Homogenization of Spiking Neural P Systems

Ren Tristan A. de la Cruz[a,*], Francis George C. Cabarle[a], Author Three[b]

[a] *Algorithms and Complexity Lab, Department of Computer Science, University of the Philippines Diliman, 1101, Quezon City, Philippines.*
[b] *(address 2)*

Abstract

ABSTRACT

Keywords: WORD, WORD, WORD, WORD

1. Background

MEMBRANE COMPUTING
SNP
[2] [1] [7] [5] [3]
[4] [6]
SNP VARIANTS
SNP NORMAL FORMS + HOMOGENEOUS
SNP HOMOGEN LITERATURE
SNP HOMOGENIZATION Section 2 Section 2.1 Section 2.2 Section 3 Section 3.1 Section 3.2 Section 3.3 Section 4

2. Preliminaries

2.1. Languages and Regular Languages

An alphabet V is a finite set of symbols, a string s is a concatenation of symbols from some V, so that the string is said to be *over the alphabet* V. If s is a string over V, we denote as $|s|$ the length of s while $|s|_a$ where $a \in V$ denotes the number of occurrences of symbol a in s.

A language L is a set of strings. When talking about languages, the term *word* can be used as a synonym for string. For some alphabet V we have V^* as the language that contains strings of all lengths over V including the empty string, denoted as λ. Further, $V^+ = V^* - \{\lambda\}$.

In defining a specific type of language known as *regular languages*, we can use *regular expressions*. We define regular expressions in an iterative manner over

*Corresponding Author
Email addresses: radelacruz@up.edu.ph (Ren Tristan A. de la Cruz), fccabarle@up.edu.ph (Francis George C. Cabarle), author3@mail.com (Author Three)

an alphabet V, as follows: (1) each $a \in V$ and $\emptyset$ are regular expressions, (2) if E_1 and E_2 are regular expressions, then $E_1 \cup E_2$, $E_1 E_2$, and E_1^* are also regular expressions. The language defined by a regular expression E is denoted as $L(E)$. If E_1 is a regular expression $E_1 = a$ where $a \in V$, then the language $L(E_1)$ is $\{a\}$. The language defined by regular expression $\emptyset$ is the empty language $\{\}$. Given regular expressions E_1 and E_2 and the languages they define, $L(E_1)$ and $L(E_2)$ respectively, the language defined by regular expression $E_1 \cup E_2$ is $L(E_1 \cup E_2) = L(E_1) \cup L(E_2)$, the language defined by regular expression $E_1 E_2$ is $L(E_1 E_2) = \{xy | x \in L(E_1) \text{ and } y \in L(E_2)\}$, and the language defined by regular expression E_1^* is $L(E_1^*) = \{xy | x \in L(E_1) \cup \{\lambda\} \text{ and } y \in L(E_1^*)\}$.

2.2. Spiking Neural P Systems

[1]

Definition 1 (Spiking Neural P Systems)**.** A *spiking neural P system* of degree $n \geq 1$ is a construct of the form

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

where

- $O = \{a\}$ is the singleton *alphabet* where the element a is called a *spike*.
- $\sigma_1, ..., \sigma_m$ are *neurons* having the form $\sigma_i = (n_i, R_i)$ for $1 \leq i \leq n$ where:
  - $n_i \geq 0$ is the *initial number of spikes* in σ_i.
  - R_i is the finite *rule set* of σ_i. A rule in R_i has the form $E/a^c \rightarrow a^p{:}d$ where E is a regular expression over O, $c \geq 1$, $p \geq 0$, $c \geq p$, and $d \geq 0$. c is the number of spikes *consumed* by the rule, p is the number of spikes *produced* by the rule, and d is the spiking *delay*.
- $syn \subseteq \{1, 2, ..., m\} \times \{1, 2, ..., m\}$ with $(i, i) \notin syn$ for any $i \in \{1, 2, ..., m\}$ is the *set of synapses* between neurons.
- $in, out \in \{1, 2, ..., m\}$ indicate the *input* and *output* neurons.

Let $E/a^c \rightarrow a^p{:}d$ be a rule in neuron σ_i (the rule is in R_i). The rule works in the following manner. If neuron σ_i has k spikes, $a^k \in L(E)$, and $k \geq c$ then the rule is *applicable*. If $E/a^c \rightarrow a^p{:}d$ is applicable and it is applied by neuron σ_i at time t, then c spikes are removed from neuron σ_i at time t (changing neuron σ_i's spike count to $k - c$) then p spikes are sent to all neurons σ_j where that $(i, j) \in syn$ at time $t + d$ (each neuron σ_j receives p spikes). At time t to $t + d - 1$ neuron σ_i is said to be *closed* which means the neuron can not receive spikes from other neurons. Spikes sent to a closed neuron are removed from the system. The rule is said to be *active* from time t to $t + d$ which means no other rules in neuron σ_i can be applied.

We say that rule $E_1/a^{c_1} \rightarrow a^{p_1}{:}d_1$ and rule $E_2/a^{c_2} \rightarrow a^{p_2}{:}d_2$ *intersect* if $L(E_1) \cap L(E_2) \neq \emptyset$. If those two rules are in neuron σ_i with k spikes, $a^k \in$

$L(E_1) \cap L(E_2)$, $k \geq c_1$, and $k \geq c_2$ then those two rules are applicable at the same time. If a neuron has multiple applicable rules, then it non-deterministically selects one rule to apply.

Given a rule $E/a^c \rightarrow a^p{:}d$, if $E = a^c$, we can write the rule as $a^c \rightarrow a^p{:}d$. If $d = 0$, we can write the rule as $E/a^c \rightarrow a^p$. If $E = a^c$ and $d = 0$, we can write the rule as $a^c \rightarrow a^p$. If $p = 0$, the rule is called a *forgetting rule*, if $p = 1$, the rule is called a *standard spiking rule*, and if $p > 1$, the rule is called an *extended spiking rule*.

