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
[?] [?]
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 an alphabet V, as follows: (1) each $a \in V$ and $\emptyset$ are regular expressions, (2) if

*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)

E_1 and E_2 are regular expressions, then $E_1 \cup E_2$, E_1E_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_1E_2 is $L(E_1E_2) = \{xy | x \in L(E_1) \text{ and } y \in L(E_2)\}$, 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

[**?**]

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 $E/a^c \to a^p{:}d$ 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 $E/a^c \to a^p{:}d$ 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 $E/a^c \to a^p{:}d$ 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 $E_1/a^{c_1} \to a^{p_1}{:}d_1$ and rule $E_2/a^{c_2} \to a^{p_2}{:}d_2$ *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 $E/a^c \to a^p{:}d$, if $E = a^c$, we can write the rule as $a^c \to a^p{:}d$. If $d = 0$, we can write the rule as $E/a^c \to a^p$. If $E = a^c$ and $d = 0$, we can write the rule as $a^c \to a^p$. 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 [**?**] only uses forgetting and standard spiking rules. A forgetting rule in the original SNP system has the form $a^c \to a^0$, written as $a^c \to \lambda$, 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 [**?**] introduces the idea of the extended spiking rule but the actual SNP system model in [**?**] 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.

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

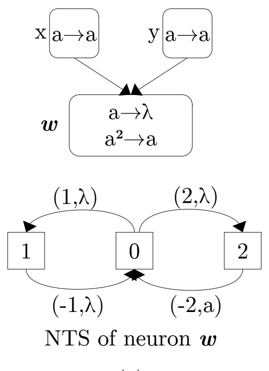
Definition 2 (Neuron Transition System)**.** A *neuron transition system* (NTS) is a tuple $(S, V, \rightarrow)$ where

- 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 $(s, (\alpha, \beta))$.

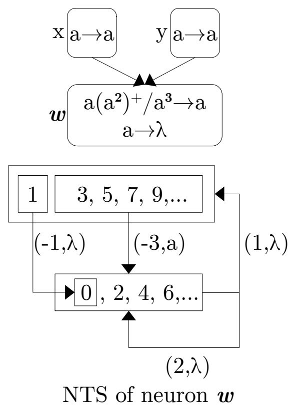
Let $nts = (S, V, \rightarrow)$ be the NTS of some neuron $\sigma_w = (n, R)$. nts is constructed using the rule set R and the possible spike trains going to neuron w. For each rule $E/a^\alpha \rightarrow \beta$ in R, the transition $(N(L(E)), (-\alpha, \beta))$ is an element of the transtion relation $\rightarrow$. $N(L(E))$ is the length set of language $L(E)$. We will call the set that contains neuron w and all neurons *connected to* neuron w, all neuron x such that $(x, w) \in syn$, as the *neuron w subsystem.* If it is possible for the neurons connected to neuron w to send α spikes to neuron w 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 the transition $(s, (\alpha, \lambda)) \in\rightarrow$ represents neuron w receiving α spikes when neuron w is in state s and the transition $(s, (\alpha', \beta)) \in\rightarrow$ represents an application of a rule that consumes α' spikes and has action β when neuron w is in state s, then the transition $(s, (\alpha + \alpha', \beta))$ is also an element of $\rightarrow$. The event $(\alpha + \alpha', \beta)$ actually represents two events, the reception of α spikes and the application of a rule that consumes α' spikes and has action β.

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 $(\{2i + 1\}_{i\geq1}, (-3, a))$ corresponds to the application of rule $a(a^2)^+/a^3 \rightarrow a$. The transitions $(\{2i\}_{i\geq0}, (1, \lambda))$ and $(\{2i\}_{i\geq0}, (2, \lambda)$ 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.



