Research Article

Sequential Spiking Neural P Systems with Local Scheduled Synapses without Delay

Alia Bibi,1,2 Fei Xu,1 Henry N. Adorna,3 and Francis George C. Cabarle3,4

1Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
2Government Girls Postgraduate College, Kohat 26000, Khyber Pakhtunkhwa, Pakistan
3Department of Computer Science, University of the Philippines Diliman, Quezon City, Philippines
4School of Information Science and Engineering, Xiamen University, Xiamen 361005, China

Correspondence should be addressed to Fei Xu; fei_xu@hust.edu.cn

Received 7 December 2018; Revised 21 February 2019; Accepted 20 March 2019; Published 14 April 2019

Academic Editor: Alex Alexandridis

Copyright © 2019 Alia Bibi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Spiking neural P systems with scheduled synapses are a class of distributed and parallel computational models motivated by the structural dynamism of biological synapses by incorporating ideas from nonstatic (i.e., dynamic) graphs and networks. In this work, we consider the family of spiking neural P systems with scheduled synapses working in the sequential mode: at each step the neuron(s) with the maximum/minimum number of spikes among the neurons that can spike will fire. The computational power of spiking neural P systems with scheduled synapses working in the sequential mode is investigated. Specifically, the universality (Turing equivalence) of such systems is obtained.

1. Introduction

Spiking neural P systems (abbreviated as SNP systems) were first introduced in [1] as a class of parallel and distributed neural-like computational models, which are inspired from the fact that spiking neurons communicate with each other through electrical impulses. An SNP system could be viewed as a directed graph with nodes consisting of a number of computing units called neurons and the arcs representing the synapses. Each neuron contains copies of a single object type called the spike and a set of spiking rules that produces one or more spikes from the source neuron to every neuron connected by a synapse and forgetting rules that removes spikes from the neuron.

Many efforts have been made to investigate the theoretical and practical aspects of SNP systems, since the computational model was introduced. The computational power of SNP systems was investigated as different computing devices, e.g., number generating/accepting devices [1], language generators [2, 3], and function computing devices [4]. By abstracting computing ideas from the human brain and biological neurons, many variants of SNP systems have been introduced, such as axon P systems [5], SNP systems with rules on synapses [6–8], SNP systems with weights [9], SNP systems with neuron division and budding [10], SNP systems with structural plasticity [11], cell-like SNP systems [12], fuzzy reasoning SNP systems [13–16], probabilistic SNP systems [17], SNP systems with multiple channels [18], SNP systems with scheduled synapses (SSN P systems for short) [19], coupled neural P systems [20], dynamic threshold neural P systems [21], SNP systems with colored spikes [22], and SNP systems simulators [23, 24].

Although biological processes in living organisms happen in parallel, they are not synchronized by a universal clock as assumed in SNP systems. Introduced in [25], sequential SNP systems are a class of computing devices that plays a role of a bridge between spiking neural networks and membrane computing [26]. Two types of sequentiality are considered in sequential SNP systems: the general sequentiality [25] and sequentiality induced by the spike(s) number [27–29]. The first case is the classical pure-sequential model of its family. The fact behind the pure-sequentiality is that these systems are completely (purely) sequential according to neurons; at each time unit, one and only one neuron among all active...
neurons is nondeterministically chosen to fire. The second case is based on the first one, where the sequentiarity is induced on the basis of number of spikes (either maximum or minimum) present in active neurons; on the basis of spikes number we have further two types: maximal sequential SN P systems and minimal sequential SN P systems.

In this work, we consider SN P systems with scheduled synapses (SSN P systems) which is inspired and motivated by the structural dynamism of biological synapses, while incorporating ideas from nonstatic (i.e., dynamic) graphs and networks from mathematics. Synapses in SSN P systems are available at a specific schedule or duration. The schedules are of two types: local and global scheduling, named after a specified reference neurons: local reference neurons and global reference neurons. In the first case synapses are called local scheduled synapses, in which synapses are scheduled locally with respect to their local reference neurons; there can be multiple reference neurons within a system; however each synapse is associated with exactly one reference neuron. In the second case synapses are called global-scheduled synapses, in which synapses are scheduled globally with respect to a single-shared global reference neuron. In particular, we consider SN P systems with local scheduled synapses working in the sequential mode without delay (SSSN P systems, for short), where the sequentiarity induced by the maximum/minimum (max/min, for short) spike(s) number is introduced. The computational power of SSSN P systems working in strong sequentiarity or pseudosequentiability strategies is investigated. Specifically, SSSN P systems working in the max/min-sequentiarity (resp., max/min-pseudosequentiability) strategy are proved to be Turing universal.

The paper is organized as follows: Section 2 provides definitions of some concepts necessary for the understanding of the paper. We introduce in Section 3 our variant of SN P systems, namely, sequential SN P systems with local scheduled synapses working in max/min-sequential strategy without delay (SSSN P systems). We present our universality results in Section 4 and finally provide final remarks in Section 5.

2. Preliminaries

In this section, various notions and notations from formal language theory and computability theory that we shall use in the remainder of the paper are explained. The reader is also assumed to have a basic familiarity with membrane computing. Both the basics of membrane computing and its formal language and computability theory prerequisites are covered in sufficient detail in [30] and the handbook in [31].

We denote by Σ a nonempty finite set of symbols called alphabet, and Σ* is known as the set of all strings of symbols generated from Σ, including λ, the empty string. The set of nonempty string over Σ* is denoted by Σ+ = Σ* \ {λ}. Each L subset of Σ+ is called a language over Σ. A language L is called λ-free, if it does not contain empty string (thus is a subset of Σ*). A set of collection of languages is called a family of languages. We define inductively regular expressions over an alphabet Σ and the language they describe as follows: λ and every symbol a ∈ Σ are regular expressions. If E1 and E2 are regular expressions, then E1 + E2, E1E2, and E1* are also regular expressions. We denote by L(E1) the language described by the regular expression E1. In particular, we have L(λ) = {λ}, L(a) = {a}, L(E1 + E2) = L(E1) ∪ L(E2), L(E1E2) = L(E1)L(E2), and L(E1*) = L(E1)*.

