Spiking Neural P Systems

Wang, J.

Chapter 2

Limited Asynchronous Spiking Neural P Systems

Abstract
In a biological system, if a long enough time interval is given, an enabled chemical reaction will finish its reaction in the given time interval. With this motivation, it is natural to impose a bound on the time interval when an enabled spiking rule in a spiking neural P system (SN P system, for short) remains unused. In this work, a new working mode of SN P systems is defined, which is called limited asynchronous mode. In an SN P system working in limited asynchronous mode, if a rule is enabled at some step, this rule is not obligatorily used. From this step on, if the unused rule may be used later, it should be used in the given time interval. If further spikes make the rule non-applicable, then the computation continues in the new circumstances. The computation result of a computation in an SN P system working in limited asynchronous mode is defined as the total number of spikes sent into the environment by the system. It is proved that limited asynchronous SN P systems with standard spiking rules are universal. If the number of spikes present in each neuron of a limited asynchronous SN P system with standard spiking rules is bounded during a computation, then the power of a limited asynchronous SN P system with standard spiking rules falls drastically, and we get a characterization of semilinear sets of numbers.

2.1 Introduction

Spiking neural P systems (SN P systems, for short) are a class of distributed and parallel computation models inspired by the way neurons communicate by means of electrical impulses of identical shape (called spikes). SN P systems were introduced in [16], and then investigated in a large number of papers. Readers can refer to [35] for general information in this area, and to the membrane computing

website from [1] for the up-to-date information.

Briefly, an SN P system consists of a set of neurons placed in the nodes of a directed graph, where neurons send signals (which are called spikes, denoted by the symbol a in what follows) along synapses (arcs of the graph). Spikes evolve by means of standard spiking rules, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1$, $d \geq 0$. In other words, if a neuron contains k spikes such that $a^k \in L(E)$, $k \geq c$, then it can consume c spikes and produce one spike after a delay of d steps. This spike is sent to all neurons connected by an outgoing synapse from the neuron where the rule was applied. There are also standard forgetting rules, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are forgotten if the neuron contains exactly s spikes. Moreover, an extension of this type of rules was considered in [6], allowing more than one spike to be generated by the rule.

An SN P system works in a synchronized manner. A global clock is assumed, and in each time unit, the rule to be applied in each neuron is nondeterministically chosen, a chosen rule must be applied for each neuron with applicable rules, and the work of the system is sequential in each neuron: only (at most) one rule is applied in each neuron. One of the neurons is considered to be the output neuron, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, and the other moments are marked with 0. This binary sequence is called the spike train of the system; it might be infinite if the computation does not stop. Various numbers can be associated with a spike train, which can be considered as computed (or generated) by an SN P system.

Synchronized SN P systems using standard rules were proved to be computationally complete both in the generating and the accepting case [16]. In the proof of these results, the synchronization plays a crucial role. However, both from a mathematical point of view and from a neuro-biological point of view, it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory. Even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used. The neuron may remain unfired, maybe receiving spikes from the neighboring neurons. If the unused rule may be used later, it is used later, without any restriction on the interval when it has remained unused. If further spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now). With such motivation, asynchronous SN P systems were introduced in [4], and it is proved that asynchronous SN P systems with extended rules are equivalent with Turing machines. However, it remains open whether asynchronous SN P systems with standard rules are universal.

In the definition of asynchronous SN P systems from [4], there is no restriction on the time interval in which an enabled spiking rule remains unused. However, in a biological system, if a long enough time interval is given, an enabled chemical reaction will finish its reaction within the given time interval. So, it is natural to impose a bound on the time interval in which a spiking rule remains unused. Such

variant of asynchronous mode is called limited asynchronous mode. In an SN P system working in limited asynchronous mode, if a rule is enabled at some step, this rule is not obligatorily used in the same step. However, if from this moment on the rule remains applicable, it should be used in the given time interval. If further spikes arriving in the neuron make the rule non-applicable, then the computation continues in the new circumstances. The computation result of a computation in an SN P system working in limited asynchronous mode is defined as the total number of spikes sent into the environment by the system. In this work, we prove that limited asynchronous SN P systems with standard spiking rules are universal.

In a general asynchronous SN P system, since there is no restriction on the time interval when an enabled spiking rule remains unused, the feature of delay is not very helpful and was not used in the universality result of [4]. However, in a limited asynchronous SN P system, the feature of delay adds functionality. Thus, in the proof of the universality result in this work, the feature of delay is used and plays a crucial role, which shows some "programming capacity". A general asynchronous SN P system with standard rules loses "programming capacity" from both extended rules and the feature of delay. So, our research gives some hint to support the conjecture that a general asynchronous SN P system with standard rules is non-universal [4].

2.2 Prerequisites

Readers can refer to [38] for basic language and automata theory, as well as to [30] for basic membrane computing. We here only introduce some necessary notations and notions.

For an alphabet V, V^* denotes the set of all finite strings over V, with the empty string denoted by λ. The set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$.

Regular expressions are built starting from λ and single symbols using the operators union ($\cup$), concatenation ($\cdot$) and star ($*$), where non-necessary parentheses are omitted. The language represented by expression E is denoted by $L(E)$, where $L(\lambda) = \varnothing$.

By NRE we denote the families of Turing computable sets of numbers. (NRE is the family of length sets of recursively enumerable languages.)

A register machine is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers (each holds a natural number), H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction HALT), and I is the set of instructions. Each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),

- l_i : (SUB(r), l_j, l_k) (if register r is non-zero, then subtract 1 from it, and go to the instruction with label l_j; otherwise, go to the instruction with label l_k),
- l_h : HALT (the halt instruction).

A register machine M computes (generates) a number n in the following way. The register machine starts with all registers empty (i.e., storing the number zero). It applies the instruction with label l_0 and proceeds to apply instructions as indicated by labels (and, in the case of SUB instructions, by the content of registers). If the register machine reaches the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE [27].

Without loss of generality, it can be assumed that l_0 labels an ADD instruction and that in the halting configuration all registers different from the first one are empty, and that the output register is never decremented during the computation (its content is only added to).

We use the following convention. When the power of two number generating/accepting devices D_1 and D_2 are compared, number zero is ignored; that is, $N(D_1) = N(D_2)$ if and only if $N(D_1) - \{0\} = N(D_2) - \{0\}$ (this corresponds to the usual practice of ignoring the empty string in language and automata theory).

2.3 Limited Asynchronous Spiking Neural P Systems

In this section, we recall the definition of spiking neural P systems, and introduce a new working mode of spiking neural P systems, which is called limited asynchronous mode.

A *spiking neural P system* (an SN P system, for short), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out), \text{ where:}$$

- $O = \{a\}$ is the singleton alphabet (a is called *spike*);
- $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where:
  a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
  b) R_i is a finite set of *rules* of the following two forms:
    (1) $E/a^c \rightarrow a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$;
    (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, $a^s \notin L(E)$;

- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
- $out \in \{1, 2, \ldots, m\}$ indicates the *output* neuron.

The rules of type (1) are *standard firing rules* (we also say *standard spiking rules*), and they are applied as follows. If neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a; d$ can be applied. The application of this rule means that c spikes are consumed (removed) (thus only $k - c$ spikes remain in σ_i), neuron σ_i is fired, which produces a spike after d time units (as usual in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately; if $d = 1$, then the spike is emitted in the next step, etc.. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ neuron σ_i is *closed* (this corresponds to the refractory period from neurobiology), so that it cannot receive further spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then the spike is lost). In step $t + d$, neuron σ_i spikes and becomes open again, so that it can receive spikes (the received spikes can be used in step $t + d + 1$). If a rule $E/a^c \to a; d$ has $E = a^c$, then we will write it in the simplified form $a^c \to a; d$.

The rules of type (2) are *forgetting* rules. They are applied as follows. If neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, which means that all s spikes are removed from σ_i.

