Spiking Neural P System Models as Formal Framework Models

Ren Tristan A. de la Cruz

December 18, 2020

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. 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 symbols as computing elements. One can think of these symbols as molecules or ions. The multisets of symbols are compartmentalized into regions and these regions 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 system, 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 system, 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 a 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 [3] 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 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 [3] 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 [3] is limited to static P system. An interaction rule from [3] 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 [2] has a much more general form of an interaction rule. The interaction rule from [2] 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 [3].

A third version of the formal framework can be found in [6]. 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* (SN P systems). SN P 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. SN P 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. formal framework's interaction i 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 [5].

We want to do an exploratory study of the different SN P system variants using the formal frameworks.

SN P systems and similar variants are neural-like P systems. The rules of these models are inspired by the mechanism of a *spiking neuron.* In most SN P system variants, there is only a one type of symbol called a *spike.* The configuration of an SN P system is a network of neurons and each neuron contains a multiset of spikes. The main type of rule used by SN P system variants is called a *spiking rule.* A spiking rule looks at number of spikes in a certain neuron and if the number of spikes fits the pattern (usually described by a regular expression) specified in the rule then the neuron can send a spike to adjacent neurons in the network. The interaction rule in third version of the formal framework [**?**] was specifically extended to have control languages in order to be able to represent the spiking rule as interaction rules.

There are many SN P system variants that have features that are not available in the original SN P system model. For example, there is a variant with two types of objects known as the *spike* and the *anti-spike.* The variant is called SN P system *with anti-spikes.* The spiking rule of this variant needs to take into account the new type of object which means both its syntax and semantics are different from the original spiking rule. It also works in a slightly different derivation mode. There is a variant called SN P system *with astrocytes.* It has a new construct called *astrocyte.* An astrocyte can be connected to different channels and it monitors the spikes travelling in those channels. It is a threshold mechanism that allows travelling spikes to propagate if the total number of spikes travelling in the channels being monitored by the astrocyte passes a certain threshold. There is a variant called SN P system *with polarity.* Similar to some cell-like P systems, in SN P systems with polarity, neurons have charges that can either be positive, negative or neutral. The spiking rules in this model not only check the number of spikes in the neuron but also the polarity of the neuron. There are also SN P system variants that are exactly like the original SN P system except for the fact that they use a different derivation modes. For example, *sequential* SN P systems are SN P systems that only allows a single rule to be applied at a time. Another example is the *asynchronous* SN P system. This variant allows any combination of eligible rules (the combination does not have to be minimally parallel).

In [**?**], after defining and describing the constructs of the framework, the authors translated the SN P system model to a model of the framework. They wrote both the spiking rule and forgetting rule of the SN P system as interaction rules. They mentioned that the SN P system works in minimally parallel mode which has already been characterized as the derivation mode min_1 in [**?**]. Only original SN P system model (without delay) and SN P systems with extended rules (also without delay) have been fully translated as models of the framework. Some features of other SN P variants were translated by Verlan et al. in [**?**] and in some conference presentations (Conference on Membrane Computing) but not the full models.

Our intention for the study is to look at different SN P system variants and fully translate them to models of the framework. We want to fully translate at least two SN P variants. After translating that variants, we can them compare the framework models with each other. This can give us insights on how similar or how different those SN P system variants are to each other.

2 Preliminaries

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 $m : V \rightarrow \mathbb{N}_\infty$ while a *finite multiset* over V is a function of the form $m : V \rightarrow \mathbb{N}$. 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 $X \subseteq Y$, $Y - X = (y_1 - x_1, ..., y_n - x_n)$. Aside from denoting the empty multiset, $\emptyset$ will also denote a vector of empty multisets. i.e. $\emptyset = (\emptyset, ..., \emptyset)$. If the context is not clear, we will specify if $\emptyset$ means an empty multiset or a vector of empty multisets.

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*$^\circ$ is the set of regular multiset languages, and *REG*$^\circ(V)$ is the set of regular multiset languages over V.