Our universality results are proved by allowing our proposed model to simulate a well-known universal computing model called register machine [1,32].

A register machine M is a construct of the form: M = (m, H, l₀, lₗ, I), where

(i) m is the number of registers,
(ii) H is the set of instruction labels,
(iii) l₀ is the start label (which labels an instruction ADD),
(iv) lₗ is the label halt (which is assigned to instruction HALT),
(v) I is the set of instructions.

A register machine M could accept or generate a number n. In this work, we use a register machine M that generates numbers. We denote the set of all numbers computed/generated by M as N(M). Since register machine computes all sets of numbers that are Turing computable, then we have N(M) = NRE, where NRE denotes the family of languages recognized by Turing machine.

3. Sequential SN P Systems with Local Scheduled Synapses Working in Max/Min-Sequential Strategy without Delay

We give here the definition of sequential SN P systems with local scheduled synapses without delay.

Definition 1. A sequential SN P system with local scheduled synapses without delay, SSSN P system for short, of degree m ≥ 1 is a construct of the form

\[ \Pi = (\Sigma, \sigma₁, \sigma₂, \ldots, \sigmaₘ, \text{ref}, \text{syn}, \text{S, out}) \]

where

(a) \( \Sigma = \{a\} \) is the singleton alphabet (a is called spike);
(b) \( \sigma₁, \sigma₂, \ldots, \sigmaₘ \) are neurons of the form \( \sigmaᵢ = (nᵢ, Rᵢ) \), \( 1 ≤ i ≤ m \) where

1. \( nᵢ ≥ 0 \) is the initial number of spikes contained in \( \sigmaᵢ \);
2. \( Rᵢ \) is a finite set of rules of the form
   (i) \( E/\sigmaᵢ^{-} → aᵢ^p \), where \( E \) is a regular expression over \( a \);
   (ii) \( aᵢ^s → _λ \) for some \( s ≥ 1 \), with restriction that \( aᵢ^s \notin L(E) \) for any rule \( E/\sigmaᵢ^{-} → aᵢ^p \) of type (i) from \( Rᵢ \), where \( E \) is a regular expression over \( a \) and \( c ≥ 1, p ≥ 0, c ≥ p \);
3. \( \text{syn} \in \{1,2,\ldots,m\} × \{1,2,\ldots,m\} × (T ∪ \{\text{ON}\}) \) is the set of synapses among neurons, where \( T \) is the set of schedules of the form \((q,r) | q, r ≥ 0, q < r\), no \( \sigmaᵢ \) has a synapse to itself, and each synapse has exactly one schedule;
\( S \in \mathcal{P}(\text{syn}) \times \mathcal{P}(\text{ref}) \) is the set of synapse references, \( \mathcal{P}(x) \) is the power set of the set \( x \), and \( \text{ref} \subseteq \{1, 2, \ldots, m\} \) is the set of reference neurons. For any two pairs \((s_1, r_1), (s_2, r_2)\) \( \in S \), we have \( s_1 \cap s_2 = \emptyset \) and \( r_1 \cap r_2 = \emptyset \).

(e) \( \text{out} \) is the label for output neurons.

The semantics of spiking rule application is as follows: Let \( \sigma_i \) be a neuron containing \( b \) spikes and a rule \( E/a^c \to a^d \in R_i \). If \( b \geq c \) and \( a^d \in L(E) \), then \( E/a^c \to a^d \in R_i \) can be applied. With this rule application \( \sigma_i \) will consume \( c \) spikes to send \( p \) spike(s) to its adjacent neurons, then it will have \( b - c \) spikes remaining. If \( p = 1 \) then rule \( E/a^c \to a^d \in R_i \) serves as a standard rule and otherwise as an extended rule. In general, a neuron can have at least two spiking rules with regular expressions \( E_1 \) and \( E_2 \) such that \( L(E_1) \cap L(E_2) \neq \emptyset \). In this case, if the regular expression is satisfied, then we nondeterministically choose one of the rules to be applied. Additionally, our system is sequential and synchronized so that at each step, among all the active neurons, the one with maximum/minimum number of spikes can apply its rule. Rule application is sequential both at the neurons and at system level. This means at each time schedule exactly one rule can be applied by a spiking neuron and similarly one and only one neuron (holding maximum/minimum number of spikes) can fire among all the active neurons. Thus all the neurons apply their rules at maximum/minimum and in sequence. As a convention if a rule \( E/a^c \to a^d \in R_2 \) has \( E = a^i \) then we write it as \( a^i \to a^d \), instead. Moreover synapse from the output neuron labeled with ON is always assumed available.

The semantics of synapse schedule is as follows: the synapses are scheduled according to the activation of the reference neuron. A scheduled synapse takes effect immediately upon activation of its reference neuron. The reference neuron is identified with the symbol \( \Rightarrow^* \) (e.g., \( \sigma_i \)). If at step \( t \), a reference neuron \( \sigma_i \) becomes activated with a synapse scheduled with \( \langle q, r \rangle \), then it means that synapse is active only from the step \( t + q \) to \( t + r \). If at some step a rule is applied by a neuron \( \sigma_j \), but there is no respective scheduled synapse at that time, then that spike gets wasted as no adjacent neuron will receive it.