Extended rules were considered in [6] to obtain universal systems. In our paper we only consider standard rules, but we recall extended rules to compare the notions. Extended rules are of the form $E/a^c \to a^p; d$, and used in the following way. When a rule $E/a^c \to a^p; d$ is used, c spikes are consumed and p spikes are produced after a delay of d steps. A rule with $p \geq 1$ is called an *extended firing rule*. A rule with $p = d = 0$ is written in the form $E/a^c \to \lambda$, which is called an *extended forgetting rule*.

In the synchronized mode, in each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i should be used. Since two firing rules, $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$, can have $L(E_1) \cap L(E_2) \neq \varnothing$, it is possible that two or more rules can be applied in a neuron, and in this case, only one of them is chosen non-deterministically. Note that the neurons work in parallel (synchronously), but each neuron sequentially processes its spikes, using only one rule in each time unit.

In this work, we consider a new working mode of SN P systems, which is called limited asynchronous mode. In an SN P system working in limited asynchronous mode, a single upper bound b ($b \geq 2$) on time intervals is given, valid for all rules. If a rule in neuron σ_i is enabled at step t and neuron σ_i receives no spike from step t to step $t + b - 2$, then this rule can and must be applied at a step in the next time interval b (that is, at a non-deterministically chosen step from t to $t + b - 1$). If the enabled rule in neuron σ_i is not applied, and neuron σ_i receives new spikes, making the rule non-applicable, then the computation continues in

the new circumstance (maybe other rules are enabled now).

It is necessary to point out that (1) when a neuron spikes, then after a delay of d steps ($d \geq 0$), its spikes immediately leave the neuron, and reach the target neurons simultaneously (that is, there is no time needed for passing along a synapse from one neuron to another neuron); (2) as in the synchronous mode, if a rule is applied at step t and $d \geq 1$, then at steps $t, t+1, t+2, \ldots, t+d-1$ neuron σ_i is closed (so it cannot receive further spikes), and it becomes again open at step $t+d$.

A configuration of the system is described by the number of spikes present in each neuron and the open-closed states of neurons as well as the time that has elapsed for each rule since it became applicable. The initial configuration is defined by the number of initial spikes $n_1, \ldots, n_m$ with all neurons being open (as no rule was used before). Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting from the initial configuration is called a *computation*. A computation is considered as successful when it reaches a configuration where all neurons are open and no rule can be used.

Because in a limited asynchronous SN P system, an enabled rule can be applied at any moment in the next time interval b, in the spike train the number of occurrences of 0 between two occurrences of 1 can have a variation from 0 to b. Hence the result of a computation can no longer be defined in terms of the steps between two consecutive spikes as in the synchronized mode. Therefore, in this work, the result of a computation is defined as the total number of spikes sent into the environment by the output neuron. Specifically, if there is a successful computation of a limited asynchronous SN P system where the output neuron sends out exactly n spikes, then the system generates a number n. Equivalently, the result of a computation can also be the number of spikes present in a specified neuron in the halting configuration: consider an additional neuron, which receives the spikes emitted by the previous output neuron and has no rule inside. When the computation halts, the content of this neuron is the result of the computation.

Successful computations that send no spike out can be considered as generating number zero, but in this work, we adopt the convention to ignore number zero when the computation power of two devices is compared.

We denote by $N_{gen}^{lasyn}(\Pi)$ the set of numbers generated in the limited asynchronous way by an SN P system Π. By $N_{gen}^{lasyn}SNP$ we denote the family of such sets of numbers generated by limited asynchronous systems with standard rules.

In what follows, we only consider limited asynchronous SN P systems with standard rules. Because there is no confusion, limited asynchronous SN P systems with standard rules are often simply called limited asynchronous SN P systems.

As usual, SN P systems are represented graphically, which may be easier to understand than in a symbolic way. We use an oval containing the spiking rules and (if present) the number of spikes (in the form a^n for n spikes present in a neuron) inside to represent a neuron, and a directed graph to represent the structure of an SN P system: the neurons are placed in the nodes of the graph

and the edges represent the synapses. The output neuron has an outgoing arrow, suggesting its communication with the environment. For simplicity, neuron σ_i in the picture is labelled by i.

We here point out that the asynchronous mode given in this work is not a "true" asynchronous mode in the following sense, and hence the word "limited" is used. (1) SN P systems considered in this work have a global clock to mark the time for the whole system instead of clock freeness. (2) The fixed time bound for the execution intervals actually gives the possibility to predict some timing relatives. For example, in the system given in Figure 2.1, the spike passing along the path $\sigma_s\sigma_1 \ldots \sigma_{b+1}\sigma_t$ arrives at least one step later than the one passing along the path $\sigma_s\sigma_{b+2}\sigma_t$. However, in a true asynchronous mode, it would be not possible to predict such timing relations.

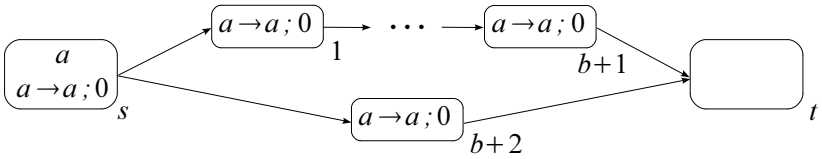

Figure 2.1: A limited asynchronous SN P system with time bound b.

2.4 An Example

In this section, we give an example to clarify the definition of limited asynchronous mode.

Example 1. Consider the SN P system Π shown in Figure 2.2, which consists of four neurons. In the initial configuration, all neurons are empty except that the output neuron σ_{out} has one spike. The number $b \geq 2$ is arbitrary.

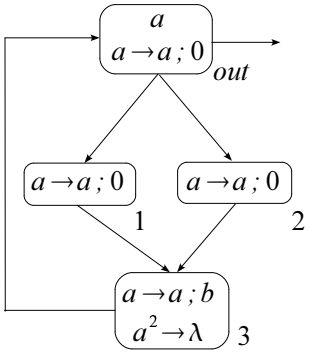

Figure 2.2: SN P system Π.

First, we consider that system Π works in the synchronous mode. At step 1, neuron σ_{out} fires, sending one spike to neurons σ_1, σ_2, and the environment, respectively. At step 2, with one spike inside, each of neurons σ_1 and σ_2 fires, sending a spike to neuron σ_3. At step 3, neuron σ_3 removes its spikes by rule $a^2 \to \lambda$. From now on, there is no spike in each neuron of system Π, and each neuron in system Π is open, so the computation halts. During the computation, system Π sends only one spike into the environment. The set of numbers generated by system Π in the synchronous mode is $\{1\}$.

Now, we consider that system Π works in the limited asynchronous mode, where the time bound associated with all rules is b. With one spike in neuron σ_{out}, rule $a \to a; 0$ is enabled at step 1. In the next b steps, neuron σ_{out} cannot receive any spike from other neurons. So rule $a \to a; 0$ can and must be applied at one of steps $1, 2, \ldots, b$. We assume that rule $a \to a; 0$ in neuron σ_{out} is applied at step t_1 ($1 \leq t_1 \leq b$), sending a spike to neurons σ_1, σ_2 and the environment, respectively, then each of neurons σ_1 and σ_2 will fire by rule $a \to a; 0$ at one of steps $t_1 + 1, t_1 + 2, \ldots, t_1 + b$.

If neurons σ_1 and σ_2 fire at a same step, then neuron σ_3 receives two spikes. With two spikes inside, rule $a^2 \to \lambda$ in neuron σ_3 is enabled, and these two spikes are removed at one of the next b steps. In this way, system Π contains no spike and all neurons are open, so the computation halts.

If neurons σ_1 and σ_2 fire at different steps, then we assume that neuron σ_3 receives the first spike at step t_2 and the second one at step t_3, where $t_1 + 1 \leq t_2 < t_3 \leq t_1 + b$. Rule $a \to a; b$ in neuron σ_3 is enabled and free to be applied at one of steps from t_2 to t_3 (more precisely, it can be applied before neuron σ_3 receives the second spike; after neuron σ_3 receives the second spike, the content of neuron σ_3 changes and rule $a \to a; b$ cannot be applied). We consider the following two cases.