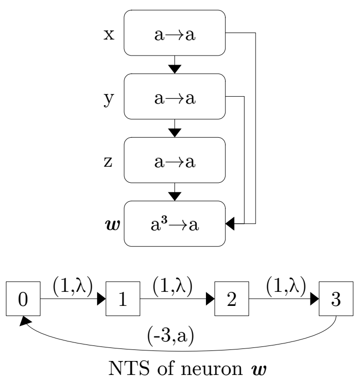
(a)



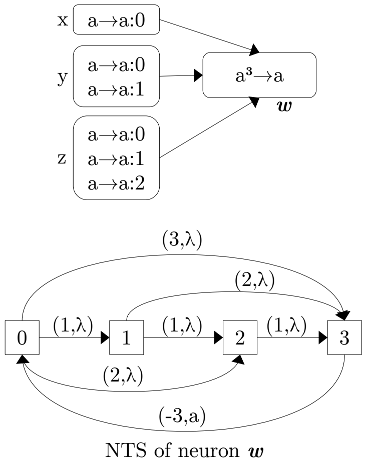
(b)

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 $a^3 \to a$ 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.



(a)



(b)

Figure 2: Neurons with same rule set but different NTSs

3.2. Operations on Neuron Transition Systems

Definition 3 (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.*
- $\rightarrow' = \{t + \delta \mid t \in \rightarrow\}$. If $t = (s, (\alpha, \beta))$, $t + \delta = (s + \delta, (\alpha, \beta))$. We say that $t + \delta$ is *transition t translated by δ.* The translation of a transition t by δ is simply the translation of the state component s by δ.

We denote nts' as $nts + \delta$. We say that $nts + \delta$ is the neuron transition system *nts translated by δ.*

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 4 (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 $R' = \{a^\delta E/a^c \rightarrow \beta \mid E/a^c \rightarrow \beta \in R\}$.

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 $E/a^c \rightarrow \beta$ 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 $a \rightarrow \lambda$ and $a^2 \rightarrow a$. Neuron w', being a δ-translated neuron w, has δ initial spikes and has rules $a^{\delta+1}/a \rightarrow \lambda$ and $a^{\delta+2}/a \rightarrow a$. 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.

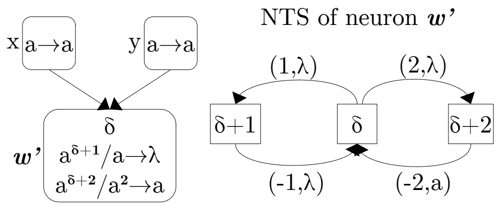


Figure 3: Neuron w' and its NTS

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 $n + \delta$. $\square$

Definition 5 (NTS Scaling)**.** *NTS scaling* 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:

- $S' = \{\delta s \mid s \in S\}$. $\delta s = \{\delta i \mid i \in s\}$. We say that δs is *state s scaled by δ.*
- $V' = \{(\delta\alpha, \beta) \mid (\alpha, \beta) \in V\}$.
- $\rightarrow' = \{\delta t \mid t \in \rightarrow\}$. If $t = (s, (\alpha, \beta))$, $\delta t = (\delta s, (\delta\alpha, \beta))$. We say that δt is *transition t scaled by δ.* The scaling transition t by δ 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 the neuron transition system *nts scaled by δ.*

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 6 (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:

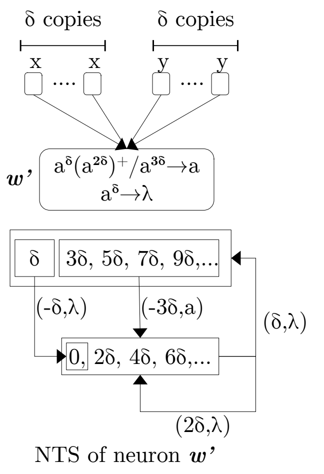
- sub' contains neuron $\sigma' = (\delta n, R')$. $R' = \{\delta E/a^{\delta c} \rightarrow \beta \mid E/a^c \rightarrow \beta \in R\}$. δE is the regular expression that is the result of replacing all instances of subexpression a in E by the subexpression a^δ. i.e. If $E = a(a^2)^*$, then $\delta E = a^\delta((a^\delta)^2)^* = a^\delta(a^{2\delta})^*$. 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'.