The original SNP system [2] only uses forgetting and standard spiking rules. A forgetting rule in the original SNP system has the form $a^c \rightarrow a^0$, written as $a^c \rightarrow \lambda$, which means the regular expression of the forgetting rule is always $E = a^c$ and it has no delay ($d = 0$). Additionally, the original SNP system has the restriction that in a neuron no forgetting rules should intersect with any spiking rule. The SNP system in [1] introduces the idea of the extended spiking rule but the actual SNP system model in [1] does not use the concept of delay.

The system assumes a global clock. The system operates in the following manner: for each step, each neuron in the system that does not have an active rule will check if any of its rules is applicable and if at least one rule is applicable then the neuron will apply an applicable rule. This mode of operation is called *minimally parallel*. i.e. Neurons work in parallel but each neuron only applies only one applicable rule if there are any (a neuron has to apply one rule if there are any applicable rules). The system will continue this operation until it reaches a halting state. The system is in a *halting state* if all neurons in the system have no applicable rules and no active rules.

A configuration of the system is defined as the tuple $C = (n_1/d_1, ..., n_i/d_i, ..., n_m/d_m)$ where n_i is the number of spikes in neuron σ_i and $d_i \in \mathbb{N} \cup \{-1\}$ is a number that represents the state of neuron σ_i. State $d_i = -1$ means the neuron σ_i has no active rules. State $d_i = 0$ means neuron σ_i is open but has an active rule that is about to send spikes. State $d_i > 1$ means neuron σ_i is closed and will only open and send spikes after d_i steps. We call n_i/d_i the *configuration of neuron* σ_i and the d_i component the *delay counter* of neuron σ_i.

Given the configuration $C = (n_1/d_1, ..., n_i/d_i, ..., n_m/d_m)$ we can know what events will occur and can occur in the system. An *event* is either a neuron spiking, a neuron applying a rule, or a neuron 'counting down' before spiking. An event changes a configuration. Neuron spiking sends out spikes to other neurons changing their spike counts, a neuron applying a rule consumes spikes, while a neuron with active rule 'counting down' decrements the state d of the neuron. If the component n_i/d_i of C has $d_i = 0$, then neuron σ_i spiking is an event in C. If the component n_i/d_i has $d_i > -1$, then neuron σ_i counting down is an event in C. If the component n_i/d_i has $d_i = -1$ and neuron σ_i has some applicable rule r, then neuron σ_i applying rule r is a possible event in C. We say that a set of events S is *consistent with configuration* C if it includes all neuron spiking events and neuron countdown events in C and it includes one rule application event for each neuron with no active rules but has some applicable rules. We say that configuration C *transitions* to configuration C', written as $C \Rightarrow C'$, if S is a set of events consistent with C and C' is the result

events in S happening in Π while its configuration is C. A *computation* of Π is simply a sequence of configuration transitions starting from the system's initial configuration $(n_1/\text{-}1, n_2/\text{-}1, ..., n_m/\text{-}1)$.

SNP systems can be used as an *acceptor*, as a *generator*, or as a *transducer*. In general, SNP systems use *spike trains* as input and/or output. A spike train is simply a sequence of spikes which can also be interpreted as a sequence of spike counts. For example, the spike train $a^3a^0a^2a^1$ is the sequence that starts with 3 spikes, followed by 0 spikes, then by 2 spikes and the by 1 spike. As a sequence of spike count, the spike train $a^3a^0a^2a^1$ can be written as $3, 0, 2, 1$. As an acceptor, the systems receives a spike train input in its input neuron. The spike train is accepted if the system halts. An acceptor SNP system computes the set of spike trains it accepts. As a generator, the system produces an output spike train via its output neuron. If the system halts, the spike train produced by the output neuron is the spike train generated by the system. A generator SNP system computes that set of spike trains it generates. As a transducer, the system receives a spike train x in its input neuron. The system either it halts with spike train x as input or it does not. If the system halts, the spike train y generated by the output neuron will be the output of the system for input x. The transducer system computes some binary relation that contains the spike train pairs (x, y) where x is the input spike train and the system halts on x with output y.

3. Homogenization of Spiking Neural P Systems

3.1. Representing Neurons as Transition Systems

Definition 2 (Neuron Transition System)**.** A *neuron transition system* (NTS) is a tuple $(S, V, \rightarrow)$ where

- S is a finite *set of states*. A *state* $s \in S$ is a subset of $\mathbb{N}$. A state represents a set of spike counts.
- V is a finite *set of events*. An *event* $e \in V$ has the form (α, β) where $\alpha \in \mathbb{Z}$, $\beta \in \mathcal{A}$, and $\mathcal{A}$ is some *set of rule actions*. i.e. $\beta = \lambda$ (forgetting action), $\beta = a^p{:}d$ spiking action with delay). If $\alpha < 0$, then the event (α, β) represents the application of a rule that consumes α spikes and performs action β. i.e. $E/a^\alpha \rightarrow \beta$. If $\alpha > 0$, then the event (α, β) represents the reception of α spikes and its β can be set as a *non-action*. i.e. $\beta = \lambda$ (non-action, 'action' of the forgetting rule).
- $\rightarrow \subseteq S \times V \times S$ is the *transition relation*. A $(s, e, s') \in \rightarrow$ is called a *transition*. If $e = (\alpha, \beta)$, the transition $(s, (\alpha, \beta), s')$ has the property $s' = \{n + \alpha \mid n \in s\}$. Since the next state s' can be derived from the current state s and α, the transition $(s, (\alpha, \beta), s')$ can simply be written as $(s, (\alpha, \beta))$.

Let $nts = (S, V, \rightarrow)$ be the NTS of some neuron $\sigma_w = (n, R)$. nts is constructed using the rule set R and the possible spike trains going to neuron w.

