science*, 372(2):196 – 217, 2007. Membrane Computing.

[10] Oscar H. Ibarra, Andrei Păun, and Alfonso Rodríguez-Patón. Sequential snp systems based on min/max spike number. *Theoretical Computer Science*, 410(30):2982 – 2991, 2009. A bird's eye view of theory.

[11] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. Spiking neural p systems with an exhaustive use of rules. *International Journal of Unconventional Computing*, 3:135–153, 2007.

[12] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. Spiking neural p systems. *Fundamenta Informaticae*, 71(2,3):279–308, February 2006.

[13] Yun Jiang, Yansen Su, and Fen Luo. An improved universal spiking neural p system with generalized use of rules. *Journal of Membrane Computing*, 1(4):270–278, December 2019.

[14] Venkata Padmavati Metta, Srinivasan Raghuraman, and Kamala Krithivasan. Spiking neural p systems with cooperating rules. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing*, pages 314–329, Cham, 2014. Springer International Publishing.

[15] Linqiang Pan and Gheorge Păun. Spiking neural p systems with anti-spikes. *International Journal of Computers, Communications, and Control*, 4(3):273–282, 2009. cited By 81.

[16] Linqiang Pan, Gheorghe Păun, and Mario J. Pérez-Jiménez. Spiking neural p systems with neuron division and budding. *Science China Information Sciences*, 54(8):1596, July 2011.

[17] Linqiang Pan, Gheorghe Păun, Gexiang Zhang, and Ferrante Neri. Spiking neural p systems with communication on request. *International Journal of Neural Systems*, 27(08):1750042, November 2017.

[18] Linqiang Pan, Jun Wang, and Hendrik Jan Hoogeboom. Spiking neural p systems with astrocytes. *Neural Computation*, 24(3):805–825, March 2012.

[19] Linqiang Pan, Xiangxiang Zeng, Xingyi Zhang, and Yun Jiang. Spiking neural p systems with weighted synapses. *Neural Processing Letters*, 35(1):13–27, February 2012.

[20] Gheorghe Păun. Spiking neural p systems with astrocyte-like control. *Journal of Universal Computer Science*, 13(11):1707–1721, November 2007.

[21] Hong Peng, Bo Li, Jun Wang, Xiaoxiao Song, Tao Wang, Luis Valencia-Cabrera, Ignacio Pérez-Hurtado, Agustín Riscos-Núñez, and Mario J. Pérez-Jiménez. Spiking neural p systems with inhibitory rules. *Knowledge-Based Systems*, 188:105064, 2020.

[22] Hong Peng, Jinyu Yang, Jun Wang, Tao Wang, Zhang Sun, Xiaoxiao Song, Xiaohui Luo, and Xiangnian Huang. Spiking neural p systems with multiple channels. *Neural Networks*, 95:66 – 71, 2017.

[23] Tao Song, Faming Gong, Xiyu Liu, Yuzhen Zhao, and Xingyi Zhang. Spiking neural p systems with white hole neurons. *IEEE Transactions on NanoBioscience*, 15(7):666–673, 2016.

[24] Tao Song and Linqiang Pan. Spiking neural p systems with request rules. *Neurocomputing*, 193(C):193–200, June 2016.

[25] Tao Song, Linqiang Pan, and Gheorghe Păun. Asynchronous spiking neural p systems with local synchronization. *Information Sciences*, 219:197 – 207, 2013.

[26] Tao Song, Linqiang Pan, and Gheorghe Păun. Spiking neural p systems with rules on synapses. *Theoretical Computer Science*, 529:82 – 95, 2014.

[27] Tao Song, Alfonso Rodríguez-Patón, Pan Zheng, and Xiangxiang Zeng. Spiking neural p systems with colored spikes. *IEEE Transactions on Cognitive and Developmental Systems*, 10(4):1106–1115, 2018.

[28] Sergey Verlan. Using the formal framework for p systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, pages 56–79, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[29] Sergey Verlan, Rudolf Freund, Artiom Alhazov, Sergiu Ivanov, and Linqiang Pan. A formal framework for spiking neural p systems. *Journal of Membrane Computing*, 2(4):355–368, December 2020.

[30] Jun Wang, Hendrik Jan Hoogeboom, and Linqiang Pan. Spiking neural p systems with neuron division. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto S alomaa, editors, *Membrane Computing*, pages 361–376, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[31] Jun Wang, Hendrik Jan Hoogeboom, Linqiang Pan, Gheorghe Păun, and Mario J. Pérez-Jiménez. Spiking neural p systems with weights. *Neural Computation*, 22(10):2615–2646, October 2010.

[32] Tingfang Wu, Andrei Păun, Zhiqiang Zhang, and Linqiang Pan. Spiking neural p systems with polarizations. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3349–3360, 2018.

[33] Tingfang Wu, Zhiqiang Zhang, Gheorghe Păun, and Linqiang Pan. Cell-like spiking neural p systems. *Theoretical Computer Science*, 623:180 – 189, 2016.

[34] Xiangxiang Zeng, Xingyi Zhang, Tao Song, and Linqiang Pan. Spiking neural p systems with thresholds. *Neural Computation*, 26(7):1340–1361, 2014. PMID: 24708366.

[35] Xingyi Zhang, Bangju Wang, and Linqiang Pan. Spiking neural p systems with a generalized use of rules. *Neural Computation*, 26(12):2925–2943, 2014. PMID: 25149700.

[36] Yuzhen Zhao, Xiyu Liu, and Wenping Wang. Spiking neural p systems with neuron division and dissolution. *PLOS ONE*, 11(9):e0162882, September 2016.