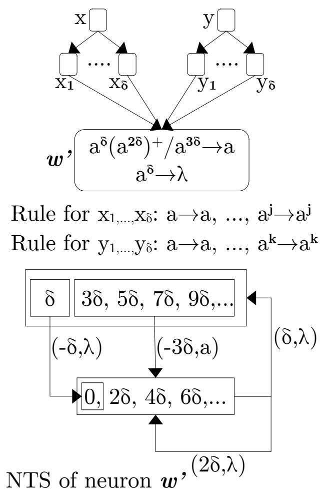
In *Type 2* subsystem scaling, sub' also contains the same neuron $\sigma' = (\delta n, R')$ described above. If in subsystem sub, neuron x is connected to neuron σ, then neuron x will also be in subsystem sub' and for each neuron x there will be additional δ neurons called $x_1, ..., x_\delta$ that have the same rule set with rules of the form $a^j \rightarrow a^j$ where j is a number of spikes a rule in neuron x can produce. Instead of neuron x being connected to neuron σ', neuron x will be connected to neurons $x_1, ..., x_\delta$ while neurons $x_1, ..., x_\delta$ will be connected to neuron σ'.

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 that is the result of a type 1 subsystem δ-scaling of the neuron w subsystem in Figure 1b.



(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

SECTION INTRO PARAGRAPH

Definition 7 (Rule Transitions)**.** A rule set R is a set of rules $\{r_1, ..., r_i, ..., r_n\}$ where each rule r_i has the form (s_i, a_i). For rule $r_i = (s_i, a_i)$, s_i is called the state while a_i is called action. The *scope* of a rule set $R = \{(s_1, a_1), ..., (s_n, a_n)\}$, denoted by *scope*$(R)$, is the state $s = s_1 \cup s_2 \cup \cdots \cup s_n$.

SHORT INFO ON RULE SET, why rule set

Definition 8 (State-Actions Pair)**.** Given a rule set $R = \{(s_1, a_1), ..., (s_n, a_n)\}$, we can associate a set of actions to any spike count c in the scope of the rule set R. Let spike count c be in the scope of R, $c \in$ *scope*(R), the actions associated with spike count c, denoted by *act*(c), is the set $\{a_i \mid (s_i, a_i) \in R, c \in s_i\}$. We say that "*act*$(c)$ is the set of actions at spike count c". A set of actions can also be associated with a set of spike counts instead of simply being associated with a single count. We say that set of spike counts C is associated with actions *act*(C) if for all $c \in C$ the actions *act*(c) at spike count c is equal to *act*(C). i.e. All spike counts in C have the same set of actions. spike count,

MOTIVATION EXAMPLES

Definition 9 (Partition-Actions Set)**.** DEF

For example, $R = \{(s_1, a_1), (s_2, a_2), (s_3, a_3)\}$ $A = s_1 \backslash (s_2 \cap s_3)$ $B = s_2 \backslash (s_1 \cap s_3)$ $C = s_3 \backslash (s_1 \cap s_2)$ $D = (s_1 \cup s_2) \backslash s_3$ $E = (s_1 \cup s_3) \backslash s_2$ $F = (s_2 \cup s_3) \backslash s_1$ $G = (s_1 \cup s_2 \cup s_3)$

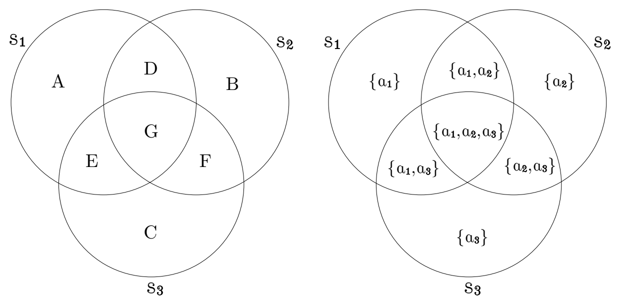


Figure 5: Partitions and Actions

ELABORATE AND GIVE EXAMPLES(3)