- If rule $a \to a; b$ is applied before step t_3 (we assume that the moment is step t_4, $t_2 \leq t_4 \leq t_3$), then neuron σ_3 sends a spike to neuron σ_{out} at step $t_4 + b$, and it is closed at steps from t_4 to $t_4 + b - 1$ because of the delay of rule $a \to a; b$. Especially, neuron σ_3 is closed at step t_3, since $t_4 \leq t_3 < t_4 + b$. So, neuron σ_3 cannot receive the second spike sent out from one of neurons σ_1 and σ_2 at step t_3. After step $t_4 + b$, all neurons in system Π have no spike except that the output neuron σ_{out} has one spike, which is a same configuration with the initial one. In this way, the computation continues.

- If rule $a \to a; b$ is not applied before step t_3, then neuron σ_3 is open and receives the second spike at step t_3. In this way, the content of neuron σ_3 is changed, rule $a \to a; b$ cannot be applied, and rule $a^2 \to \lambda$ is enabled. These two spikes in neuron σ_3 are removed at one of the next b steps, and the computation halts.

In general, the computation in system Π halts if neurons σ_3 receives two spikes at a same step or accumulates two spikes that are received at different steps; otherwise, system Π reaches a same configuration as the initial one (especially, in this case,

system Π will send a spike to the environment). Therefore, $N_{gen}^{lasyn}(\Pi) = \mathbb{N} \setminus \{0\}$, where $\mathbb{N}$ is the set of natural numbers.

2.5 Universality of Limited Asynchronous SN P Systems

We prove that limited asynchronous SN P systems with standard spiking rules are Turing universal by simulating a register machine. Although not explicitly obvious from the statement of the result, the characterization is valid for every fixed time bound of SN P systems.

Theorem 1
$N_{gen}^{lasyn}SNP = NRE$.

Proof
We show that $NRE \subseteq N_{gen}^{lasyn}SNP$; the converse inclusion is straightforward but cumbersome (for similar technical details, please refer to Section 8.1 in [30]).

Let us consider a register machine $M = (m, H, l_0, l_h, I)$ with the properties specified in Section 2.2: the result of a computation is the number from register 1, and this register is never decremented during the computation. In what follows, a specific limited asynchronous SN P system with standard spiking rules Π will be constructed to simulate the register machine M, where a time bound b is associated with all spiking rules.

System Π is composed of some modules. These modules are interconnected by shared neurons. We present these modules graphically instead of symbolic way. By the structure of these modules, we can easily see how these modules are interconnected by shared neurons to form the whole system Π. So, we omit the formal description of system Π, and focus on the construction of modules and the explanation of these modules.

We construct modules ADD and SUB to simulate the instructions of M, as well as an output module FIN to output computation results. Each register r of M will have a neuron σ_r in Π, and if the register contains the number n, then the associated neuron will have $2n$ spikes. The rules in neuron σ_1 differ in those for the other registers — neurons σ_r ($r \geq 2$). The reason is that σ_1 must additionally interact with the output module FIN (and the number stored in register 1 is never decremented). Neuron σ_1 contains a rule $a(a^2)^+/a^3 \to a; b$; while neuron σ_r ($r \geq 2$) contains rules $a(a^2)^+/a^3 \to a; b$ and $a \to a; 9b$. A neuron σ_{l_i} is associated with each label $l_i \in H$, and some auxiliary neurons $\sigma_{l_i^{(j)}}$, $j = 1, 2, 3, \ldots$, will also be considered (remember that each $l_i \in H$ is associated with a unique instruction of M, hence all neurons $\sigma_{l_i}, \sigma_{l_i^{(j)}}$ are precisely associated with a unique instruction of M).

In the initial configuration, all neurons are empty except that neuron σ_{l_0} associated with label l_0 of M has one spike inside. In general, when a neuron σ_{l_i},

$l_i \in H$, has one spike inside, then neuron σ_{l_i} becomes active and the module associated with instruction l_i starts to work, simulating the instruction l_i.

The initial instruction of M, the one with label l_0, is an ADD instruction.

Module ADD: Simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

Module ADD is designed for simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$. Although this module is valid for each register r, in our arguments we have to distinguish between $r = 1$ and $r \geq 2$ as register 1 has been implemented differently from the other registers. This is a consequence of the property of M: the number stored in register 1 is not decremented during a computation and the result of a computation is stored in that register (more precisely, in system Π, neuron σ_1 is related to module FIN that takes care of outputting computation results). The ADD module for ADD instructions that act on register r ($r \geq 2$) is shown in Figure 2.3, which consists of two parts. The first part contains the "initial" neuron σ_{l_i} together with 8 auxiliary neurons responsible for updating the contents of neuron σ_r for register r. The second part consists of six additional neurons and will send a spike to exactly one of the consecutive "initial" neurons $\sigma_{l_j}, \sigma_{l_k}$ for the instructions with labels l_j, l_k. The ADD module for an ADD instruction $l_i : (\texttt{ADD}(1), l_j, l_k)$ is the same with the module shown in Figure 2.3 except that σ_1 has only a rule $a(a^2)^+/a^3 \to a; b$.

In what follows, we first check the work of module ADD assuming that $r \geq 2$. After we have shown correctness in this case, we add some remarks for the special register $r = 1$. Due to the inherent nondeterminism in the application of the rules our argumentation involves a tedious case-by-case analysis.

Let us assume that M is at a step when we have to simulate an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$, with one spike present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spike in any other neurons except those neurons associated with the registers. Having one spike in neuron σ_{l_i}, rule $a \to a; 0$ is enabled. At one of the next b steps (assume it is at step t_1), neuron σ_{l_i} fires, sending a spike to neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$, respectively. With one spike inside, rule $a \to a; 0$ in each of neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ is enabled. Each of these neurons will fire at one of the next b steps (that is, at a step from $t_1 + 1$ to $t_1 + b$), sending a spike to neurons σ_r, $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(8)}}$, respectively. In what follows, we check the work of neurons σ_r, $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(8)}}$.

For neuron $\sigma_{l_i^{(5)}}$, we consider the following three possible cases (according to the time when neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ fire).

(1) If neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ fire at a same step, then neuron $\sigma_{l_i^{(5)}}$ receives 3 spikes at the same step. Neuron $\sigma_{l_i^{(5)}}$ will wait since no rule in neuron $\sigma_{l_i^{(5)}}$ is enabled.

(2) If any two of neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ fire at a same step while the other one fires at a different step, then neuron $\sigma_{l_i^{(5)}}$ will have two possible

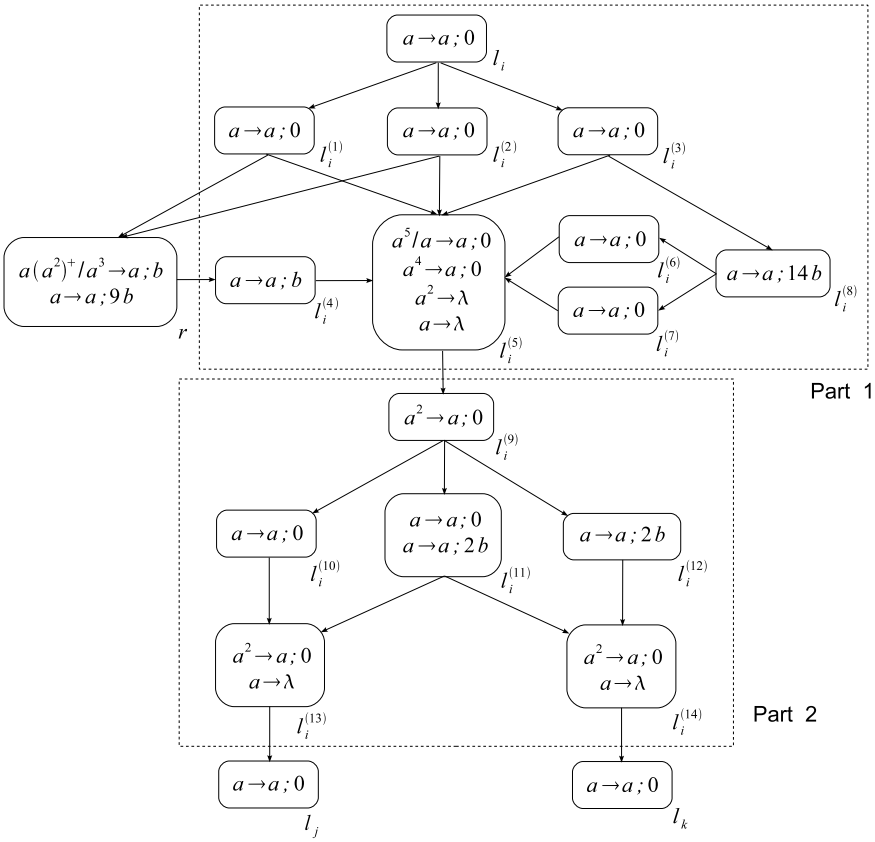

Figure 2.3: Module ADD for simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$, where $r \geq 2$.

situations, no matter whether the number of the first received spikes is two or one.