3 Formal Framework for Spiking Neural P Systems

[3] [2] [5] [6]

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 and the entire rule can be written as:

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

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 $(X_j \to Y_j; K_j) \in R'$ is eligible with respect to configuration C and (2)

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

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, $\mathit{Applicable}(\Pi, C, \delta) \subseteq \mathit{Applicable}(\Pi, C)$ 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:

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

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 $\mathit{Input}(\Pi') : \mathbb{N} \to V^{\circ n}$ 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. $\mathit{Input}(\Pi')(t) = I$. 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:

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

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

Definition 13. **[Output Function]** An *output function* $\mathit{Output}(\Pi', C)$ 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 $\mathit{Applicable}(\Pi, C_t, \delta) = \emptyset$. 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

4.1 Spiking Neural P System

Definition 15. [Spiking Neural P System] A *spiking neural P system (SNP system)* [4] 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:* $E/a^c \to a; d$ where E is regular expression over O, $d \in \mathbb{N}$,$c \in \mathbb{N}\backslash\{0\}$.
    * *Forgetting Rule:* $a^s \to \lambda$ 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 *output neuron.*

SNP System' Spiking Rule:

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

SNP System's Forgetting Rule

$$[a^c]_i \to \emptyset; [L^\circ(E)]_i$$

SNP Systems with Extended Spiking Rule [1]

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

SNP Systems with Weights

$$[a^c]_i \to [a^{w_{j_1}}]_{j_1} \cdots [a^{w_{j_k}}]_{j_k}; [L^\circ(E)]_i$$

SNP Systems with Extended Rules and Weights

$$[a^c]_i \to [a^{p \cdot w_{j_1}}]_{j_1} \cdots [a^{p \cdot w_{j_k}}]_{j_k}; [L^\circ(E)]_i$$

Appendices

A Derivation Modes

[3]

- $\mathit{Applicable}(\Pi, C, \delta) \subseteq \mathit{Applicable}(\Pi, C)$ - Set of applicable multisets of rules in δ-mode.
- $\mathit{Applicable}(\Pi, C, asyn) = \mathit{Applicable}(\Pi, C)$ - *Asynchronous* Mode ($\delta = asyn$).

- $\mathit{Applicable}(\Pi, C, sequ) = \{R' \in \mathit{Applicable}(\Pi, C) \mid |R'| = 1\}$ - *Sequential* Mode ($\delta = sequ$) - One rule instance only.

- $\mathit{Applicable}(\Pi, C, max) = \{R' \in \mathit{Applicable}(\Pi, C) \mid \not\exists R'' \in \mathit{Applicable}(\Pi, C),\ R'i \not\subseteq R''\}$ - *Maximally Parallel* Mode ($\delta = max$) - Adding any rule to a maximally parallel R' will result in an inapplicable multiset of rules.

- $\mathit{Applicable}(\Pi, C, min) = \{R' \in \mathit{Applicable}(\Pi, C) \mid \not\exists R'' \in \mathit{Applicable}(\Pi, C),\ R' \subseteq R'', \exists j, (R'' - R') \cap R_j \neq \emptyset, R' \cap R_j = \emptyset\}$ - *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

- $\mathit{Applicable}(\Pi, C, max_{rule}\delta) = \{R' \in \mathit{Applicable}(\Pi, C, \delta) \mid \not\exists R'' \in \mathit{Applicable}(\Pi, C, \delta)\ |R''| > |R'|\}$
  - *Maximum Rules* δ-Mode.

- $\mathit{Applicable}(\Pi, C, max_{set}\delta) = \{R' \in \mathit{Applicable}(\Pi, C, \delta) \mid \not\exists R'' \in \mathit{Applicable}(\Pi, C, \delta)\ ||R''|| > ||R'||\}$
  - *Maximum Sets (Partitions)* δ-Mode.

- $\mathit{Applicable}(\Pi, C, all_{set}\delta) = \{R' \in \mathit{Applicable}(\Pi, C, \delta) \mid \forall j, 1 \leq j \leq h, (R_j \cap \bigcup_{X \in \mathit{Applicable}(\Pi, C)} X \neq \emptyset) \rightarrow (R_j \cap R' \neq \emptyset)\}$ - *All Set* δ-Mode.

B Halting Conditions

1. Total Halting
2. Adult Halting
3. Partial Halting

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.