Homogenization of Spiking Neural P Systems

Ren Tristan A. de la Cruz[1]

[1]Algorithms and Complexity Laboratory
Department of Computer Science, University of the Philippines - Diliman
Diliman 1101, Quezon City, Philippines
radelacruz@up.edu.ph,

Abstract. (ABSTRACT)

Keywords: Membrane Computing, Spiking Neural P Systems, Homogeneous Neurons, Structural Plasticity

1 Introduction

2 Spiking Neural P System and Some Variants

3 Homogenization of Spiking Neural P Systems

3.1 Representing Neurons as Labelled Transition Systems

We will use the *labelled transition system* construct to represent the activities and behavior of neurons in SNP systems. Each neuron in a SNP system will have a corresponding labelled transition system.

Definition 1 (Labelled Transition System)**.** A *labelled transition system (LTS)* is a tuple $(S, L, \rightarrow)$ where S is a *set of states*, L is a *set of labels*, and $\rightarrow \subseteq S \times L \times S$ is a *transition relation*.

We will call an LTS that represents an SNP system neuron as a *neuron LTS*. Neuron LTSs use a particular set of states and a particular set of labels that are relevant to SNP systems.

States of neuron LTSs are a sets of natural numbers that represents sets of spike counts. For example, the state $\{4, 5\}$ represents spike counts 4 and 5, the state $\{0, 2, 4, 8, ...\}$ represents even spike counts, and the state $\{15, 20, 25, 30, ...\}$ represents spike counts that are multiples of 5 greater than or equal to 15. Since a state s is a set of natural numbers, the set of states S of some neuron LTS $(S, L, \rightarrow)$ is a set of subsets of natural numbers. i.e. For a neuron LTS $(S, L \rightarrow)$, $S \subseteq \mathcal{P}(\mathbb{N})$ where $\mathcal{P}(\mathbb{N})$ is the power set of $\mathbb{N}$.

Labels of neuron LTSs represent receptions of spikes or rule applications. Labels will have the form (α, β) where α is an integer that represents the number of spikes consumed if $\alpha < 0$ or the number of spikes received if $\alpha > 0$ while β presents some action, spiking or forgetting, during rule application. For example,

let labels with $\beta = 0$ represent applications of forgetting rules while labels with $\beta = 1$ represent applications of spiking rules, the label $(-3, 0)$ represents the application of a forgetting rule that consumes 3 spikes while the label $(-2, 1)$ represents the application of a spiking rule that consumes 2 spikes. Labels that represent receptions of spikes will have $\alpha > 0$ and $\beta = 0$. For example, the label $(4, 0)$ represents the reception of 4 spikes while the label $(1, 0)$ represents the reception of 1 spike.

For any label (α, β), $\alpha \in \mathbb{Z}$ while β is an element of some *set of actions* $\mathcal{A}$ that is relevant to SNP systems. In the previous examples, we use $\mathcal{A} = \{0, 1\}$ where 0 represents a forgetting action and 1 represents a spiking action. The assumption when using $\mathcal{A} = \{0, 1\}$ as the set of actions is that we are only dealing with SNP systems that do not use delays (all spiking rules have delays set to 0). For SNP systems in general, $\mathcal{A} \subseteq \mathbb{N}$ can be used as the set of actions. For a $\beta \in \mathcal{A}$, if $\beta = 0$ then β represents a forgetting action and if $\beta = d > 0$ then β represents a spiking action with d-1 delay. For SNP systems with extended rules, $\mathcal{A} \subseteq \mathbb{N} \times \mathbb{N}$ can be used as the set of actions. $\beta = (p, d) \in \mathcal{A}$ represents an action of producing p spikes after a d delay. For example, $\beta = (3, 2)$ represents the action of producing 3 spikes after 2 steps, $\beta = (5, 0)$ represents the action of producing 5 spikes immediately, and $\beta = (0, 0)$ represents the forgetting action. For SNP systems with extended rules, we can have labels like $(-6, (5, 0))$ that represents consumption of 6 spikes and immediate production of 5 spikes, $(-5, (3, 2))$ that represents consumption of 5 spikes and production of 3 spikes after 2 steps, and $(-7, (0, 0))$ that represents consumption of 7 spikes (forgetting action). In general, the set of labels that will be used by neuron LTSs is $L \subseteq \mathbb{Z} \times \mathcal{A}$ where $\mathcal{A}$ is a set of actions relevant to the SNP system variant.