(a) If the number of first received spikes is two (resp. one) and the enabled rule $a^2 \to \lambda$ (resp. $a \to \lambda$) fires before the later spike (or spikes) from neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ arrives (or arrive) in neuron $\sigma_{l_i^{(5)}}$, then the later received spike (or spikes) is (or are) removed by rule $a \to \lambda$ (resp. $a^2 \to \lambda$). So neuron $\sigma_{l_i^{(5)}}$ has 0 spike inside.

(b) If neuron $\sigma_{l_i^{(5)}}$ does not forget the first received spike (or spikes) before the later spikes (or spike) arrive in it, then neuron $\sigma_{l_i^{(5)}}$ will accumulate 3 spikes and no rule is enabled in it.

(3) If each of neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, $\sigma_{l_i^{(3)}}$ fires at different steps, then neuron $\sigma_{l_i^{(5)}}$ will have the following three possible situations.

(a) If neuron $\sigma_{l_i^{(5)}}$ forgets the first received spike by rule $a \to \lambda$ before the second spike arrives, and the second received spike is also removed before the third one arrives, then the third spike will also be removed by rule $a \to \lambda$. So neuron $\sigma_{l_i^{(5)}}$ has 0 spike inside.

(b) If neuron $\sigma_{l_i^{(5)}}$ does not forget the first received spike, but both the first and second received spikes are removed by rule $a^2 \to \lambda$ before the third spike arrives, then the third received spike is also removed by rule $a \to \lambda$. So neuron $\sigma_{l_i^{(5)}}$ has 0 spike inside.

(c) If neuron $\sigma_{l_i^{(5)}}$ does not forget any spike received from neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$, then it has 3 spikes inside.

Note that computations in all the above cases start from step $t_1 + 1$, and end not more than step $t_1 + 2b$. Because of delays, the possible spikes from neurons $\sigma_{l_i^{(6)}}$ (along the path $\sigma_{l_i^{(3)}}\sigma_{l_i^{(8)}}\sigma_{l_i^{(6)}}\sigma_{l_i^{(5)}}$), $\sigma_{l_i^{(7)}}$ (along the path $\sigma_{l_i^{(3)}}\sigma_{l_i^{(8)}}\sigma_{l_i^{(7)}}\sigma_{l_i^{(5)}}$), $\sigma_{l_i^{(4)}}$ (along the paths $\sigma_{l_i^{(1)}}\sigma_r\sigma_{l_i^{(4)}}\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(2)}}\sigma_r\sigma_{l_i^{(4)}}\sigma_{l_i^{(5)}}$) arrive in neuron $\sigma_{l_i^{(5)}}$ as not earlier than step $t_1 + 2b + 1$. That is, neuron $\sigma_{l_i^{(5)}}$ receives spikes only from neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ from step $t_1 + 1$ to step $t_1 + 2b$. Therefore, neuron $\sigma_{l_i^{(5)}}$ has 0 or 3 spikes inside at step $t_1 + 2b$. (As we will see below, for the case neuron $\sigma_{l_i^{(5)}}$ has 0 spike inside at step $t_1 + 2b$, all the related computations go to a "wrong" simulation. System Π will abort and send no spike into the environment. In this way, all computation results of Π are results of correct simulations of M in the sense that the number 0 is ignored when the power of two computation devices are compared.)

For neuron σ_r, we consider the following three possible cases.

(1) If neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at a same step, then the number of spikes in neuron σ_r increases by two (corresponding to that the number stored in

register r of M is increased by one) and no rule in neuron σ_r is enabled. Note that the possible spikes from neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ arrive in neuron $\sigma_{l_i^{(5)}}$ after step $t_1 + 14b + 1$; and neuron $\sigma_{l_i^{(5)}}$ has 0 or 3 spikes inside at step $t_1 + 2b$. So neuron $\sigma_{l_i^{(5)}}$ has 0 or 3 spikes inside at step $t_1 + 14b$.

(2) If neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at different steps (note that the distance between these two steps is less than $b-1$), and neuron σ_r applies its enabled rule after it receives the first spike, then neuron σ_r will be closed for at least b steps because of the delay b and $9b$ in rules $a(a^2)^+/a^3 \to a; b$ and $a \to a; 9b$. So the second spike from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ is lost. In this case, the number of spikes in neuron σ_r is not increased, which is a "wrong" simulation (as we will see below, the simulation in system Π aborts and outputs no spike into the environment).
The spike sent out from neuron σ_r moves to neuron $\sigma_{l_i^{(4)}}$, and then to neuron $\sigma_{l_i^{(5)}}$, where this spike arrives in neuron $\sigma_{l_i^{(5)}}$ at one of steps from $t_1 + 2b$ to $t_1 + 13b$ because of the time bound and the delay associated with rules. Note that the possible spikes from neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ arrive in neuron $\sigma_{l_i^{(5)}}$ after step $t_1 + 14b + 1$; and neuron $\sigma_{l_i^{(5)}}$ has 0 or 3 spikes inside at step $t_1 + 2b$. So neuron $\sigma_{l_i^{(5)}}$ has 1 or 4 spikes after it receives a spike from neuron $\sigma_{l_i^{(4)}}$; at one of the next b steps, rule $a^4 \to a; 0$ or $a \to \lambda$ is applied (so, it is possible that neuron $\sigma_{l_i^{(9)}}$ receives a spike from $\sigma_{l_i^{(5)}}$; as we will see below, after that, neuron $\sigma_{l_i^{(9)}}$ will receive no spike anymore; so rule $a^2 \to a; 0$ in neuron $\sigma_{l_i^{(9)}}$ will not be enabled, and the computation will abort). Therefore, neuron $\sigma_{l_i^{(5)}}$ has 0 spikes inside at step $t_1 + 14b$.

(3) If neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at different steps, but the enabled rule in neuron σ_r is not applied before the second spike from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ arrives in neuron σ_r, then the number of spikes in neuron σ_r is increased by two and no rule in neuron σ_r is enabled. Note that the possible spikes from neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ arrive in neuron $\sigma_{l_i^{(5)}}$ after step $t_1 + 14b + 1$; and neuron $\sigma_{l_i^{(5)}}$ has 0 or 3 spikes inside at step $t_1 + 2b$. So neuron $\sigma_{l_i^{(5)}}$ has 0 or 3 spikes inside at step $t_1 + 14b$.

Therefore, in all computations in the above cases, neuron $\sigma_{l_i^{(5)}}$ has 0 or 3 spikes inside at step $t_1 + 14b$.

According to the number of spikes in neuron $\sigma_{l_i^{(5)}}$ at step $t_1 + 14b$ and the time when neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ fire, we consider the following three cases.

(1) If neuron $\sigma_{l_i^{(5)}}$ has 0 spike at step $t_1 + 14b$, then no matter when neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ fire, neuron $\sigma_{l_i^{(5)}}$ always removes the spikes received from neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ by rules $a \to \lambda$ and $a^2 \to \lambda$. In this case, computations in system Π abort and no spike is sent to the environment.