The semantics of sequentiaity either maximum/minimum is as follows: if at any step there are more than one active neurons (neurons with enabled rules are called active), then only the neuron(s) containing the maximum/minimum number of spikes (among the currently active neurons) will be able to fire. If there is a tie among two or more active neurons (all holding equal number of spikes), then there are two different strategies considered in SSSN P systems: strong sequentiality and pseudosequentiality. In the first strategy one and only one active neuron is nondeterministically chosen among all the tied neurons to fire, while in the second strategy all the tied neurons fire simultaneously; let us assume that, in max-sequential case, at a given step there are four active neurons: \( x_1, x_2, x_3, \) and \( x_4 \) with spikes stored \( 4, 5, 5, \) and \( 1 \), respectively; it is obvious that there is a tie between neurons \( x_2 \) and \( x_3 \); so in strong max-sequential case either neuron \( x_2 \) or \( x_3 \) will be nondeterministically chosen to fire, but in max-pseudosequential case both neurons will fire simultaneously.

A configuration of the system in a given step is the distribution of spikes among neurons and the status of each neuron, whether closed or open. The initial configuration is given by \( \eta \) of each neuron \( \sigma_i \) and all neurons are open.

Starting from the fixed initial configuration (distribution of spikes among neurons) and applying the rules in synchronized manner (a global clock is assumed), the system gets evolved. Applying the rules according to the above description, transitions among configuration can be defined.

Applying the rules in this way, the system passes from one configuration to another configuration. Such a step is known as transition. Given two configurations \( C_1, C_2 \) of \( \Pi \), the direct transition between these two configurations is represented by \( C_1 \Rightarrow C_2 \). While the reflexive and transitive closure of the relation \( \Rightarrow \) is represented by \( \Rightarrow^* \). A transition is sequential provided that, among all the candidate neurons, one holding maximum/minimum number of spikes must apply the rule. A computation is the sequence of transitions, starting from the initial configuration to the final configuration. A computation is said to be successful or a halting computation, if it reaches a configuration where no further rule(s) can be applied. Moreover all the neurons must be restored to their valid states.

There are various ways to define the result or output of the computation, but in this work we use the following type: we consider only the first two spikes fired or produced by the output neuron at steps \( t_1 \) and \( t_2 \). The number computed by SSSN P systems in max-sequential case is the difference between first two spikes minus one, i.e., \( t_2 - t_1 - 1 \), while in min-sequential case it is the difference between first two spikes minus two, i.e., \( t_2 - t_1 - 2 \).

We denote with \( N_a(\Pi^f_\alpha) \) the family of all sets of numbers generated by SSSN P systems working in sequential mode with local scheduled synapses and without delay. Here \( \alpha \in \{2, \text{gen}\} \) means that the number generated is encoded by the first two spikes of the output neuron; \( \text{gen} \) means that system works in generative mode; \( \beta = \text{loc} \) means only local scheduled synapses is used; \( \gamma \in \{\text{max,min,} \maxp, \text{minp}\} \) indicates the max/min-sequentiality and max/min-pseudosequentiality modes of the systems; \( e \) means extended rules. Moreover we will take into account only halting computation.

We follow the conventional practice by ignoring the number zero, when comparing the power of two number generating devices, as empty string is ignored in formal languages and automata theory while comparing two language generating devices.

4. Results on Computational Power

In what follows, we start describing our results for SSSN P systems with local schedules and without delays by giving theorems for both sequential strategies: based on max/min-sequentiality and on max/min-pseudosequentiality. Due to the scheduled synapses our systems are deterministic at the system level; nondeterminism is shifted to the level of neurons in both sequential strategies: strong sequential and
 pseudosequential. By simulating register machines we prove universality for both strategies.

4.1. Max-Sequential SN P Systems with Local Scheduled Synapses without Delays. In this section we provide the universality results of SSSN P systems with local schedules and without delays in the max-sequential strategy or $S_{\text{max}}^{\text{SN}} P_{\text{loc}}$ systems in short. Here the superscript max stands for max-sequential and the subscript loc means local scheduling. We note that for this section we make use of extended rules which are spiking rules where scheduling. We note that neuron $l_2$ nondeterministically selects which rule to apply. Either neuron $l_2$ or $l_3$ becomes activated depending on which rule of neuron $x_2$ is applied. We have the following two cases depending on which rule is applied by neuron $x_2$.

**Case I.** If neuron $x_2$ applies its rule $a^3 \rightarrow a$, then neuron $x_2$ fires only once, at schedule $[2, 3]$. Applying this rule consumes all 3 spikes of neuron $x_2$ and sends one spike to each of neurons $x_1$ and $x_3$. In this way, the single spike of neuron $x_1$ is restored. At the next step $x_3$ is the only neuron that can apply a rule at schedule $[3, 4]$. A spike is sent from neuron $x_3$ to neuron $l_2$ in order to begin simulating $l_2$ of $M$.

**Case II.** If neuron $x_2$ applies its rule $a^3 / a \rightarrow a$, then neuron $x_2$ fires twice. By firing first at step 3, neuron $x_2$ consumes one spike and sends one spike to each of neurons $x_1$ and $x_3$. In this way, the single spike of neuron $x_1$ is restored. At schedule $[3, 4]$ both neurons $x_2$ and $x_3$ can apply their rules, but only neuron $x_2$ can apply its rule since it has two spikes while neuron $x_3$ has only one spike. Hence at $[3, 4]$ rule $a^3 \rightarrow a$ of neuron $x_2$ is applied. At $[4, 5]$ neuron $x_4$ applies its rule to consume one spike and send one spike to each of neurons $x_3$ and $l_3$. At $[5, 6]$, neuron $x_3$ has the most spikes so it applies its forgetting rule. At $[6, 7]$ neuron $l_3$ becomes active to begin simulating $l_3$ of $M$.