Algorithm 1: $GetPartitionActionsSet(R)$:
Get the Partition-Actions Set that corresponds to rule transition set R

```
Input: $R = \{(s_1, a_1), ..., (s_n, a_n)\}$
Output: $P = \{(p_1, A_1), ..., (p_m, A_m)\}$
$P \leftarrow \{\}$;
for each $T \subseteq R$ do
    $S \leftarrow \{s_k \mid (s_k, a_k) \in T\}$ ;
    $S' \leftarrow \{s_l \mid (s_l, a_l) \in R \backslash T\}$ ;
    $A_i \leftarrow \{a_k \mid (s_k, a_k) \in T\}$ ;
    $p_i \leftarrow \Big(\bigcap_{s_k \in S} s_k\Big) \Big\backslash \Big(\bigcup_{s_l \in S'} s_l\Big)$ ;
    if $p_i \neq \varnothing$ then
        Add $(p_i, A_i)$ to $P$ ;
    end
end
for each $(p_x, A_x), (p_y, A_y) \subseteq P$ do
    if $A_x = A_y$ then
        Remove $(p_x, A_x)$ and $(p_y, A_y)$ from $P$;
        Add $(p_x \cup p_y, A_x)$ to $P$ ;
    end
end
```

Definition 10 (Compatibility of Two Partition-Actions Sets)**.** Two partition-actions sets, $P = \{(p_i, A_i)\}$ and $P' = \{(p'_j, A'_j)\}$, are compatible if for all $c \in$ *scope*$(P) \cup$ *scope*(P') there is a $(p_i, A_i) \in P$ and a $(p'_j, A'_j) \in P'$ such that $c \in p_i$, $c \in p'_j$, and $A_i = A'_j$.

Algorithm 2: *Compatible*(P, P')
Return *true* if partition-actions set P is compatible with partition-actions set P'

```
Input: $P = \{(p_1, A_1), ..., (p_n, A_n)\}, P' = \{(p'_1, A'_1), ..., (p'_m, A'_m)\}$
Output: True or False
$s \leftarrow scope(P)$;
$s' \leftarrow scope(P')$;
$P \leftarrow \{(p_1 \cap s', A_1), ..., (p_n \cap s', A_n)\}$;
$P' \leftarrow \{(p'_1 \cap s, A'_1), ..., (p'_n \cap s, A'_m)\}$;
if $P = P'$ then
    return true;
else
    return false;
end
```

Algorithm 3: 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 $R_0$;
```

Algorithm 4: Merging Two Rule Sets

```
Input: $R_x, R_y$
Output: $R_z$
Convert rule set $R_x$ to partition-actions set $P_x$;
Convert rule set $R_y$ to partition-actions set $P_y$;
if $P_x$ is compatible with $P_x$ then
    $R_Z \leftarrow R_x \cup R_y$;
else
    Match $P_x$ and $P_y$;
    for each $(p_i, A_i) \in P_x$ and each $(p_j, A_j) \subseteq P_y$ do
    end
end
return $R_Z$;
```

Algorithm 5: Matching of Two Partition-Actions Pairs

```
Input: $(p, A = (\alpha, \beta)), (p', A' = (\alpha', \beta'))$
Output: (matching parameters)/no match
if $\beta \neq \beta'$ then
    return 'no match';
else
    Solve $u_1\alpha + v_1 = u_2\alpha' + v_2$;
    //This linear diophantine equation is solved using Euclidean GCD
      algorithm;
end
return $(u_1, v_1, u_2, v_2)$;
```

4. Remarks and Conclusions