(2) If neuron $\sigma_{l_i^{(5)}}$ has 3 spikes at step t_1+14b and neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ fire at a same step, then neuron $\sigma_{l_i^{(5)}}$ has 5 spikes. Rule $a^5/a \to a;0$ in neuron $\sigma_{l_i^{(5)}}$ is enabled and applied, sending a spike to neuron $\sigma_{l_i^{(9)}}$; then rule $a^4 \to a;0$ in neuron $\sigma_{l_i^{(5)}}$ is enabled and applied, sending another spike to neuron $\sigma_{l_i^{(9)}}$. With two spikes inside, rule $a^2 \to a;0$ in neuron $\sigma_{l_i^{(9)}}$ is enabled, and the computation continues.

(3) If neuron $\sigma_{l_i^{(5)}}$ has 3 spikes at step t_1+14b and neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(7)}}$ fire at different steps, then there are the following two possible situations.

(a) If neuron $\sigma_{l_i^{(5)}}$ receives the first spike from neurons $\sigma_{l_i^{(6)}}$, $\sigma_{l_i^{(7)}}$, and neuron $\sigma_{l_i^{(5)}}$ consumes 4 spikes by rule $a^4 \to a;0$ before the second spike from neurons $\sigma_{l_i^{(6)}}$, $\sigma_{l_i^{(7)}}$ arrives in neuron $\sigma_{l_i^{(5)}}$, then neuron $\sigma_{l_i^{(5)}}$ will remove the second spike by rule $a \to \lambda$. In this case, neuron $\sigma_{l_i^{(9)}}$ receives only one spike from neuron $\sigma_{l_i^{(5)}}$, its rule is not enabled and the computation aborts.

(b) If neuron $\sigma_{l_i^{(5)}}$ receives the first spike from neurons $\sigma_{l_i^{(6)}}$, $\sigma_{l_i^{(7)}}$, and the enabled rule $a^4 \to a;0$ is not applied before the second spike from neurons $\sigma_{l_i^{(6)}}$, $\sigma_{l_i^{(7)}}$ arrives in neuron $\sigma_{l_i^{(5)}}$, then neuron $\sigma_{l_i^{(5)}}$ accumulates 5 spikes. First, rule $a^5/a \to a;0$ in neuron $\sigma_{l_i^{(5)}}$ is applied, sending a spike to neuron $\sigma_{l_i^{(9)}}$; then rule $a^4 \to a;0$ in neuron $\sigma_{l_i^{(5)}}$ is applied, sending another spike to neuron $\sigma_{l_i^{(9)}}$. With two spikes inside, rule $a^2 \to a;0$ in neuron $\sigma_{l_i^{(9)}}$ is enabled, and the computation continues.

Therefore, neuron $\sigma_{l_i^{(9)}}$ receives two spikes from neuron $\sigma_{l_i^{(5)}}$ only if neuron σ_r does not fire (it is the case the number of spikes in neuron σ_r is increased by two) and neuron $\sigma_{l_i^{(5)}}$ accumulates 5 spikes (then neuron $\sigma_{l_i^{(5)}}$ fires for two times by rules $a^5/a \to a;0$ and $a^4 \to a;0$).

Hence, we have shown that Part 1 of module ADD ensures that the operation ADD of M (the number stored in register r is increased by one) is correctly simulated. Now we will turn to Part 2 of module ADD which ensures the correctly simulation in that instruction l_j or l_k is non-deterministically chosen.

Assume that neuron $\sigma_{l_i^{(9)}}$ fires at step t_2 (that is, neurons $\sigma_{l_i^{(10)}}$, $\sigma_{l_i^{(11)}}$, $\sigma_{l_i^{(12)}}$ receive a spike from neuron $\sigma_{l_i^{(9)}}$ at step t_2). At step t_2+1, both of rules $a \to a;0$ and $a \to a;2b$ in neuron $\sigma_{l_i^{(11)}}$ are enabled; at one of the next b steps (from step t_2+1 to step t_2+b), one of them is non-deterministically chosen and applied.

(1) If rule $a \to a;0$ is applied at one of steps from t_2+1 to t_2+b, then neurons $\sigma_{l_i^{(13)}}$ and $\sigma_{l_i^{(14)}}$ receive a spike from neuron $\sigma_{l_i^{(11)}}$, respectively. Because the spike from neuron $\sigma_{l_i^{(12)}}$ arrives in neuron $\sigma_{l_i^{(14)}}$ after step t_2+2b, the enabled

rule $a \to \lambda$ in neuron $\sigma_{l_i^{(14)}}$ must be applied; that is, the spike received from neuron $\sigma_{l_i^{(11)}}$ is removed before step $t_2 + 2b$. Furthermore, the spike received from neuron $\sigma_{l_i^{(12)}}$ will also be removed by rule $a \to \lambda$. So neuron σ_{l_k} receives no spike. For neuron $\sigma_{l_i^{(13)}}$, there are the following two cases.

(a) If the spikes from neurons $\sigma_{l_i^{(10)}}$ and $\sigma_{l_i^{(11)}}$ arrive at neuron $\sigma_{l_i^{(13)}}$ at a same step, then rule $a^2 \to a; 0$ is enabled and must be applied at one of the next b steps. So neuron σ_{l_j} receives a spike from neuron $\sigma_{l_i^{(13)}}$, which means that system Π starts to simulate the instruction l_j of M.

(b) If the spikes from neurons $\sigma_{l_i^{(10)}}$ and $\sigma_{l_i^{(11)}}$ arrive at neuron $\sigma_{l_i^{(13)}}$ at different steps, then there are two situations.

- If neuron $\sigma_{l_i^{(13)}}$ receives the first spike from neurons $\sigma_{l_i^{(10)}}$ and $\sigma_{l_i^{(11)}}$, and rule $a \to \lambda$ is applied before the second spike from neurons $\sigma_{l_i^{(10)}}$ and $\sigma_{l_i^{(11)}}$ arrives in neuron $\sigma_{l_i^{(13)}}$, then the second spike is also removed by rule $a \to \lambda$. So neuron σ_{l_j} receives no spike. With no spike in neurons σ_{l_j} and σ_{l_k}, computations in system Π abort and output no spike into the environment.
- If neuron $\sigma_{l_i^{(13)}}$ receives the first spike from neurons $\sigma_{l_i^{(10)}}$ and $\sigma_{l_i^{(11)}}$, and rule $a \to \lambda$ is not applied before the second spike from neurons $\sigma_{l_i^{(10)}}$ and $\sigma_{l_i^{(11)}}$ arrives in neuron $\sigma_{l_i^{(13)}}$, then neuron $\sigma_{l_i^{(13)}}$ accumulates two spikes inside. Rule $a^2 \to a; 0$ is enabled and must be applied at one of the next b steps. So neuron σ_{l_j} receives a spike from neuron $\sigma_{l_i^{(13)}}$, which means that system Π starts to simulate the instruction l_j of M.

(2) If rule $a \to a; 2b$ in neuron $\sigma_{l_i^{(11)}}$ is applied, then, similar to case (1), system Π has two possible situations: (i) neuron σ_{l_k} receives a spike, while neuron σ_{l_j} receives no spike, which means that system Π starts to simulate the instruction l_k of M; (ii) neurons σ_{l_j} and σ_{l_k} receive no spike, which means that computations in system Π abort and output no spike into the environment.

Therefore, in module ADD shown in Figure 2.3, from the step when neuron σ_{l_i} fires, system Π has two possibilities: either (1) system Π non-deterministically fires one of neurons σ_{l_j}, σ_{l_k} and the number of spikes in neuron σ_r is increased by two; or (2) computations in system Π abort and no spike is emitted into the environment. In the sense that the number 0 is ignored when the power of two computation devices are compared, instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ of M is correctly simulated by module ADD of system Π.