For each rule $E/a^{\alpha} \rightarrow \beta$ in R, the transition $(N(L(E)), (-\alpha, \beta))$ is an element of the transtion relation $\rightarrow$. $N(L(E))$ is the length set of language $L(E)$. If it is possible for neuron w to receive α spikes while neuron w is *in state* s (neuron w has n spikes and $n \in s$), then the transition $(s, (\alpha, \lambda))$ is in the transition relation $\rightarrow$. If transitions $(s, (\alpha, \lambda))$ and $(s, (\alpha', \beta))$ are both in the transition relation $\rightarrow$, then the transition $(s, (\alpha + \alpha', \beta))$ is also in $\rightarrow$ if $\beta = a^p{:}d$ where $d = 0$. Transition $(s, (\alpha', \beta))$ represents a rule application. If the rule has a delay ($d > 0$), then it is not possible for the neuron to receive α spikes while the rule is also applied since the neuron closes immediately upon rule application. In transition $(s, (\alpha + \alpha', \beta))$, the event $(\alpha + \alpha', \beta)$ represents two events: the reception of α spikes and the application of a rule that consumes α' spikes and has action β.

We will call the set that contains some neuron w and all neurons *connected to* neuron w, all neuron x such that $(x, w) \in syn$, as the *neuron* w *subsystem*.

Figure 1 shows two neurons with their subsystems and their corresponding NTSs. For an NTS diagram, a state s is drawn as a rectangle that contains the element of s. For the transition $(s, (\alpha, \beta), s')$, the event (α, β) is drawn as an arrow from s to s' with "(α, β)" being the arrow's label. Figure 1a shows neuron w and its NTS. The NTS of neuron w has the transition $(\{1\}, (-1, \lambda))$ that corresponds to the application of rule $a \rightarrow \lambda$ and the transition $(\{2\}, (-2, a))$ that corresponds to the application of rule $a^2 \rightarrow a$. The transition $(\{0\}, (1, \lambda))$ corresponds to the event when neuron w has no spikes and it receives 1 spikes from either neuron x or neuron y while transition $(\{0\}, (2, \lambda))$ corresponds to the event when neuron w has no spikes and it receives 2 two spikes from neurons x and y. The NTS of neuron w in Figure 1a assumes that neuron w only receives spikes from neurons x and y when neuron w has no spikes.

Figure 1b shows another neuron w with its NTS. The transition $(\{1\}, (-1, \lambda))$ corresponds to the application of rule $a \rightarrow \lambda$ while the transition $(\{2i + 1\}_{i \geq 1}$, $(-3, a))$ corresponds to the application of rule $a(a^2)^+/a^3 \rightarrow a$. The transitions $(\{2i\}_{i \geq 0}, (1, \lambda))$ and $(\{2i\}_{i \geq 0}, (2, \lambda)$ represent the events where neuron w receives either 1 spike or 2 spikes from neuron x and/or neuron y when neuron w has even number of spikes.

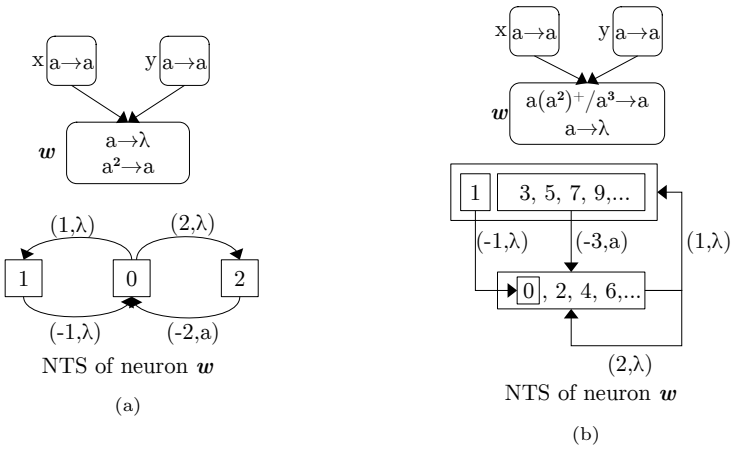


Figure 1: Examples of Neurons and their NTSs

Figure 2 highlights the effect of the behavior of the neuron subsystem to the neuron's NTS. Figure 2a's neuron w and Figure 2b's neuron w have the same rule set with the single rule $a^3 \rightarrow a$ but have different subsystems. In Figure 2a, the subsystem (neurons x, y, z) sends a total of 3 spikes to neuron w and the spikes are sent one spike at a time. In Figure 2b, the subsystem also sends a total of 3 spikes to neuron w but there are four different ways for the 3 spikes to arrive at neuron w. These four ways are: (1) 3 spikes are sent one at a time, (2) 1 spike is sent first then 2 spikes are sent later, (3) 2 spikes sent first then 1 spike is sent later, and (4) 3 spikes are sent at the same time. Different behavior of the subsystems imply different sets of transitions and hence different NTSs.