The *transition relations* of neuron LTSs will contain *transitions* of the form $(s, (\alpha, \beta), s')$ where s and s' are states and (α, β) is a label. The transition relation describes how the number of spikes in a neuron changes due to incoming spikes (events) or due to rule applications (actions). If a neuron has n spikes and $n \in s$, then we say that the neuron *is in state* s. The transition $(s, (\alpha, \beta), s')$ means that if the neuron is in state s and the event/action (α, β) occurred, then the neuron will transition to having $n + \alpha$ spikes which is state s' (the neuron *transitions to state* s'). For example, the transition $(\{2, 4, 6, 8, ...\}, (-1, 1), \{1, 3, 5, 7, ...\})$ means that if the neuron has n spikes which is even (the neuron is in the 'even spike count' state) and the action $(-1, 1)$ occurred (a spiking rule that consumes 1 spike is applied), then the neuron will transition to having n-1 spikes (the neuron transitions to the 'odd spike count' state). The transition $(s, (\alpha, \beta), s')$ can simply be written as $(s, (\alpha, \beta))$ because the next state s' can be derived from the current state s and the α component of the label (α, β). The next state is defined as $s' = \{n + \alpha \mid n \in s\}$.

For a transition $(s, (\alpha, \beta))$ that represents an SNP system rule, state s represents the set of spike counts where the rule is applicable, $\alpha < 0$ represents the number of spikes the rule consumes, and β represents the action performed by the rule (spiking or forgetting). Figure 1 shows two examples of neurons and their LTSs. Figure 1a shows neuron w with two incoming synapses, one from

neuron x and another from neuron y, and it contains the rules $a/a \to \lambda$ and $a^2/a^2 \to a$. Rule $a/a \to \lambda$ is represented by the transition $(\{1\}, (-1, 0))$. The state $\{1\}$ means the rule can only be applied when the neuron has 1 spike. The label $(-1, 0)$ means the rule consumes 1 spike ($\alpha = -1$) and is a forgetting rule ($\beta = 0$). Rule $a^2/a^2 \to a$ is represented by the transition $(\{2\}, (-2, 1))$. The state $\{2\}$ means the rule can only be applied when the neuron has 2 spikes while the label $(-2, 1)$ means the rule is a spiking rule that consumes 2 spikes. In Figure 1a and the rest of the figures with LTS, a state is drawn as set of numbers inside a rectangle and a label is drawn as an arrow. The transition $(\{0\}, (+1, 0))$ represents the reception of one spike while the neuron has 0 spikes while the transition $(\{0\}, (+2, 0))$ represents the reception of 2 spikes while the neuron has 0 spikes.

Figure 1b shows another example of a neuron and its LTS. Figure 1b shows neuron w with two incoming synapses, one from neuron x and another from neuron y, and it contains the rules $a(a^2)^+/a^3 \to a$ and $a/a \to \lambda$. Rule $a(a^2)^+/a^3 \to a$ is represented by the transition $(\{3, 5, 7, 9, ...\}, (-3, 1))$ which means it is applicable when the neuron has an odd number of spikes equal that is at least 3, it consumes 3 spikes ($\alpha = -3$), and it is a spiking rule ($\beta = 1$). Rule $a/a \to \lambda$ is represented by the transition $(\{1\}, (-1, 0))$ which means it is only applicable when the neuron has 1 spikes and it is a forgetting rule ($\beta = 0$) that consumes 1 spike ($\alpha = -1$). The transition $(\{0, 2, 4, 6, ...\}, (+2, 0))$ means that if then neuron has even number of spikes (in the even state) and it receives 2 spikes ($\alpha = +2$) the neuron will simply stay in the even state. The transition $(\{0, 2, 4, 6, ...\}, (+1, 0))$ means that if the neuron is in the even state and it receives 1 spike then the neuron will transition to the odd state (neurons will now have odd number of spikes).