Similar to module ADD for $r \geq 2$, we can check that module ADD for $r = 1$ correctly simulates an ADD instruction that acts on register 1. Note that, during the simulation, module ADD for $r = 1$ has possible interference with the FIN module shown in Figure 2.5. If neuron σ_1 fires, then it sends a spike to neurons

$\sigma_{l_i^{(4)}}$ and σ_{c_4}. In this case, computations in module ADD abort. In module FIN, neuron σ_{c_4} sends a spike to neuron σ_{c_3}. With 3 spikes inside (initially, neuron σ_{c_3} has two spikes inside), rule $a^3 \to \lambda$ in neuron σ_{c_3} is enabled and applied. So, in module FIN, although neuron σ_1 fires, there is no spike sent into the environment by system Π. Therefore, the interference between module ADD (for $r = 1$) and module FIN will not cause undesired computation results.

It is also possible to have interference among neurons in two ADD modules accessing the same register. Specifically, if there are several ADD instructions l_t that act on register r, then neurons $\sigma_{l_t^{(4)}}$ have the following two cases when instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is simulated. (1) If neuron $\sigma_{l_t^{(4)}}$ receives no spike from neuron σ_r, then it is clear that there is no interference happened among neurons in these ADD modules. (2) If neuron $\sigma_{l_t^{(4)}}$ receives a spike from neuron σ_r, then this spike moves to neuron $\sigma_{l_t^{(5)}}$ and then this spike is removed by rule $a \to \lambda$. Consequently, the interference among ADD modules will not cause undesired steps in Π (i.e., steps that do not correspond to correct simulations of instructions of M).

Module SUB: Simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$. Note that $r \geq 2$.

Module SUB, shown in Figure 2.4, is composed of 14 neurons: neuron σ_r for register r, neurons $\sigma_{l_i}, \sigma_{l_j}, \sigma_{l_k}$ for instructions with labels l_i, l_j, l_k, and 10 auxiliary neurons.

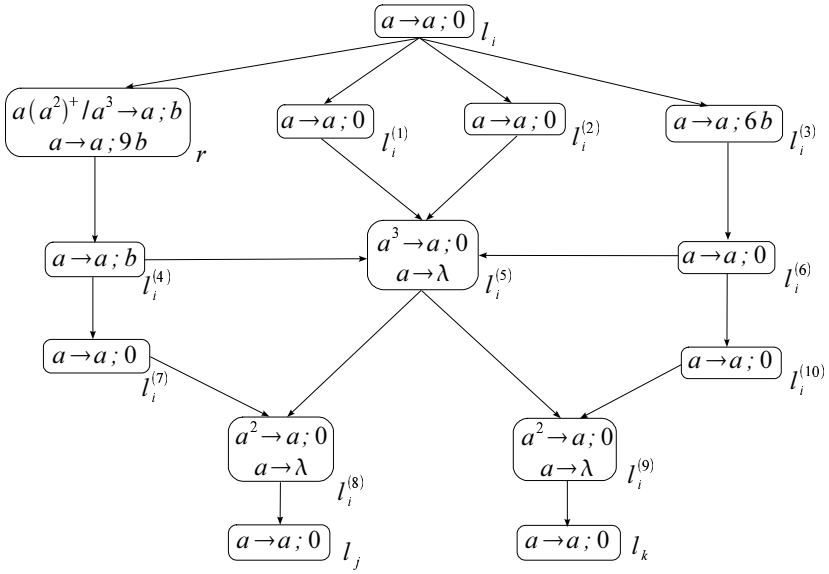

Figure 2.4: Module SUB for simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

Instruction l_i is simulated in Π in the following way. Initially, neuron σ_{l_i} has a spike inside, and other neurons have no spike except for neurons associated with registers. Let t be the moment when neuron σ_{l_i} fires. At step t, neurons σ_r, $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, $\sigma_{l_i^{(3)}}$ receive a spike, respectively.

For neuron $\sigma_{l_i^{(5)}}$, there are the following two possible cases.

(1) If neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at a same step, then neuron $\sigma_{l_i^{(5)}}$ receives 2 spikes at the same step.

(2) If neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at different steps, then neuron $\sigma_{l_i^{(5)}}$ will have two possible situations.

   (a) If neuron $\sigma_{l_i^{(5)}}$ receives the first spike from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$, and the enabled rule $a \to \lambda$ is not applied before the second spike from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ arrives in neuron $\sigma_{l_i^{(5)}}$, then neuron $\sigma_{l_i^{(5)}}$ will accumulate two spikes inside.

   (b) If neuron $\sigma_{l_i^{(5)}}$ receives the first spike from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$, and the enabled rule $a \to \lambda$ is applied before the second spike from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ arrives in neuron $\sigma_{l_i^{(5)}}$, then the second spike will also be removed by rule $a \to \lambda$.

Computations in the above cases complete before step $t + 2b$, which is before the time when spikes from neurons $\sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(6)}}$ arrive in neuron $\sigma_{l_i^{(5)}}$.

For neuron σ_r, there are the following two cases.

(1) If neuron σ_r has $2n$ ($n \geq 1$) spikes at step t, then, at step $t + 1$, neuron σ_r has $2n + 1$ spikes (it received one spike from neuron σ_{l_i}), and the enabled rule $a(a^2)^+/a^3 \to a; b$ fires at one of the next b steps, sending a spike to neuron $\sigma_{l_i^{(4)}}$. This spike arrives in neuron $\sigma_{l_i^{(5)}}$ at one of steps from $t + 2b$ to $t + 4b$. After neuron $\sigma_{l_i^{(5)}}$ receives this spike, it has 1 or 3 spikes inside because the spike along the path $\sigma_{l_i^{(3)}}\sigma_{l_i^{(6)}}\sigma_{l_i^{(5)}}$ arrives in neuron $\sigma_{l_i^{(5)}}$ after step $t + 6b + 1$.

   (a) If neuron $\sigma_{l_i^{(5)}}$ has 1 spike inside, then this spike is removed by rule $a \to \lambda$. So neuron $\sigma_{l_i^{(5)}}$ sends no spike to neuron $\sigma_{l_i^{(8)}}$. In this way, the spike arrived in neuron $\sigma_{l_i^{(8)}}$ along the path $\sigma_{l_i^{(4)}}\sigma_{l_i^{(7)}}\sigma_{l_i^{(8)}}$ will be removed by rule $a \to \lambda$. Similarly, we can check that the spike arrived in neuron $\sigma_{l_i^{(5)}}$ along the path $\sigma_{l_i^{(3)}}\sigma_{l_i^{(6)}}\sigma_{l_i^{(5)}}$ is removed by rule $a \to \lambda$, and the spike arrived in neuron $\sigma_{l_i^{(9)}}$ along the path $\sigma_{l_i^{(6)}}\sigma_{l_i^{(10)}}\sigma_{l_i^{(9)}}$ is also removed by rule $a \to \lambda$. Therefore, no spike arrives in neurons σ_{l_j} and σ_{l_k}, computations in Π abort, and system Π sends no spike into the environment.

(b) If neuron $\sigma_{l_i^{(5)}}$ has 3 spikes, then rule $a^3 \to a; 0$ is enabled and applied, sending a spike to each of neurons $\sigma_{l_i^{(8)}}$ and $\sigma_{l_i^{(9)}}$ at one of steps from $t+2b+1$ to $t+5b$. After this spike arrives in neuron $\sigma_{l_i^{(9)}}$, it is removed by rule $a \to \lambda$, because the spike along the path $\sigma_{l_i^{(3)}}\sigma_{l_i^{(6)}}\sigma_{l_i^{(10)}}\sigma_{l_i^{(9)}}$ arrives in neuron $\sigma_{l_i^{(9)}}$ after step $t+6b+2$.

(i) If spikes from neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(7)}}$ arrive in $\sigma_{l_i^{(8)}}$ at different steps, and they are removed by rule $a \to \lambda$ in turn, then neuron σ_{l_j} receives no spike. So both neurons neuron σ_{l_j} and σ_{l_k} have no spike inside, computations in Π abort, and system Π sends no spike into the environment.

(ii) If spikes from neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(7)}}$ arrive in $\sigma_{l_i^{(8)}}$ at different steps, and neuron $\sigma_{l_i^{(8)}}$ accumulates 2 spikes inside (that is, rule $a \to \lambda$ is not applied before the second spike from neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(7)}}$ arrives in $\sigma_{l_i^{(8)}}$), then rule $a^2 \to a; 0$ is enabled and applied, sending a spike to neuron σ_{l_j}. With one spike inside, neuron σ_{l_j} becomes active, which means that system Π starts to simulate instruction l_j of M.