The simulation of ADD is complete as the number of spikes in neuron $r$ increased by two, hence increasing the content of register $r$ by 1. Afterwards, the next instruction, either $l_2$ or $l_3$, is chosen nondeterministically for simulation. We note that neuron $r$ remains inactive as long as its content remains even. Hence, when simulating ADD neuron $r$ does not apply any of its rules. The simulation ends by restoring all spikes in all neurons to their initial configuration. Hence, the module is ready for another simulation of ADD.

**Module SUB.** In Figure 2 we give the graphical description of the module SUB simulating a rule of the form $l_1 : (\text{SUB}(r))$. The module functions are as follows: here the reference neuron is $l_1$, so it is labeled with $\bullet$, i.e., $l^*_1$. The simulation starts from the reference neuron $l^*_1$. Once $l^*_1$ applies its rule, it sends one spike to each of neurons $x_1$ and $x_2$ at schedule $[0, 1]$. In step 2, neuron $x_1$ is active with two spikes. Neuron $x_1$ applies its extended rule at schedule $[1, 2]$ to send two spikes to each of neurons $x_2$ and $r$. Neuron $r$ now has two new spikes added to its previous spikes. If the number of spikes in neuron $r$ is even then neuron $r$ does not apply any rule. It is worth noting that we use here an extended rule instead of a standard rule as it is a strong sequential case, which means at most one neuron can fire at any step. At the next step neuron $x_2$ nondeterministically selects which rule to apply.
$l_2, l_3$). The module functions are as follows: at schedule $[0, 1)$, the reference neuron $l_1'$ is activated and fires sending one spike to each of neurons $r, s_1$, and $c_{i1}$. The SUB module always has two cases depending on the value $n$ in register $r$: either $n = 0$ (when register $r$ is empty) or $n \geq 1$ (when register $r$ is nonempty). We explain both the cases separately beginning at schedule $[1, 2)$.

**Case I.** When $n = 0$ in register $r$, then neuron $r$ has $(2n + 2) + 1 = 3$ spikes. Neuron $s_1$ and $c_{i1}$ have 2 and 4 spikes, respectively. Due to max-sequentiality, $c_{i1}$ is the activated neuron so it fires consuming one spike and sending a spike at $[1, 2)$. Due to the spiking of neuron $c_{i1}$, neuron $c_{i2}$ now has 4 spikes. Neuron $c_{i2}$ is activated and it fires one spike each to $s_1$ and $c_{i2}$ at $[2, 3)$. Neurons $s_1$ and $c_{i3}$ now have 3 and 5 spikes each, respectively. Neuron $c_{i3}$ is activated so it consumes one spike and sends a spike to neuron $s_1$ at $[3, 4)$. Now neuron $s_1$ has 4 spikes and applies its forgetting rule at $[4, 5)$. Neuron $r$ is now activated so applying its second rule, it consumes one spike and sends it to neuron $s_3$ at $[5, 6)$. At $[6, 7)$ neuron $s_3$ fires sending a spike to neurons $l_1$ and $s_1$. Neurons $r$ and $s_1$ now each have their original spikes with 2 and 1 spikes, respectively. Lastly at $[7, 8)$ neuron $l_3$ is activated to begin simulating instruction $l_3$ of $M$.

**Case II.** When $n \geq 1$ in register $r$, this means neuron $r$ has $(2n + 2) + 1 \geq 5$ spikes. Neuron $r$ fires sending a spike to neuron $s_1$ at $[1, 2)$ and consuming 3 spikes. Hence, neuron $r$ now has $2n$ spikes corresponding to subtracting the content of register $r$ by 1. Neuron $c_{i1}$ has the most spikes, so it fires sending a spike to each of neurons $s_1$ and $c_{i3}$ at $[2, 3)$. At $[3, 4)$ neuron $c_{i3}$ has the most spikes so it fires sending a spike to neuron $s_1$. At $[4, 5)$ neuron $s_1$ now has 5 spikes and it fires sending a spike to neuron $s_2$. At $[5, 6)$ neuron $s_2$ fires sending one spike each of neurons $s_1$ (restoring the single spike of $s_1$) and $l_2$. Lastly at $[6, 7)$ neuron $l_2$ is activated to simulate the instruction $l_2$ of $M$.

In both scenarios the module SUB is restored to its initial configuration after simulating a SUB instruction. Since a register $r$ in $M$ can be associated with two or more SUB instructions, we must check for interference among several SUB modules. We find that there is no interference due to the semantics of local schedule. Let $l_1$ and $l_2$ be SUB instructions on register $r$, and let $l_3$ be the instruction to be simulated. Neuron $\sigma_r$ can have a synapse to neurons $s_1$ and $s_2$ in the modules associated with instructions $l_1$ and $l_2$. Due to the local schedule of synapses, only synapses associated with the simulated instruction $l_3$ are available. Neurons not associated with $l_3$ are not available; hence they do not receive any spikes from $\sigma_r$. In this way, no wrong simulations are performed by $\Pi$.