To construct the LTS of some neuron w, you need to know its rule set and the behavior of the subsystem of neurons connected to neuron w (neurons that have synapses going to neuron w). From the rule set, you get the transitions that represent the rules. The other transitions that represent receptions of spikes are derived from the behavior of subsystem of neurons connected to neuron w. This means that neurons with the same rule set may have different LTSs. Figure 2 shows two different LTSs for neuron w. In Figures 2a and 2b, neuron w has the single rule $a^3/a^3 \to a : 0$ but the subsystem connected to neuron w in Figure 2a is different from the subsystem connected to neuron w in Figure 2b. In Figure 2a, neurons x, y, and z give one spike each to neuron w one spike at a time. The transition $(\{0\}, (+1, 0))$ represents neuron x sending one spike to neuron w when neuron w has 0 spike. The transition $(\{1\}, (+1, 0))$ represents neuron y sending a spike to neuron w when neuron w has one spike. And, the transition $(\{2\}, (+1, 0))$ represents neuron z sending a spike to neuron w when neuron w has two spikes. Those 3 transitions along with the transition $(\{3\}, (-3, 1))$ for the rule are all the transitions for neuron w's LTS.

In Figure 2b, starting with neuron w having 0 spikes, there are 4 scenarios that lead to neuron w having 3 spikes and these scenarios are: (scenario 1) neurons x, y, z send one spike each to neuron w at the same time [transition sequence:

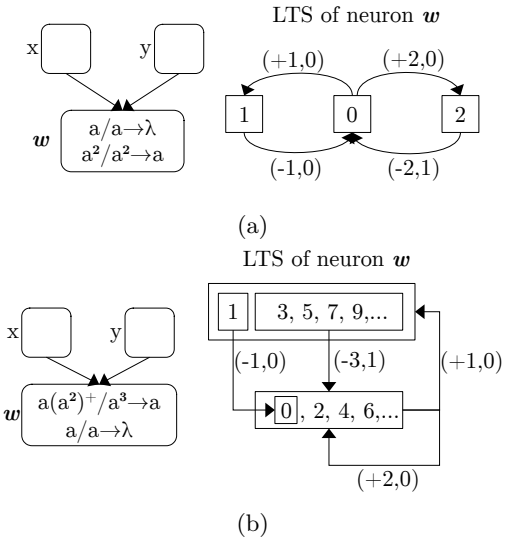
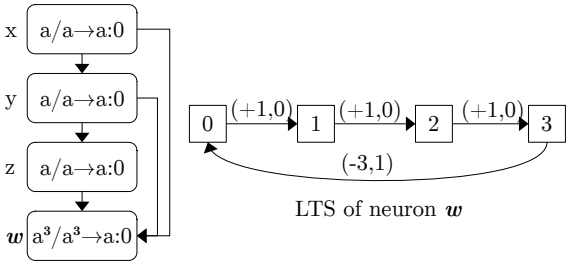


Fig. 1: Examples of Neurons and their LTSs

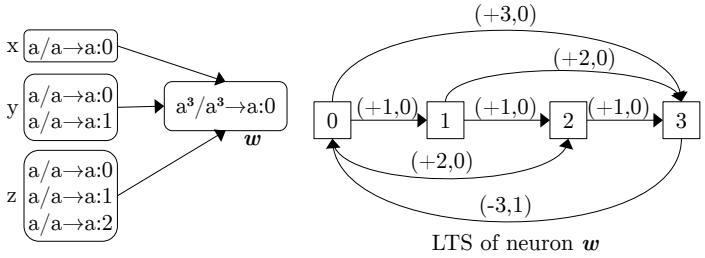
$(\{0\}, (+3, 0))]$, (scenario 2) neuron x sends a spike, then neurons y and z send a spike each at the same time [transition sequence: $(\{0\}, (+1, 0)), (\{1\}, (+2, 0))]$, (scenario 3) neurons x and y send one spike each to neuron w at the same time, then neuron z sends a spike [transition sequence: $(\{0\}, (+2, 0)), (\{2\}, (+1, 0)))]$, and (scenario 4) neuron x sends one spike follow by neuron y sending another spike then neuron z sends the third spike [transition sequence: $(\{0\}, (+1, 0))$, $(\{1\}, (+1, 0)), (\{2\}, (+1, 0))]$. The two subsystems in Figure 2 produce different sequences of events which means different sets of transitions and therefore different LTSs for neuron w.