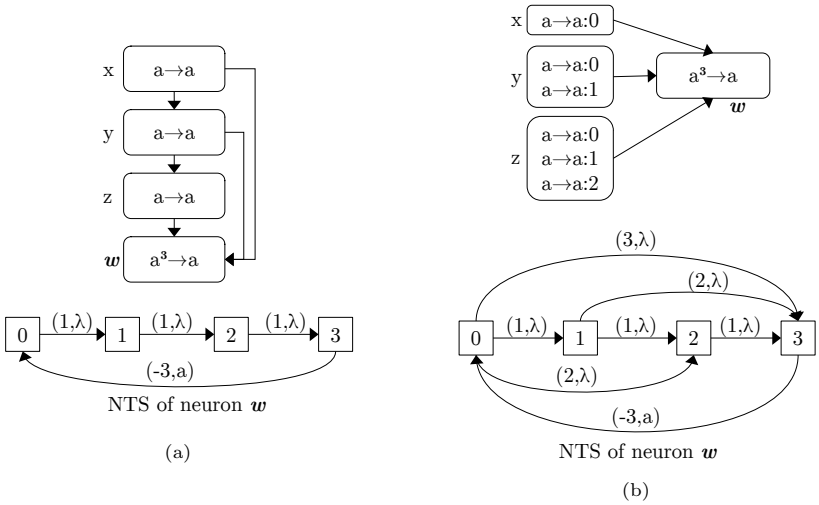


Figure 2: Neurons with same rule set but different NTSs

Definition 3 (Full Event Sequence)**.** Let neuron σ have the initial configuration $c_0 = (n_o/\text{-}1)$ and neuron transition system $nts = (S, V, \rightarrow)$. Let spike train $st = a^{\alpha_0}a^{\alpha_1} \cdots a^{\alpha_t} \cdots$ be a valid input spike train to neuron σ. In NTS terms, the input spike train st can be viewed as the *input event sequence* $se_{input} = e_0, e_1, ..., e_t, ... = (\alpha_0, \lambda), (\alpha_1, \lambda), ..., (\alpha_t, \lambda), ...$ A *full event sequence* for a given neuron σ and input event sequence se_{input} is an event sequence is:

$$se_{full} = e'_0, e'_1, ..., e'_t, ... = (\alpha'_0, \beta'_0), (\alpha'_1, \beta'_1), ..., (\alpha'_t, \beta'_t), ...$$

where, given the configuration $c_t = (n_t/d_t)$ and event $e_t = (\alpha_t, \lambda)$, $e'_t = (\alpha'_t, \beta'_t)$ and c_{t+1} are defined in the following manner:

- Case A: $d_t > 0$: $c_{t+1} = (n_t/d_t\text{-}1)$ and $e'_t = (\alpha'_t, \beta'_t) = (0, \lambda)$.
- Case B: $d_t = 0$: $c_{t+1} = (n_t + \alpha_t/d_t\text{-}1)$ and $e'_t = (\alpha'_t, \beta'_t) = (\alpha_t, \lambda)$.
- Case C_1: $d_t = \text{-}1$: $c_{t+1} = (n_t + \alpha_t + \alpha/d\text{-}1)$ and $e'_t = (\alpha'_t, \beta'_t) = (\alpha_t + \alpha, \beta)$ if $(s, (\alpha_t + \alpha, \beta)) \in \rightarrow$, $n_t \in s$, $\beta = a^p{:}d$, and $d = 0$.
- Case C_2: $d_t = \text{-}1$: $c_{t+1} = (n_t + \alpha/d\text{-}1)$ and $e'_t = (\alpha'_t, \beta'_t) = (\alpha, \beta)$ if $(s, (\alpha, \beta)) \in \rightarrow$, $n_t \in s$, $\beta = a^p{:}d$, and $d > 0$.

3.2. Operations on Neuron Transition Systems

Definition 4 (NTS Translation)**.** *NTS translation* is an operation on an entire NTS. It takes a neuron transition system $nts = (S, V, \rightarrow)$ and a natural number δ and produce the neuron transition system $nts' = (S', V, \rightarrow')$ where:

- $S' = \{s + \delta \mid s \in S\}$. $s + \delta = \{i + \delta \mid i \in s\}$. We say that $s + \delta$ is *state* s *translated by* δ or is *delta-translated state* s.
- $\rightarrow' = \{t + \delta \mid t \in \rightarrow\}$. If $t = (s, (\alpha, \beta))$, $t + \delta = (s + \delta, (\alpha, \beta))$. We say that $t + \delta$ is *transition* t *translated by* δ or is δ*-translated transition* t. δ-transition of transition t is simply the δ-translation of the component s of the transition.

We denote nts' as $nts + \delta$. We say that $nts + \delta$ is *nts translated by* δ or is δ*-translated nts*.

When you translate an NTS of some neuron w, the resulting new NTS is a neuron transition system of a different neuron, say neuron w'. For neurons, we will use an operation called *neuron translation*. Neuron translation is the analogue of NTS translation. NTS translation operates on NTSs while neuron translation operates on neurons.

Definition 5 (Neuron Translation)**.** *Neuron translation* is an operation on a neuron. It takes a neuron $\sigma = (n, R)$ and a natural number δ and produce the neuron $\sigma' = (n + \delta, R')$ where $R' = \{a^\delta E/a^c \rightarrow \beta \mid E/a^c \rightarrow \beta \in R\}$. We say that neuron σ' is σ *translated by* δ or is δ*-translated* σ.

A δ-translation of neuron w involves adding δ spikes to the neuron's initial spike count and *δ-translating* the rules of the neuron. δ-translating a rule $E/a^c \to \beta$ means changing the regular expression of the rule to $a^\delta E$ (δ-translation of regular expression E).

Figure 3 shows neuron w' which is a δ-translated version of neuron w from Figure 1a and the NTS of neuron w' which is a δ-translated version of the NTS of neuron w. Neuron w has 0 initial spikes and the rules $a \to \lambda$ and $a^2 \to a$. Neuron w', being a δ-translated neuron w, has δ initial spikes and has rules $a^{\delta+1}/a \to \lambda$ and $a^{\delta+2}/a \to a$. The rules of neuron w' are δ-translated rules of neuron w. The translated rules in neuron w' have the same actions and consume the same amount of spikes and only the regular expressions are modified (δ-translated). The NTS of neuron w', being a δ-translated NTS of neuron w, has all δ-translated transistions of the NTS of neuron w.