(iii) If spikes from neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(7)}}$ arrive in $\sigma_{l_i^{(8)}}$ at a same step, then neuron $\sigma_{l_i^{(8)}}$ receives 2 spikes, and rule $a^2 \to a; 0$ is enabled. Rule $a^2 \to a; 0$ is applied at one of the next b steps, sending a spike to neuron σ_{l_j}. With one spike inside, neuron σ_{l_j} becomes active, which means that system Π starts to simulate instruction l_j of M.

In general, if neuron σ_r has $2n$ ($n \geq 1$) spikes at step t, then there are the following two possibilities:

- computations in Π abort and system Π sends no spike into the environment;
- neuron σ_{l_j} becomes active and system Π starts to simulate instruction l_j of M.

(2) If neuron σ_r has no spike inside at step t, then, at step $t+1$, neuron σ_r has 1 spike (it received one spike from neuron σ_{l_i}), and rule $a \to a; 9b$ is enabled. Similar to case (1), we can check that there are two possibilities: (a) computations in Π abort and system Π sends no spike into the environment; (b) neuron σ_{l_k} becomes active and system Π starts to simulate instruction l_k of M.

Therefore, instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ of M is correctly simulated by module SUB of system Π, in the sense that the number 0 is ignored when the power of two computation devices are compared.

It is possible to have interference among neurons in two SUB modules. Specifically, if there are several SUB instructions l_t that act on register r, then when

instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated, neurons $\sigma_{l_t^{(5)}}$ and $\sigma_{l_t^{(8)}}$ receive a spike along the paths $\sigma_r \sigma_{l_t^{(4)}} \sigma_{l_t^{(5)}}$ and $\sigma_r \sigma_{l_t^{(4)}} \sigma_{l_t^{(7)}} \sigma_{l_t^{(8)}}$, respectively. After neurons $\sigma_{l_t^{(5)}}$ and $\sigma_{l_t^{(8)}}$ receive these spikes, rule $a \to \lambda$ is enabled and applied. Consequently, these spikes are removed and the interference among SUB modules will not cause undesired steps in system Π.

Similarly, it is possible to have interference among neurons in an ADD module and a SUB module, which also does not cause undesired steps in system Π. Because register 1 is not subject to SUB instructions, there is no interference among neurons in SUB module and FIN module.

Module FIN: Outputting results of computations.

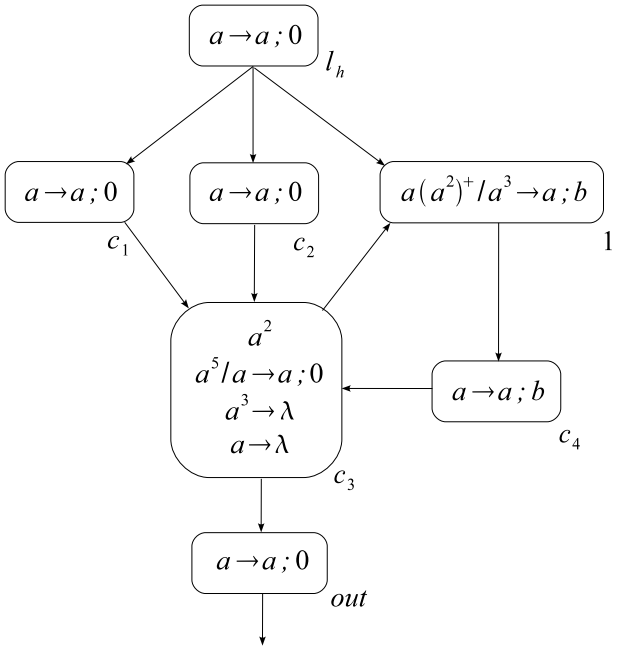

Figure 2.5: Module FIN for outputting results of computations

Module FIN is shown in Figure 2.5. Assume that the computation in M halts, which means that the halting instruction is reached; and the number stored in register 1 is n (we may assume that $n \geq 1$, because number 0 is ignored when the computation power of computing devices is compared). This means that neuron σ_{l_h} in Π has one spike inside and rule $a \to a; 0$ is enabled; and neuron σ_1 has $2n$ spikes. Suppose that neuron σ_{l_h} fires at step t_h. At step t_h, neurons σ_{c_1}, σ_{c_2} and σ_1 receive a spike from neuron σ_{l_h}, respectively. Each of neurons σ_{c_1} and σ_{c_2} fires at one of the next b steps, sending a spike to neuron σ_{c_3}. No matter whether neurons σ_{c_1} and σ_{c_2} fire at a same step or different steps, neuron σ_{c_3} has two possibilities: (1) it accumulates 4 spike inside (note that it initially has 2 spikes

inside); (2) rules $a^3 \to a; 0$ and $a \to \lambda$ are applied in turn, and all spikes in neuron σ_{c_3} are removed. The spike along the path $\sigma_1\sigma_{c_4}\sigma_{c_3}$ arrives in neuron σ_{c_3} after step $t_h + 2b$, so neuron σ_{c_3} has 0 or 4 spikes at step $t_h + 2b$.

- If neuron σ_{c_3} has no spike at step $t_h + 2b$, then the spike received along the path $\sigma_1\sigma_{c_4}\sigma_{c_3}$ is removed by rule $a \to \lambda$. The computation aborts and system Π sends no spike into the environment.
- If neuron σ_{c_3} has 4 spikes at step $t_h + 2b$, then after the spike along the path $\sigma_1\sigma_{c_4}\sigma_{c_3}$ arrives in neuron σ_{c_3}, neuron σ_{c_3} has 5 spikes. Rule $a^5/a \to a; 0$ is enabled and applied, sending a spike to neurons σ_{out} and σ_1, respectively. With a spike inside, neuron σ_{out} fires, sending a spike into the environment. After neuron σ_1 receives a spike from neuron σ_{c_3}, the number of spikes in it becomes odd again.
  - If the number of spikes in neuron σ_1 is odd and not less than 3, then neuron σ_1 fires again by rule $a(a^2)^+/a^3 \to a; b$, sending a spike to neuron σ_{c_4}. This spike then moves to neuron σ_{c_3} by rule $a \to a; b$. Neuron σ_{c_3} will fire again by rule $a^5/a \to a; 0$.
  - If the number of spikes in neuron σ_1 is 1, then no rule is enabled in neuron σ_1. Furthermore, all neurons in system Π are open. The computation in Π ends, and the number of spikes sent into the environment by system Π is n, which is exactly the number stored in register 1 of M when the computation of M halts.

From the description of the modules and their work, it is clear that the register machine M is correctly simulated by system Π. It is easy to check that all rules in system Π are standard, so $N_{gen}^{lasyn}(\Pi) = N(M)$ and Theorem 1 holds. ∎

2.6 Limited Asynchronous SN P Systems with an Observer

In the proof of Theorem 1, even if neuron σ_1 associated with register 1 becomes active (this means that the computation in M reaches instruction `HALT`, and it is already correctly simulated by system Π), during the stage of outputting results of computations, it is still possible that computations in system Π abort and no spike is sent into the environment. That is, it is possible that system Π fails to signal an observer the correct computation result stored in neuron σ_1. It is natural to consider that all computation results of correct simulations are collected by an observer.

In the proof of Theorem 1, neurons are introduced for the registers, where neuron σ_1 for the first register is implemented differently from the other neuron registers σ_r. From a mathematical point of view, it is nice to have a uniform neuron translation for the registers, so we can verify the ADD – module independent from the register used.

With the above two motivations from the proof of Theorem 1, we give a new definition of successful computations and associated computation results. A computation in a system Π is considered as successful if it satisfies the following two conditions: (1) it reaches a configuration where all neurons are open and no rule can be used; (2) system Π sends at least one spike into the environment, which tells an observer that the computation has finished. The result of a successful computation is defined as the number encoded by the number of spikes present in a specified neuron. Note that even if a computation reaches a configuration where all neurons are open and no rule can be used, but system Π sends no spike into the environment, then the number encoded by the number of spikes present in a specified neuron will not be collected by an observer.