3.2 Operations on Labelled Transition Systems

We will define two operations on labelled transition systems, *LTS translation* and *LTS scaling*. The idea behind these operations is that they take an LTS and a parameter δ and produce a new LTS that behaves exactly like the original LTS. When you are combining two different rule sets from two different neurons in order to have common rule set, simply getting the union of the two rule sets can cause conflicts. For example, let $\{a \to a, a^2/a \to a\}$ be the rule set of neuron x and $\{a \to \lambda, a^2/a \to a\}$ be the rule set of neuron y. If we simply combine the rule sets to have the common rule set $\{a \to \lambda, a \to a, a^2/a \to a\}$ for both neurons x and y, there will be an unwanted behavior. If neuron x has 1 spike it should use the rule $a \to a$ and if neuron y has 1 spike it should use the rule $a \to \lambda$. If neurons x and y use the common rule set, when they have 1 spike both of them will have a non-deterministic choice, use rule $a \to \lambda$ or use rule $a \to a$. This non-determinism is an unwanted behavior that is the result of rule $a \to \lambda$



(a) Neuron w in a subsystem and neuron w's LTS



(b) Neuron w in a different subsystem with a different LTS

Fig. 2: Neurons with same rule set but different LTSs

'conflicting' with rule $a \to a$. The common rule set changes the behavior of both neuron x and neuron y. LTS translation and LTS scaling will be used to avoid such conflicts and changes in the neurons' behavior when combining different rule sets.

Definition 2 (State Translation)**.** *State translation* is an operation on a state. It takes a state s and a natural number δ and produces the state s' defined as $s' = \{n + \delta \mid n \in s\}$. We denote state translation with the $+$ symbol. i.e. $s + \delta = \{n + \delta \mid n \in s\}$. We say that s' is *s translated by δ*.

For example, if $s_1 = \{3, 5, 7\}$ and $\delta_1 = 3$, then $s_1 + \delta_1 = s_1 + 3 = \{6, 8, 10\}$. If $s_2 = \{3, 6, 9, 12, ...\} = \{3i\}_{i\geq 1}$ and $\delta_2 = 2$, then $s_2 + \delta_2 = s_2 + 2 = \{5, 8, 11, 14, ...\}$ $= \{3i + 2\}_{i\geq 1}$. If $s_3 = \{1, 3, 5, 7, ...\} = \{2i - 1\}_{i\geq 1}$ and $\delta_3 = 1$, then $s_3 + \delta_3 = s_3 + 1 = \{2, 4, 6, 8, ..\} = \{2i\}_{i\geq 1}$.

Definition 3 (Transition Translation)**.** *Transition translation* is an operation on a transition. It takes a transition $t = (s, (\alpha, \beta))$ and a natural number δ and produces the transition $t' = (s', (\alpha, i\beta))$ where $s' = s + \delta$ (state s is translated by δ). We also will use the same symbol $+$ for transition translation. i.e. $t' = t + \delta$. We say that t' is *t translated by δ*.

A translation of the transition $(s, (\alpha, \beta))$ simply involves the translation of the state s component. The label component (α, β) is not modified. Since some transitions represent rules, the change due to transition translation has a cor-

responding change to the rule that transition represents. If transition is translated by δ, the regular expression E of the rule is also translated by δ. i.e. $a^\delta E$ is E translated by δ. For example, if $E = a(a^2)^*$ then E translated by δ is $a^\delta a(a^2)^* = a^{\delta+1}(a^2)^*$. Let r be the rule $E/a^c \rightarrow \beta$ (where β is some action e.g. spiking, forgetting, spiking with delay, etc). We say some rule r' is rule r *translated by* δ if r' is the rule $a^\delta E/a^c \rightarrow \beta$. We will call this operation *rule translation* which is the exact analogue of transition translation.

Definition 4 (LTS Translation)**.** *LTS translation* is an operation on an entire transition system. It takes a labelled transition system $lts = (S, L, \rightarrow)$ and a natural number δ and produces the labelled transition system $lts' = (S', L, \rightarrow')$ where $S' = \{s + \delta \mid s \in S\}$ and $\rightarrow' = \{t + \delta \mid t \in \rightarrow\}$. We also use the same symbol + for LTS translation. i.e. $lts' = lts + \delta$. We say that lts' is *lts translated by* δ.

The translation of $lts = (S, L, \rightarrow)$ simply involves the translation of all transitions in the relation $\rightarrow$ by the same amount δ. This implies a change of the set of states S. The set of labels L is unchanged.

When you translate a neuron LTS, the translated LTS represents the same neuron behavior as the original LTS but the translated LTS is for a different neuron. The translated LTS is for the *translated neuron*. Let some neuron w have an initial number of spike n. We say some neuron w' is neuron w *translated by* δ if neuron w' has $n+\delta$ initial number of spikes and contains δ-translated rules of neuron w. We will call this operation *neuron translation* which is the exact analogue of LTS translation. Figure 3 shows neuron w' which is a δ-translated neuron w from Figure 1a. It also shows the LTS of neuron w' which is a δ-translated LTS of neuron w from Figure 1a. Neurons w and w' have the same behavior because for any sequence of events (sequence of receptions of spikes) they will have the same sequence of actions (sequence of applied rules).