**Module FIN: Halting the Computation.** To complete the computation, the module FIN is depicted in Figure 3. Assume that register machine $M$ has halted; i.e., its instruction $l_h$ has been applied. This means that $\Pi$ simulates $l_h$ and begins to output the result. Recall that register 1 is never decremented; i.e., it is never associated with SUB instruction. This means that $\sigma_1$ holds $2n$ spikes. At $[0, 1)$ reference neuron $l_{h1}$ fires sending a spike to each of neurons $l_{h1}$ and the output neuron $\sigma_{out}$. At $[1, 2)$ neuron $l_{h1}$ fires sending a spike to neuron $\sigma_{out}$. At $[2, 3)$ neuron $\sigma_{out}$ fires sending the first spike (out of three spikes) to the environment and to neuron $\sigma_1$. At the next step neuron $\sigma_1$ is activated since it has $2n + 1 \geq 3$ spikes. For the next $n$ steps neuron $\sigma_1$ continues to apply its rule to consume 2 spikes. After $n + 1$ steps neuron $\sigma_1$ has only one spike so at this step neuron $\sigma_{out}$ fires sending a spike to the environment for the second and last time. Finally at step $n + 2$, neuron $\sigma_1$ forgets its last spike. The first and second spike of neuron $\sigma_{out}$ were sent out at step $t$ and $t + n + 1$, respectively. Hence the result of the computation of $\Pi$ is $(t + n + 1) - t - 1 = n$, exactly the value in register $r$ when register machine $M$ halts. All parameters for rule, forg, and cons are satisfied, and an extended rule is used in ADD module. This completes the proof.
In what follow, we consider max-pseudosequential strategy: a more realistic approach of spiking for neurons in the system: in case of a tie among active neurons, then all the tied neurons will fire simultaneously.

4.2. Max-Pseudosequential SN P Systems with Local Scheduled Synapses without Delays. In this section we provide the universality results of SSSN P systems with local schedules and without delays in the max-pseudosequential strategy, abbreviated as $S^{max}$SN $P_{loc}$ systems.

**Theorem 3.** $N_{2\text{,}g\text{,}m}^{S^{max}}SNP_{loc}(\text{rule}_3, \text{for}_{\text{g},4}, \text{cons}_3) = NRE$.

**Proof.** In order to prove Theorem 3, we simulate a register machine $M$ with an $S^{max}$SN $P_{loc}$ system $\Pi$. Prior to the construction of $\Pi$, we give a brief description of the computation as follows: each neuron $\sigma_i$ in $\Pi$ is associated with each register $r$ in $M$. If register $r$ stores the number $n$, then neuron $\sigma_i$ has $2n+2$ spikes. If some instruction $l_i$ is applied by $M$ this means neuron $\sigma_i$ begins simulating the instruction. In contrast to Theorem 2, in Theorem 3, due to max-pseudosequentiality we do not need extended rules; only standard rules are enough to prove universality.

**Module ADD.** The template module for the ADD instruction is depicted in Figure 4. The local reference neuron $l^*_1$ fires at schedule $[0, 1)$ sending one spike to each of neurons $x_1$, $x_2$, and $x_3$. At $[1, 2)$ there is a tie between neurons $x_1$ and $x_2$ (both have equal number of spikes), but due to max-pseudosequentiality, both neurons fire simultaneously and both send one spike each to neurons $x_3$ and $r$. In this way neuron $\sigma_r$ receives two spikes corresponding to an increment of 1 in the stored value in register $r$. At the next step neuron $x_3$ must nondeterministically decide which rule to apply, so we have the following two cases.

**Case I.** When neuron $x_3$ applies rule $a^3/aa \rightarrow a$ at $[2, 3)$, one spike is sent to neuron $x_4$. Neuron $x_4$ fires sending a spike to neuron $l_2$ at $[3, 4)$, which means system $\Pi$ will simulate the next instruction $l_2$ of $M$.

**Case II.** When neuron $x_3$ applies its rule $a^3/aa \rightarrow a$ it consumes one spike, so two spikes remain. Neuron $x_4$ receives a spike from neuron $x_3$ but cannot fire since neuron $x_3$ still has the most spikes. At $[3, 4)$ neuron $x_4$ applies $a^2 \rightarrow a$ firing and sending a spike to neuron $x_5$. Again there is a tie between neurons $x_4$ and $x_5$ at $[4, 5)$ but due to max-pseudosequentiality both fires at a single step, sending a spike but only $l_3$ receives a spike; the spike of $x_4$ is wasted as there is no synapse from it at schedule $[4, 5)$. This means system $\Pi$ is ready to simulate the next instruction $l_3$ of $M$. 

**Figure 3:** Module FIN.

**Figure 4:** Module ADD: simulating instruction $l_1 : (ADD(r) : l_2, l_3)$. 

In what follow, we consider max-pseudosequential strategy: a more realistic approach of spiking for neurons in the system: in case of a tie among active neurons, then all the tied neurons will fire simultaneously.
The simulation of an ADD instruction is correct: contents of \( \sigma \) are increased by two, followed by nondeterministically activating either neuron \( l_2 \) or \( l_3 \).

**Module SUB.** To simulate instruction \( l_1 \) : (SUB\((r)\), \( l_2\), \( l_3\)) we have the SUB module in Figure 5. At schedule \([0, 1]\) the local reference neuron \( l_1 \) fires sending one spike to each of neurons \( c_1, c_2 \), and \( s_1 \). At the next step neurons \( c_1, c_2, \) and \( s_1 \) have 1, 1, and 2 spikes, respectively. Due to max-pseudosequentiality, both tied neurons \( c_1 \) and \( c_2 \) with equal number of spikes fire simultaneously at \([1, 2]\). Neuron \( r \) receives one spike from neuron \( c_1 \) while neuron \( s_1 \) receives two, one each from neurons \( c_1 \) and \( c_2 \). At this point neuron \( \sigma_r \) has the following two cases depending on the value of \( n \) in register \( r \).

**Case I.** When \( n = 0 \), this means neuron \( \sigma_r \) has \((2n + 2) + 1 = 3\) spikes. Neuron \( s_1 \) has 4 spikes, the most at \([2, 3]\) so it applies its forgetting rule. At \([3, 4]\) neuron \( r \) applies its rule consuming and producing one spike, sending the spike to neuron \( s_2 \). In this way the spikes in \( \sigma_r \) return to \( 2n + 2 \) indicating \( n = 0 \) in register \( r \). At \([4, 5]\) neuron \( s_3 \) fires and so \( \Pi \) begins to simulate the next instruction \( l_3 \).