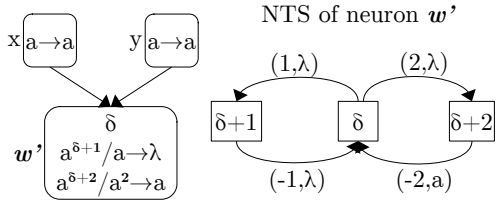


Figure 3: Neuron w' and its NTS

Theorem 1. A neuron and a translated version of the neuron have the same behavior.

Proof. Let neuron w' with LTS lts' be a δ-translated version of some neuron w with LTS lts. If neuron w has an initial n_0 spikes, then neuron w' has an initial $n_0 + \delta$ spikes. Let T_0 be the set of transition in lts such that $(s, (\alpha, \beta)) \in T_0$ if and only if $n_0 \in s$. Let T_0' be the corresponding set for lts'. i.e. $T_0' = \{(s', (\alpha', \beta')) \mid n_0 + \delta \in s'\}$. If $(s, (\alpha, \beta)) \in T_0$, then $(s + \delta, (\alpha, \beta)) \in T_0'$ since $n_0 \in s$ and $n \in s$ implies $n_0 + \delta \in s + \delta$. If $(s + \delta, (\alpha, \beta)) \in T_0'$, then $(s, (\alpha, \beta)) \in T_0$ since $n_0 + \delta \in s + \delta$ and $n_0 + \delta \in s + \delta$ implies $n_0 \in s$ (by reversing the δ-translation of state $s + \delta$). This means that the transitions in T_0' are all δ-translated transitions of T_0. Since transition translation does not change the label (α, β), the transitions in T_0 has the same set of labels as the transitions in T_0'. This means that neuron w at n_0 spike count and neuron w' at $n_0 + \delta$ have the same set of actions they can perform. The same argument can be used for when neuron w has spike count n (which is not necessarily the initial spike count) while neuron w' has spike count $n + \delta$. □

Definition 6 (NTS Scaling)**.** *NTS scaling* is an operation on an entire NTS. It takes a neuron transition system $nts = (S, V, \to)$ and a natural number δ and produce the neuron transition system $nts' = (S', V', \to')$ where:

- $S' = \{\delta s \mid s \in S\}$. $\delta s = \{\delta i \mid i \in s\}$. We say that δs is *state s scaled by δ* or is *δ-scaled state s*.

- $V' = \{(\delta\alpha, \beta) \mid (\alpha, \beta) \in V\}$.
- $\rightarrow' = \{\delta t \mid t \in \rightarrow\}$. If $t = (s, (\alpha, \beta))$, $\delta t = (\delta s, (\delta\alpha, \beta))$. We say that δt is *transition t scaled by δ* or is *δ-scaled transition t*. δ-scaling of transition t means scaling its state component s and α component by δ.

We denote nts' as $\delta \cdot nts$. We say that $\delta \cdot nts$ is *nts scaled by δ* or is *δ-scaled nts*.

When you scale an NTS of some neuron w, the resulting new NTS is a neuron transition system of a different subsystem, say neuron w' subsystem . If neuron translation is the analogue of NTS translation, *subsystem scaling* is the analogue of NTS scaling. NTS scaling operates on NTSs while subsystem scaling operates on neuron subsystems.

Definition 7 (Subsystem Scaling)**.** *Subsystem scaling* is an operation on an SNP subsystem. Let neuron $\sigma = (n, R)$ and the all neurons x connected to neuron σ be collectively known as subsystem *sub*. *Type 1 subsystem scaling* takes a subsystem *sub* and a natural number δ and produces a new subsystem *sub'* where:

- *sub'* contains neuron $\sigma' = (\delta n, R')$. $R' = \{\delta E/a^{\delta c} \rightarrow \beta \mid E/a^c \rightarrow \beta \in R\}$. δE is the regular expression that is the result of replacing all instances of subexpression a in E by the subexpression a^δ. i.e. If $E = a(a^2)^*$, then $\delta E = a^\delta((a^\delta)^2)^* = a^\delta(a^{2\delta})^*$. We call this regular expression operation as *regular expression scaling*. *Rule scaling* is the operation where you scale the rule's regular expression and scale the number of spikes the rule consumes by the same amount δ.
- If neuron x is connected to neuron σ, then δ copies of neuron x are connected to neuron σ' and all these δ copies of neuron x are also in subsystem *sub'*.

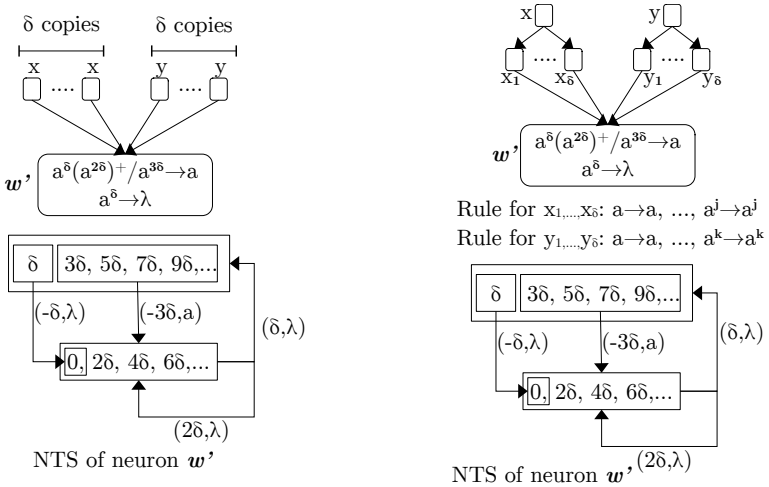
In *Type 2* subsystem scaling, *sub'* also contains the same neuron $\sigma' = (\delta n, R')$ described above. Instead of having δ copies of neuron x for each neuron x connected to neuron σ, in type 2 subsystem scaling *sub'* will include neuron x and δ *multiplier neurons* $x_1, ..., x_\delta$ for each neuron x. In the subsystem *sub'*, each neuron x is connected to its multiplier neurons $x_1, ..., x_\delta$ while all the multiplier neurons are connected to neuron σ'. Neurons $x_1, ..., x_\delta$ will have the same rule set with rules of the form $a^j \rightarrow a^j$ where j is a non-zero number of spikes a rule in neuron x can produce.