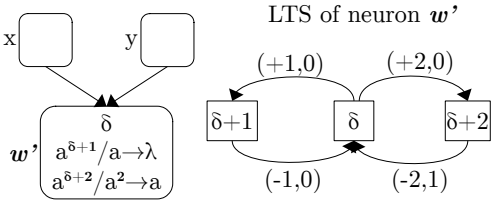


Fig. 3: Neuron w' and its LTS

Lemma 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 5 (State Scaling)**.** *State scaling* is an operation on a state. It takes a state s and a natural number δ and produces the state s' defined as $s' = \{\delta \cdot n \mid n \in s\}$. We denote state scaling with the $\cdot$ symbol. i.e. $s' = \delta \cdot s$. We say that s' is s *scaled by* δ.

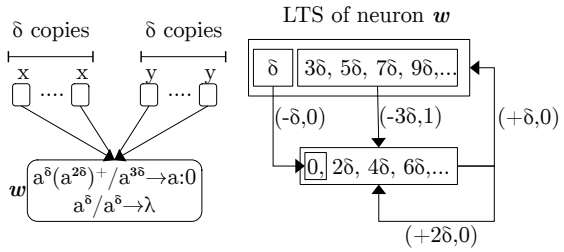
For example, if $s_1 = \{1, 2, 3\}$ and $\delta_1 = 5$, then $\delta_1 \cdot s_1 = 5 \cdot s_1 = \{5, 10, 15\}$. If $s_2 = \{2, 4, 6, 8, ...\} = \{2i\}_{i \geq 1}$ and $\delta_2 = 2$, then $\delta_2 \cdot s_2 = 2 \cdot s_2 = \{4, 8, 12, 16, ...\} = \{4i\}_{i \geq 1}$. If $s_3 = \{4, 7, 10, 13, ...\} = \{3i + 1\}_{i \geq 1}$ and $\delta_3 = 2$, then $\delta_3 \cdot s_3 = 2 \cdot s_3 = \{8, 14, 20, 26, ...\} = \{6i + 2\}_{i \geq 1}$.

Definition 6 (Transition Scaling)**.** *Transition scaling* is an operation on a transition. It takes a transition $t = (s, (\alpha, \beta))$ and a natural number δ and produces the transition $t' = (s', (\alpha', \beta))$ where $s' = \delta \cdot s$ and $\alpha' = \delta \cdot \alpha$. We also use the $\cdot$ symbol for transition scaling. i.e. $t' = \delta \cdot t$. We say that t' is t *scaled by* δ.

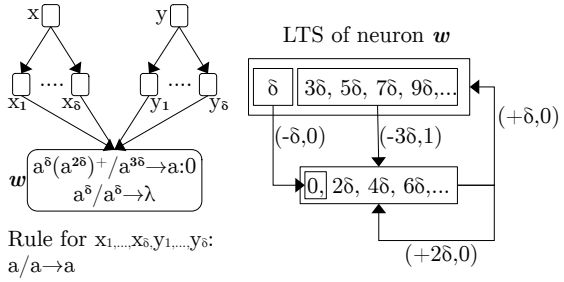
Transition scaling scales both the state component s and the α component (number of spikes consumed or received) of the label (α, β) by the same factor δ. Since some transitions represent rules, similar to transition translation there is a rule analouge for transition scaling which we will call *rule scaling*. If r is the rule $E/a^c \to \beta$, r' is rule r *scaled by* δ, written as $r' = \delta \cdot r$, if it is the rule $E'/a^{\delta \cdot c} \to \beta$ where E' is the modified expression E where all instances of expression a is replace by expression a^δ. For example, if r is the rule $a(a^3)^*/a^2 \to a$ with the corresponding transition $(\{3i + 1\}_{i \geq 0}, (-2, 1))$ then $r' = \delta \cdot r$ is the rule $(a^\delta)((a^\delta)^3)^*/a^{2\delta} \to a$, or simply $a^\delta(a^{3\delta})^*/a^{2\delta} \to a$, with the corresponding transition $(\{3i\delta + \delta\}_{i \geq 0}, (-2\delta, 1))$.