**Case II.** When \( n \geq 1 \), this means that neuron \( \sigma_r \) has \((2n + 2) + 1 \geq 5\) spikes. Neuron \( r \) fires at \([2, 3]\), consuming 3 spikes (to simulate the decrement of \( n \) by 1) and sending a spike to neuron \( s_1 \). At \([3, 4]\) neuron \( s_1 \) has 5 spikes and fires sending a spike to neuron \( s_2 \). At \([5, 6]\) neuron \( s_2 \) fires sending one spike each to neurons \( s_3 \) (restoring the single spike of \( s_1 \)) and \( l_2 \). Hence, system \( \Pi \) will simulate the next instruction \( l_2 \).

As due to the semantics of local reference neuron, there is no interference with any SUB module. The simulation of the SUB instruction is correct: if register \( r \) is nonempty then spikes in neuron \( \sigma_r \) are decreased by 2 followed by activating neuron \( l_2 \); if register \( r \) is empty then spikes in neuron \( \sigma_r \) are not decreased and neuron \( l_3 \) is activated. It is to be noted that with slight high complexity the SUB module in Figure 2 for max-sequentiality can also be used in \( M^{max}\text{SSN P}_{loc} \) systems.

**Module FIN: Halting the Computation.** The module FIN in Figure 3 can also be used in \( M^{max}\text{SSN P}_{loc} \) systems to produce the output of \( \Pi \).

It is clear that all three modules have utilized at most 3 rules in each neuron, consumed at most 5 spikes while forgetting at most 4 spikes. The synapses of each module are synchronized with respect to their related local reference neurons. We also note that no extended rules are needed in max-pseudosequentiality. The parameters of the theorem are satisfied, thus completing the proof.

### 4.3. Min-Sequential SN P Systems with Local Scheduled Synapses without Delays

In this section we provide the universality results of SSSN P systems with local schedules and without delays in the min-sequential strategy or \( S^{min}\text{SSN P}_{loc} \) systems in short. Here the superscript \( min \) stands for min-sequential and the subscript \( loc \) means local scheduling. We note that, due to min-sequentiality, in ADD module we do not make use of extended rules as compared to Theorem 2. Extended rule is used only in FIN module. In our results we use parameters, similar to [19] and other works on universality. Parameters \( rule_j, forg_j, cons_k \) specify at most \( i \geq 1 \) rules in each neuron, forgetting at most \( j \geq 1 \) spikes, and consuming at most \( k \geq 1 \) spikes, respectively.

**Theorem 4.** \( N_{2, gen}^{S^{min}\text{SSN P}_{loc}}(rule_3, forg_4, cons_5) = NRE \).

**Proof.** In order to prove Theorem 4, we will simulate a register machine \( M \) with an \( S^{min}\text{SSN P}_{loc} \) system \( \Pi \). Prior to construction of \( \Pi \), we give a brief description of the computation as follows: each neuron \( \sigma \) in \( \Pi \) is associated with each register \( r \) in \( M \). The local reference neuron is labeled with "*". If the content of \( r \) is the number \( n \), then spikes stored in its corresponding \( \sigma \) are \( 2n + 2 \).

![Figure 5: Module SUB: simulating instruction \( l_1 \) : (SUB\((r)\), \( l_2, l_3\)).](image-url)
Module ADD. In Figure 6 we give the graphical description of the module simulating a rule of the form $l_1: (ADD(r), l_2, l_3)$. The module functions as follows: the simulation starts from the reference neuron $l'$. Once $l'$ applies its rule, it sends one spike to each of neurons $x_1$, $x_3$, and $r$ at schedule $[0, 1]$. It is worth noting that, in order to keep neuron $r$ inactive, its content must be even; adding one spike to the content of register $r$ means that it is active to fire now, but due to min-sequentiality neuron $x_3$ will fire next at step 2. Neuron $x_1$ applies its rule at schedule $[1, 2)$ to send a spike to each of neurons $x_2$ and $r$. Neuron $r$ has now one more spike added to its previous spikes. As the number of spikes in neuron $r$ is even now so it will not apply any rule. At the next step neuron $x_2$ nondeterministically selects which rule to apply. Either neuron $l_1$ or $l_3$ becomes activated depending on which rule of neuron $x_3$ is applied. We have the following two cases depending on which rule is applied by neuron $x_2$.

Case I. If neuron $x_2$ applies its rule $a^3/a^2 \rightarrow a$, then neuron $x_2$ fires only once, at schedule $[2, 3)$. Applying this rule consumes two spikes of neuron $x_2$ and sends one spike to $x_3$. In this way, the single spike of neuron $x_2$ is restored. At the next step $x_3$ is the only neuron that can apply a rule at schedule $[3, 4)$. A spike is sent from neuron $x_3$ to neuron $l_3$ in order to begin simulating $l_3$ of $M$.

Case II. If neuron $x_2$ applies its rule $a^3/a \rightarrow a$, then neuron $x_2$ fires twice. By firing first at step 3, neuron $x_2$ consumes and sends one spike to neuron $x_3$. At schedule $[3, 4)$ both neurons $x_2$ and $x_3$ are active, but only neuron $x_2$ can apply its rule since it has two spikes while neuron $x_3$ has three spike. Hence at $[3, 4)$, rule $a^5/a \rightarrow a$ of neuron $x_2$ is applied and one spike is sent to neuron $x_3$. At $[4, 5)$ both neurons $x_3$ and $x_4$ are active, but only neuron $x_3$ with less spikes applies its rule; the spike fired by neuron $x_3$ is not received by any neuron, since neuron $x_3$ has no synapse at schedule $[4, 5)$. At $[5, 6)$, neuron $x_4$ fires sending a spike to $l_2$. At $[6, 7)$ neuron $l_2$ becomes active to begin simulating $l_2$ of $M$.