Figure 4 shows an NTS which is a δ-scaling of the NTS of neuron w from Figure 1b. The NTS of neuron w from Figure 1b has the following transitions: $(\{1\}, (-1, \lambda))$, $(\{2i+1\}_{i\geq 1}, (-3, a))$,$(\{2i\}_{i\geq 0}, (1, \lambda))$, $(\{2i\}_{i\geq 0}, (2, \lambda))$. The NTS in Figure 4 has the following transitions: $(\{\delta\}, (-\delta, \lambda))$, $(\{2i\delta + \delta\}_{i\geq 1}, (-3\delta, a))$, $(\{2i\delta\}_{i\geq 0}, (\delta, \lambda))$, $(\{2i\delta\}_{i\geq 0}, (2\delta, \lambda))$. All transitions in the NTS in Figure 4 are all δ-scaled versions of transitions in the NTS in Figure 1b.

Figure 4a shows neuron w' subsystem which is the result of type 1 subsystem δ-scaling of the neuron w subsystem in Figure 1b. The neuron w subsystem in

Figure 1b includes neuron w with rules $a(a^2)^*/a^3 \to a$ and $a \to \lambda$ and neurons x and y connected to neuron w. The neuron w' subsystem in Figure 4a includes neuron w' with rules $a^\delta(a^{2\delta})^*/a^3 \to a$ and $a^\delta \to \lambda$, δ copies of neuron x all connected to neuron w', and δ copies of neuron y all connected to neuron w'. The rules in neuron w' are all δ-scaled rules of neuron w. The initial number of spikes in neuron w' should be δ times that of the initial number of spikes of neuron w but since neuron w has 0 initial spikes neuron w' also has 0 initial spikes.

Figure 4b shows another neuron w' subsystem which is the result of type 2 subsystem δ-scaling of the neuron w subsystem in Figure 1b. The neuron w' in this subsystem is identical to the neuron w' in Figure 4a. The neuron w' subsystem in Figure 4b includes neuron x with its multiplier neurons $x_1, ..., x_\delta$ and neuron y with its multiplier neurons $y_1, ..., y_\delta$. The rules in neurons $x_1, ..., x_\delta$ have the form $a^i \to a^i$ where i is a number of spikes neuron x can produce while rules in neurons $y_1, ..., y_\delta$ have the form $a^k \to a^k$ where k is a number of spikes neuron y can produce.



(a) Type 1 Subsystem Scaling

(b) Type 2 Subsystem Scaling

Figure 4: NTS Scaling and Subsystem Scaling

3.3. Procedures for Homogenizing Neurons' Rule Sets

Definition 8 (Rule Transition Set)**.** Given a neuron transition system $nts = (S, V, \to)$, the *rule transition set* R of nts is a subset of the transition relation $\to$ that contains only those transitions that represent rules. Let $R = \{r_1, ..., r_i, ..., r_n\}$ be a rule transition set. Recall that $r_i \in R$ is a (rule) transition that has the form (s_i, e_i) where s_i is a state and e_i is an event (α_i, β_i). The *scope* of a rule transition set $R = \{(s_1, e_1), ..., (s_n, e_n)\}$, denoted by $scope(R)$, is the state $s_1 \cup s_2 \cup \cdots \cup s_n$.

Definition 9 (Partition-Events Set)**.** A *partition-events set* is another way of representing a rule transition set. Given a rule transition set $R = \{(s_1, e_1), ..., (s_n, e_n)\}$, a partition-events set P of R is some set $\{(p_1, \mathcal{E}_1), ..., (p_m, \mathcal{E}_m)\}$ where:

- The set of states $p_1, ..., p_m$ is a *partition* of $scope(R)$. i.e. $scope(R) = p_1 \cup \cdots \cup p_m$ and for any $p_i, p_j \in \{p_1, ..., p_m\}$, $p_i \cap p_j = \emptyset$. All states in the partition are non-empty.
- If $n \in scope(R)$ is some spike count, the set $events(n)$ is the set of events that can occur when the neuron has n spikes. i.e. $events(n) = \{e \mid (s, e) \in R, n \in s\}$.
- For each $(p_i, \mathcal{E}_i) \in P$, $\mathcal{E}_i = events(n)$ where $n \in p_i$ and for any other $n' \in p_i$, $events(n') = \mathcal{E}_i$. All spike counts in p_i have the same set of events $\mathcal{E}_i$.
- $scope(P) = scope(R)$.