Definition 7 (LTS Scaling)**.** *LTS scaling* is an operation on an entire transition system. It takes a labelled transition system $lts = (S, L, \to)$ and a natural number δ and produces the labelled transition system $lts' = (S', L', \to')$ where $S' = \{\delta \cdot s \mid s \in S\}$, $L' = \{(\delta \cdot \alpha, \beta) \mid (\alpha, \beta) \in L\}$, and $\to' = \{\delta \cdot t \mid t \in \to\}$. We also use the $\cdot$ symbol for LTS scaling. i.e. $lts' = \delta \cdot lts$. We say that lts' is lts *scaled by* δ.

Fig. 4: Neurons with Same Rule Set but Different LTSs



(a) Neuron w in a Subsystem and Neuron w's LTS



(b) Neuron w in a Different Subsystem with a Different LTS

3.3 Procedures for Homogenizing Neurons' Rule Sets

Definition 8 (Rule Set)**.** 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 $s = s_1 \cup s_2 \cup \cdots \cup s_n$.

ELABORATE AND GIVE EXAMPLES(3)

Definition 9 (Scope Partitioning)**.** A partitioning of the scope of a rule set refers to the process of dividing the scope of the rule set into non-overlapping subsets. Let $R = \{(s_1, a_1), ..., (s_n, a_n)\}$ be a rule with scope $s = scope(R)$. A partitioning of s, denoted by $part(s)$, is any set $\{\overline{s}_1, ..., \overline{s}_k\}$ such that:

- $\overline{s} \subseteq s$, for each $\overline{s} \in part(s)$
- $\overline{s}_x \cap \overline{s}_y = \varnothing$, for every $\overline{s}_x, \overline{s}_y \in part(s)$
- $s = \bigcup_{\overline{s} \in part(s)} \overline{s}$, $part(s) = \{\overline{s}_1, ..., \overline{s}_k\}$

Any element of $\overline{s}$ of $part(s)$ is called a partition of s.

ELABORATE AND GIVE EXAMPLES(3)

Definition 10 (Actions)**.** 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,

Definition 1 (Partition-Actions Set). *A partition-actions set of a rule set* $R = \{(s_1, a_1), ..., (s_n, a_n)\}$

$$R = \{(s_1, a_1), (s_2, a_2), (s_3, a_3)\}$$

- $A = s_1 \backslash (s_2 \cap s_3)$
- $B = s_2 \backslash (s_1 \cap s_3)$
- $C = s_3 \backslash (s_1 \cap s_2)$
- $D = (s_1 \cup s_2) \backslash s_3$
- $E = (s_1 \cup s_3) \backslash s_2$
- $F = (s_2 \cup s_3) \backslash s_1$
- $G = (s_1 \cup s_2 \cup s_3)$

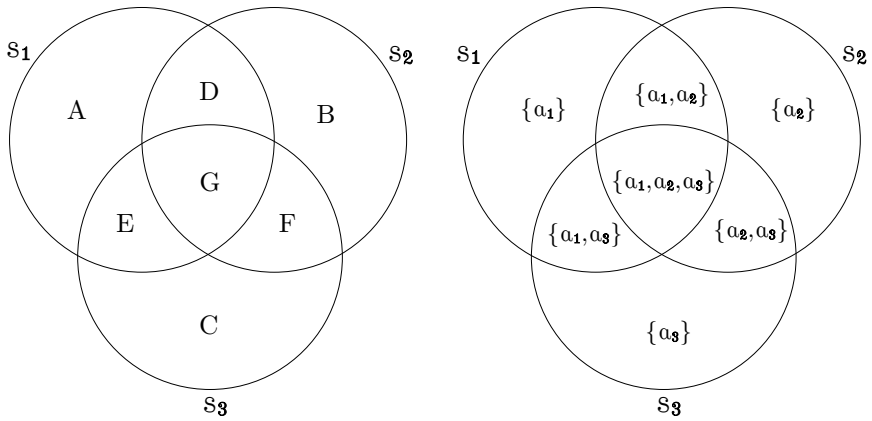


Fig. 5: Partitions and Actions

ELABORATE AND GIVE EXAMPLES(3)

Algorithm 1: Transforms a Rule Set to a Partition-Actions Set

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 11 (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: $Compatible(P, P')$

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

Fig. 6: Checks for Compatibility of Two Partition-Actions set

Definition 12 (Common Transition Set)**.**

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**
    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
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);