The simulation of ADD is complete as the number of spikes in neuron $r$ is increased by two, hence increasing the content of register $r$ by 1. Afterwards, the next instruction, either $l_1$ or $l_3$, is chosen nondeterministically for simulation. Since neuron $r$ remains inactive as long as its content remains even. Hence, when simulating ADD neuron $r$ does not apply any of its rules. The simulation ends by restoring all spikes in all neurons to their initial configuration. Hence, the module is ready for another simulation of ADD.

Module SUB. In Figure 7 we give the graphical description of the module simulating a rule of the form $l_1: (SUB(r), l_2, l_3)$. The module functions are as follows: At schedule $[0, 1)$, the reference neuron $l'_{1}$ is activated and fires sending one spike to each of neurons $r$, $s_1$, and $c_1$. Due to min-sequentiality, $c_1$ is the activated neuron so it fires and sends a spike at $[1, 2)$. Due to the spiking of neuron $c_1$, neurons $s_1$ and $c_2$, now have 3 and 1 spikes, respectively. Neuron $c_2$ is activated and it fires one spike to $s_1$ at $[2, 3)$.

The SUB module always has two cases depending on the value $n$ in register $r$: either $n = 0$ (when register $r$ is empty) or $n \geq 1$ (when register $r$ is nonempty). We explain both cases separately beginning at schedule $[3, 4)$. 

Case I. When $n = 0$ in register $r$, then neuron $r$ has $2n + 2 + 1 = 3$ spikes. Neuron $s_1$ has 4 spikes. Neuron $r$ is now activated with less spikes so applying its second rule, it consumes one spike and sends it to neuron $s_1$ at $[3, 4)$. At $[4, 5)$ neuron $s_1$ fires sending a spike to neuron $s_2$. At $[5, 6)$ neuron $s_2$ fires sending a spike to $s_1$ and $l_3$. Neurons $r$ and $s_1$ now each have their original spikes with 2 and 1 spikes, respectively. Lastly at $[6, 7)$ neuron $l_3$ is activated to begin simulating instruction $l_3$ of $M$.

Case II. When $n \geq 1$ in register $r$, this means neuron $r$ has $2n + 2 + 1 \geq 5$ spikes. Neuron $s_1$ has 4 spikes, the less at
Module FIN: Halting the Computation. To complete the computation, the module FIN is depicted in Figure 8. Assume that register machine $M$ has halted; i.e., its instruction $l_{2n}$ has been applied. This means that $\Pi$ simulates $l_{2n}$ and begins to output the result. At $[0,1]$ reference neuron $l^*_{2n}$ fires sending a spike to the output neuron $\sigma_{out}$. At $[1,2]$ neuron $\sigma_{out}$ fires sending the first spike (hence $t_1$) out of three spikes to the environment and to neuron $s_3$. At the next step neuron $s_3$ is activated since it has $2n + 1 \geq 3$ spikes. For the next $n$ steps neuron $s_3$ continues to apply its extended rule to consume and produce 2 spikes. At $n + 1$ step, neuron $s_3$ has only one spike so by applying second rule neuron $s_3$ spikes sending one spike to neuron $\sigma_{out}$ (hence activating it for the second time with odd number of spikes). At $n + 2$ step, neuron $\sigma_{out}$ spikes (hence $t_2$) sending a spike to the environment for the second and last time. The first and second spike of neuron $\sigma_{out}$ were sent out at step $t$ and $t + n + 2$, respectively. Hence the result of the computation of $\Pi$ is $(t + n + 2) - t - 2 = n$, exactly the value in register $r$ when register machine $M$ halts. All parameters for rule, $\text{forg}$, and $\text{cons}$ are satisfied, and an extended rule is used in FIN module. This completes the proof.

4.4. Min-Pseudosequential SN P Systems with Local Scheduled Synapses without Delays. In this section we provide the universality results of SSSN $P$ systems with local schedules and without delays in the min-pseudosequential strategy, abbreviated as $S_{\text{min}}^{\text{SSSN}} P$ loc systems.

Theorem 5. $N_{2,\text{gen}}S_{\text{min}}^{\text{SSSN}} P_{\text{loc}}(\text{rule}_3, \text{forg}_4, \text{cons}_2) = \text{NRE}$.

Proof. In order to prove the theorem we simulate a register machine $M$ with an $S_{\text{min}}^{\text{SSSN}} P_{\text{loc}}$ system $\Pi$. Prior to the construction of $\Pi$, we give a brief description of the computation as follows: each neuron $\sigma_i$ in $\Pi$ is associated with each register $r$ in $M$. If register $r$ stores the number $n$, then neuron $\sigma_i$ has $2n + 2$ spikes. If some instruction $l_i$ is applied by $M$ this means neuron $\sigma_i$ begins simulating the instruction.

Module ADD. The template module for the ADD instruction is depicted in Figure 9. The module functions are as follows: the simulation starts from the reference neuron $l^*_1$. Once $l^*_1$ applies its rule, it sends one spike to each of neurons $x_1, x_2$, and $r$ at schedule $[0,1)$. Due to min-sequentiality neuron $x_1$
will fire next at step 2. Neuron $x_1$ applies its rule at schedule [1, 2) to send a spike to each of neurons $x_2$ and $r$. Simulation of neuron $r$ is complete now with two spikes added to its content. At the next step neuron $x_2$ nondeterministically selects which rule to apply. Either neuron $l_2$ or $l_3$ becomes activated depending on rule selection of neuron $x_2$. We have the following two cases depending on the rule application by neuron $x_2$.