Figure 5 is a visualization of a partition-events set of the rule transition set $R = \{(s_1, e_1), (s_2, e_2), (s_3, e_3)\}$. $scope(R) = s_1 \cup s_2 \cup s_3$ is partitioned into the following subsets/states: $J = s_1 \backslash (s_2 \cup s_3)$, $K = s_2 \backslash (s_1 \cup s_3)$, $L = s_3 \backslash (s_1 \cup s_2)$, $M = (s_1 \cap s_2) \backslash s_3$, $N = (s_1 \cap s_3) \backslash s_2$, $O = (s_2 \cap s_3) \backslash s_1$ $Q = (s_1 \cap s_2 \cap s_3)$. If any of the states is empty, then it will not be in the partition. The partition-events set P of R is the set $\{(J, \mathcal{E}_1), (K, \mathcal{E}_2), (L, \mathcal{E}_3), (M, \mathcal{E}_4), (N, \mathcal{E}_5), (O, \mathcal{E}_6), (Q, \mathcal{E}_7)\}$ where $\mathcal{E}_1 = \{e_1\}$, $\mathcal{E}_2 = \{e_2\}$, $\mathcal{E}_3 = \{e_3\}$, $\mathcal{E}_4 = \{e_1, e_2\}$, $\mathcal{E}_5 = \{e_1, e_3\}$, $\mathcal{E}_6 = \{e_2, e_3\}$, and $\mathcal{E}_7 = \{e_1, e_2, e_3\}$. P is a valid partition-events set because the states J, K, L, M, N, O, Q form a partition of $scope(R)$ and for any element $(s, \mathcal{E}) \in P$ for all $n \in s$, $events(n) = \mathcal{E}$. i.e. For $(M, \mathcal{E}_4) = ((s_1 \cup s_2) \backslash s_3, \{e_1, e_2\})$, all elements of M are from the subset of the intersection $s_1 \cap s_2$, the subset that does not contain elements from s_3. This means for any $n \in M$, $events(n) = \{e_1, e_2\} = \mathcal{E}_4$ since $n \in s_1$ and $n \in s_2$ and either of the rule transition (s_1, e_1) or rule transition (s_2, e_2) can be used. For any element $n \in M$, the set of events that can occur at spike count n is $\mathcal{E}_4$.

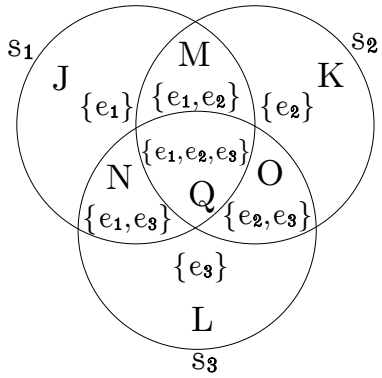


Figure 5: Partition and Events

Definition 10 (Translation and Scaling of a Partition-Events Set)**.** *Partition-events set translation* is an operation on a partition-events set. It takes a partition-events set $P = \{..., (p, \mathcal{E}), ...\}$ and a natural number δ and produce the partition-events set P' where:

- For each $(p, \mathcal{E}) \in P$, $(p + \delta, \mathcal{E}) \in P'$. We denote P' as $P + \delta$ and we say that P' is P *translated by* δ.

Partition-events set scaling is an operation on a partition-events set. It takes a partition-events set $P = \{..., (p, \mathcal{E}), ...\}$ and a natural number δ and produce the partition-events set P' where:

- For each $(p, \mathcal{E}) \in P$, $(\delta p, \mathcal{E}') \in P'$ where $\mathcal{E}' = \{(\delta\alpha, \beta) \mid (\alpha, \beta) \in \mathcal{E}\}$. We denote P' as δP and we say that P' is P *scaled by* δ.

Definition 11 (Compatibility of Two Partition-Eventss Sets)**.** Two partition-events sets, $P = \{..., (p_i, \mathcal{E}_i), ...\}$ and $P' = \{..., (p'_j, \mathcal{E}'_j), ...\}$, are *compatible* if for all $n \in scope(P) \cap scope(P')$ there is a $(p_i, \mathcal{E}_i) \in P$ and a $(p'_j, \mathcal{E}'_j) \in P'$ such that $n \in p_i$, $n \in p'_j$, and $\mathcal{E}_i = \mathcal{E}'_j$.

Algorithm 1

Algorithm 1: $PartitionEventsSet(R)$:

Input: $R = \{(s_1, e_1), ..., (s_n, e_n)\}$
Output: $P = \{(p_1, \mathcal{E}_1), ..., (p_m, \mathcal{E}_m)\}$

$P' \leftarrow \{\}$;
for *each* $T \subseteq R$ **do**
  $T' \leftarrow R \backslash T$;
  $S \leftarrow \{s \mid (s, e) \in T\}$;
  $S' \leftarrow \{s \mid (s, e) \in T'\}$;
  $\mathcal{E}' \leftarrow \{e \mid (s, e) \in T\}$;
  $p' \leftarrow \Big(\bigcap_{s \in S} s \Big) \Big\backslash \Big(\bigcup_{s' \in S'} s' \Big)$;
  if $p' \neq \emptyset$ **then**
    Add $(p', \mathcal{E}')$ to P' ;
  end
end
$P \leftarrow \{\}$;
$EV \leftarrow \{\mathcal{E}' \mid (p', \mathcal{E}') \in P'\}$;
for *each* $\mathcal{E}' \in EV$ **do**
  $p \leftarrow \{\}$;
  for *each* $(p'_i, \mathcal{E}'_i) \in P'$ **do**
    if $\mathcal{E}'_i = \mathcal{E}'$ **then**
      $p \leftarrow p \cup p'_i$;
    end
  end
  Add $(p, \mathcal{E}')$ in P ;
end

Algorithm 2

Algorithm 2: $Compatible(P_1, P_2)$

Input: $P_1 = \{(p_1, \mathcal{E}_1), ..., (p_n, \mathcal{E}_n)\}, P_2 = \{(p'_1, \mathcal{E}'_1), ..., (p'_m, \mathcal{E}'_m)\}$
Output: True or False

$sp_1 \leftarrow scope(P_1)$;
$sp_2 \leftarrow scope(P_2)$;
$P'_1 \leftarrow \{(p_1 \cap sp_2, \mathcal{E}_1), ..., (p_n \cap sp, \mathcal{E}_m)\}$;
// if $p_i \cap sp_2 = \emptyset$, exclude $(p_i \cap sp_1, \mathcal{E}_i)$ from P'_1
$P'_2 \leftarrow \{(p'_1 \cap sp_1, \mathcal{E}'_1), ..., (p'_n \cap sp_1, \mathcal{E}'_m)\}$;
// if $p'_i \cap sp_1 = \emptyset$, exclude $(p'_i \cap sp_1, \mathcal{E}'_i)$ from P'_2
if $P'_1 = P'_2$ **then**
    return True;
else
    return False;
end

Algorithm 3

Algorithm 3: $PossibleMatch((p, \mathcal{E}), (p', \mathcal{E}'))$

Input: $(p, \mathcal{E}), (p', \mathcal{E}')$
Output: Solutions $\{..., (u, v), ...\}$ or 'no match'

$min_1 \leftarrow min(\{\alpha \mid (\alpha, \beta) \in \mathcal{E})\})$;
$min_2 \leftarrow min(\{\alpha' \mid (\alpha', \beta') \in \mathcal{E}')\})$;
$\overline{\mathcal{E}} \leftarrow \{(\alpha / min_1, \beta)) \mid (\alpha, \beta) \in \mathcal{E}\}$;
$\overline{\mathcal{E}'} \leftarrow \{(\alpha' / min_2, \beta')) \mid (\alpha', \beta') \in \mathcal{E}'\}$;
$min_2 \leftarrow min(cons(\mathcal{E}'))$;
if $\overline{\mathcal{E}} = \overline{\mathcal{E}'}$ **then**
    /* This homogeneous linear diophantine equation can be solved using with help of Eclid's GCD algorithm */
    Solve $min_1 u = min_2 v$;
    if $min_1 u = min_2 v$ *has solutions* **then**
        return $\{..., (u, v), ...\}$;
        // $\{..., (u, v), ...\}$ stands for a set of solutions
    else
        return $\emptyset$;
    end
else
    return $\emptyset$;
end

Algorithm 4

Algorithm 4: $Merge(R, R')$

Input: R, R'
Output: R''

$P \leftarrow PartitionEventsSet(R)$;
$P' \leftarrow PartitionEventsSet(R')$;
// $P = \{(p_1, \mathcal{E}_1), ..., (p_i, \mathcal{E}_i), ..., (p_n, \mathcal{E}_n)\}$;
// $P' = \{(p'_1, \mathcal{E}'_1), ..., (p'_j, \mathcal{E}'_j), ..., (p'_m, \mathcal{E}'_m)\}$;
for $(p, \mathcal{E}) \in P$ **do**
  for $(p', \mathcal{E}') \in P'$ **do**
    $M \leftarrow PossibleMatch((p, \mathcal{E}), (p', \mathcal{E}'))$;
    if $M = \emptyset$ **then**
      continue;
    else
      $(u, v) \in M$;
      if $Compatible(uP, vP') = False$ **then**
        Solve $Compatible(uP + x, vP' + y) = True$;
      else
        $R'' \leftarrow uR \cup vR'$;
      end
    end
  end
end
return R'';

- $action(\mathcal{E}) = \{\beta \mid (\alpha, \beta) \in \mathcal{E}\}$.
- $consumption(\mathcal{E}) = \{\alpha \mid (\alpha, \beta) \in \mathcal{E}\}$.
- Example: $\mathcal{E}_i = \{(-1, \lambda), (-2, a)\}$.
- Example: $\mathcal{E}'_j = \{(-2, \lambda), (-4, a)\}$.
- $action(\mathcal{E}_i) = \{\lambda, a\}$. $consumption(\mathcal{E}_i) = \{-1, -2\}$.
- $action(\mathcal{E}'_i) = \{\lambda, a\}$. $consumption(\mathcal{E}'_i) = \{-2, -4\}$.
- Example: $\mathcal{E}_k = \{(-1, a), (-2, \lambda)\}$.
- Example: $\mathcal{E}'_l = \{(-1, \lambda), (-2, a)\}$.
- $action(\mathcal{E}_k) = \{\lambda, a\}$. $consumption(\mathcal{E}_k) = \{-1, -2\}$.
- $action(\mathcal{E}'_l) = \{\lambda, a\}$. $consumption(\mathcal{E}_k) = \{-1, -2\}$.

Algorithm 5: Homogenization Algorithm

Input: $\{R_1, ..., R_p\}$
Output: R_0
$R_0 \leftarrow \{\}$;
for *each* $R \in \{R_1, ..., R_p\}$ **do**
  $R_0 \leftarrow Merge(R_0, R)$;
end
return R_0;

4. Remarks and Conclusions

References

[1] Chen, H., Ionescu, M., Ishdorj, T.-O., Păun, A., Păun, G., Pérez-Jiménez, M. J., June 2008. Spiking neural p systems with extended rules: Universality and languages. Natural Computing 7 (2), 147–166.
DOI: 10.1007/s11047-006-9024-6

[2] Ionescu, M., Păun, G., Yokomori, T., February 2006. Spiking neural p systems. Fundamenta Informaticae 71 (2,3), 279–308.

[3] Jiang, K., Chen, W., Zhang, Y., Pan, L., Jan 2016. Spiking neural p systems with homogeneous neurons and synapses. Neurocomputing 171, 1548–1555.
DOI: 10.1016/j.neucom.2015.07.097

[4] Jiang, K., Song, T., Chen, W., Pan, L., 2013. Homogeneous spiking neural P systems working in sequential mode induced by maximum spike number. International Journal of Computer Mathematics 90 (4), 831–844.

[5] Song, T., Wang, X., Aug 2015. Homogeneous spiking neural p systems with inhibitory synapses. Neural Processing Letters 42 (1), 199–214.
DOI: 10.1007/s11063-014-9352-y

[6] Song, T., Wang, X., Zhang, Z., Chen, Z., Jun 2014. Homogenous spiking neural p systems with anti-spikes. Neural Computing and Applications 24 (7), 1833–1841.
DOI: 10.1007/s00521-013-1397-8

[7] Zeng, X., Zhang, X., Pan, L., Jan. 2009. Homogeneous spiking neural p systems. Fundam. Inf. 97 (1-2), 275–294.