Case I. If neuron $x_2$ applies its rule $a^3/a^2 \rightarrow a$, then neuron $x_3$ fires only once, at schedule [2, 3). Applying this rule consumes two spikes of neuron $x_2$ and sends one spike to neuron $x_3$. In this way, the single spike of neuron $x_2$ is restored. At the next step $x_3$ is the only neuron that can apply a rule at schedule [3, 4). A spike is sent from neuron $x_3$ to neuron $l_3$ in order to begin simulating $l_3$ of $M$.

Case II. If neuron $x_2$ applies its rule $a^3/a \rightarrow a$, then neuron $x_2$ fires twice. By firing first at step 3, neuron $x_2$ consumes and sends one spike to neuron $x_3$. At schedule [3, 4) both neurons $x_2$ and $x_3$ are active, but only neuron $x_2$ can apply its rule since it has two spikes while neuron $x_3$ has three spike. Hence at [3, 4), rule $a^3/a^2 \rightarrow a$ of neuron $x_2$ is applied and one spike is sent to neuron $x_4$. At [4, 5) both neurons $x_3$ and $x_4$ are active with equal number of spikes and due to min-pseudosequentiality both fire simultaneously, the spike fired by neuron $x_3$ is not received by any neuron, since neuron $x_3$ has no synapse at schedule [4, 5) and the spike fired by $x_4$ is received by $l_2$. At [5, 6) neuron $l_2$ becomes active to begin simulating $l_2$ of $M$.

The simulation of ADD is complete as the number of spikes in neuron $r$ is increased by two, hence increasing the content of register $r$ by 1. Afterwards, the next instruction, either $l_2$ or $l_3$, is chosen nondeterministically for simulation. Since neuron $r$ remains inactive when its content remains even. Hence, when simulating ADD, neuron $r$ does not apply any of its rules. The simulation ends by restoring all spikes in all neurons to their initial configuration. Hence, the module is ready for another simulation of ADD.

Module SUB. To simulate instruction $l_1 : (SUB(r): l_2, l_3)$ we have the SUB module in Figure 10. At schedule [0, 1), the
local reference neuron $r'$ fires sending one spike to each of neurons $r, c_1, c_2,$ and $s_1$. Due to min-pseudosequentiality, both neurons $c_1$ and $c_2$ fire at $\{1,2\}$. Neuron $s_1$ receives two, one each from neurons $c_1$ and $c_2$. At this point neuron $s_1$ has the following two cases depending on the value of $n$ in register $r$.

Case I. When $n = 0$, this means neuron $s_1$ has $(2n + 2) + 1 = 3$ spikes. Neuron $s_1$ has 4 spikes, the most at $\{2,3\}$, so neuron $r$ fires and sends a spike to neuron $s_1$. At $\{3,4\}$ neuron $s_1$ has 5 spikes and fires sending a spike to neuron $s_2$. At $\{4,5\}$ neuron $s_2$ fires sending a spike to each of neurons $s_1$ (returning the initial spike of $s_1$) and $l_3$. Hence, system $\Pi$ will simulate the next instruction $l_3$.

Case II. When $n \geq 1$, this means that neuron $s_1$ has $(2n + 2) + 1 \geq 5$ spikes. Neuron $s_1$ has 4 spikes, the less at $\{2,3\}$, so neuron $s_1$ applies it forgetting rule at $\{2,3\}$. Neuron $r$ fires at $\{3,4\}$, consuming 3 spikes (to simulate the decrement of $n$ by 1) and sending a spike to neuron $s_2$. At $\{4,5\}$ neuron $s_2$ fires sending a spike to each of neurons $s_1$ (returning the initial spike of $s_1$) and $l_2$. Hence, system $\Pi$ will simulate the next instruction $l_2$.

Due to local reference neuron, there is no interference with any SUB module. The simulation of the SUB instruction is correct: if register $r$ is nonempty then spikes in neuron $s_1$ are decreased by 2 followed by activating neuron $l_2$; if register $r$ is empty then spikes in neuron $s_1$ are not decreased and neuron $l_3$ is activated. It is to be noted that the SUB module in Figure 7 for min-sequentiality can also be used in $S^{\min}_\text{SSN}$ P$_{ax}$ systems.

Module FIN: Halting the Computation. The module FIN in Figure 8 can also be used in $S^{\min}_\text{SSN}$ P$_{ax}$ systems to produce the output of $\Pi$.

It is clear that all three modules have utilized at most 3 rules in each neuron, consumed at most 5 spikes while forgetting at most 4 spikes. The synapses of each module are synchronized with respect to their related local reference neurons. We also note that no extended rules are needed in ADD module due to min-sequentiality, but in FIN module. The parameters of the theorem are satisfied, thus completing the proof.

5. Final Remarks

In this work, the computational power of sequential SN P systems with local scheduled synapses without delay is investigated. Results show that sequential SN P systems with local scheduled synapses without delay working in both max/min-sequentiality and max/min-pseudosequentiality strategies are computationally universal with both standard and extended rules (only in case when standard rules fail to perform). In particular, we showed for both strategies that universality is achieved using at most 3 rules per neuron, consuming, and forgetting at most 5 and 4 spikes, respectively. We further note that extended rule is used only in ADD module with the max-sequential strategy and FIN module with both min-sequential and min-pseudosequential strategies. Moreover strong sequential strategy has slight higher complexity than pseudosequential.

Our future work is to prove universality of our variants with global-scheduled synapses. The open problems are to reduce complexity of our systems, to prove universality without forgetting rules and extended rules.

The other direction that might be interesting is to realize which class of problems/languages these kind of SN P systems variant is capable of solving/deciding, thereby comparing its capability in recognizing/solving languages/problems with the other variants with respect to several parameters/ingredients of the SN P systems.

Data Availability

The paper contains only theoretical derivations; no data was used in the research.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by National Natural Science Foundation of China (61502186) and China Postdoctoral Science Foundation (2016MS92335).

References