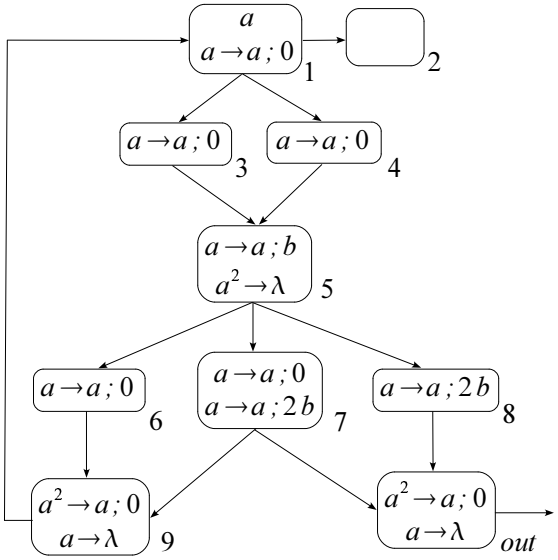

Figure 2.6: A limited asynchronous SN P system Π′ with time bound k.

An example, shown in Figure 2.6, is given to illustrate the new definition, where the time bound associated with all rules is b and the result of a successful computation is defined as the number of spikes in neuron σ_2. Actually, system Π′ in Figure 2.6 is obtained by modifying and combining system Π in Figure 2.2 and Part 2 of module ADD in Figure 2.3. Following the explanation of the work of system Π and module ADD, we can easily check the work of system Π′. So, we here only sketchily explain the work of system Π′ in order to illustrate the new definition.

Initially, neuron σ_1 has one spike. So, rule $a \to a; 0$ in neuron σ_1 is enabled at step 1, and at one of the next k steps rule $a \to a; 0$ is applied, sending a spike to neurons σ_2, σ_3, σ_4, respectively. For neuron σ_5, there are the following two cases.

(1) If rule $a^2 \to \lambda$ is applied, then all neurons in system Π′ are open and no rule

is enabled. Note that system Π' sends no spike into the environment. So, although there is one spike in neuron σ_2, by the new definition of successful computations and computation results, the number 1 encoded by the number of spikes in neuron σ_2 is not collected by an observer. (Note that, by the standard definition of successful computation and computation result given in Section 2.3, the number 1 should be collected as a computation result.)

(2) If $a \to a; b$ is applied, then, similar to the work of module ADD, there are the following three cases.

(a) If neurons σ_9 and σ_{out} remove the spikes received from neurons σ_6, σ_7, σ_8, then the computation aborts and no spike is sent into the environment. So, the computation result in neuron σ_2 is not collected by an observer.

(b) If neuron σ_9 fires and σ_{out} removes the spikes received from neurons σ_7, σ_8, then the computation continues.

(c) If neuron σ_9 removes the spikes received from neurons σ_6, σ_7, and σ_{out} fires, then the computation halts. Because neuron σ_{out} sends a spike into the environment, the computation is a successful one and the computation result in neuron σ_2 is collected by an observer.

Therefore, the set of numbers generated by system Π' is $\mathbb{N} \setminus \{0\}$.

With the new definition of successful computations and computation results, the modules in the proof of Theorem 1 can be modified as follows. We take module ADD in Figure 2.3 as a module for all ADD instructions (that is, the set of rules in neuron σ_1 is the same with the set of rules in neurons σ_r, $r \geq 2$). Module SUB keeps unchanged. Module FIN is simplified as shown in Figure 2.7, whose functioning is just to signal an observer that the computation in system halts. As in the proof of Theorem 1, number n computed by system is encoded by $2n$ spikes in neuron σ_1 associated with register 1. We leave the check of the modified system to readers, which is quite similar with the proof of Theorem 1. We summarize the above discussion as the following theorem.

Theorem 2
There exist universal limited asynchronous SN P systems with an observer.

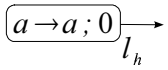

Figure 2.7: Module FIN for signalling an observer that computations halt.

2.7 Finite Limited Asynchronous SN P Systems

In this section, we investigate the computation power of limited asynchronous SN P systems with a bound on the number of spikes present in the neurons. This will restrict the behaviour of the system into a finite number of states, but we demonstrate the systems can still generate all semilinear sets of numbers.

Let us denote by $N_{gen}^{lasyn}SNP(bound_*)$ the family of sets of numbers generated by limited asynchronous SN P systems with a (unspecified) bound on the number of spikes at any time in any neuron.

Lemma 3
$SLIN \subseteq N_{gen}^{lasyn}SNP(bound_*)$.

Proof
We start by observing that every semilinear set of natural numbers is the length set of a regular language over a one letter alphabet. Looking at deterministic automata for regular languages over a single letter e, we observe that they are of a special form as shown in Figure 2.8. Specifically, the figure shows a generic deterministic automation A over a single letter alphabet $\{e\}$ with $n+1$ states, where the state set $Q = \{0, 1, \ldots, n\}$, there is a transition from every state i to the next one $i+1$, $0 \leq i < n$, while the last state n has a transition to one of the predecessors q, $0 < q \leq n$, finally the set of final states equals $F = \{i_1, \ldots, i_j\} \subseteq Q$.

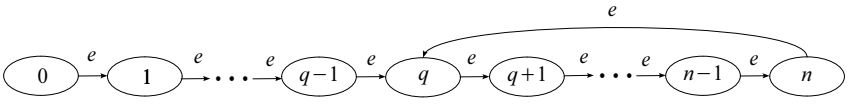

Figure 2.8: A deterministic automaton over a single letter alphabet {e}

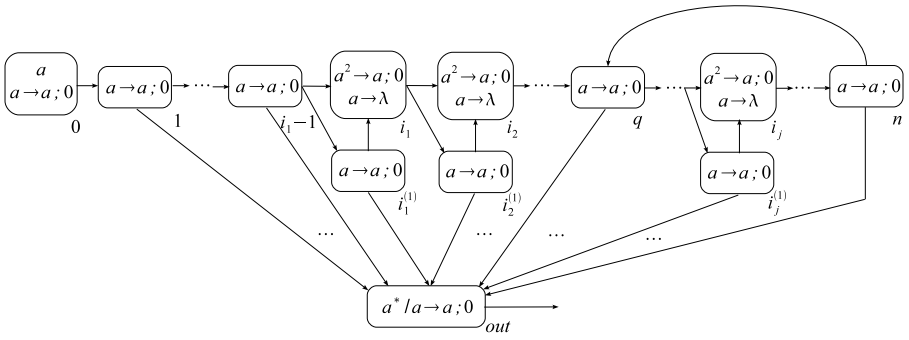

Figure 2.9: The structure of limited asynchronous SN P system Π

An automaton A of the above form can be simulated by a limited asynchronous SN P system Π (the time bound is b steps, for fixed but arbitrary $b \geq 2$). The

structure of system Π is shown in Figure 2.9. For each state i in automaton A, a neuron σ_i is associated in system Π, and for each final state i_v $(1 \leq v \leq j, i_v \in F)$ we add a neuron $\sigma_{i_v^{(1)}}$ located between neurons σ_{i_v-1} and σ_{i_v}. Each neuron $\sigma_{i_v^{(1)}}$ can receive a spike from neuron σ_{i_v-1} and send a spike to neuron σ_{i_v} and σ_{out}, respectively. Neuron σ_{out} is used to output computation results and has a spiking rule $a^*/a \to a; 0$. Each neuron σ_{i_v} (associated with a final state i_v) in system Π has two rules: $a^2 \to a; 0$ and $a \to \lambda$; the other neurons (except the output neuron) have one spiking rule $a \to a; 0$. In the initial configuration of system Π, only neuron σ_0 has one spike inside; the other neurons have no spike.

With one spike inside, rule $a \to a; 0$ in neuron σ_0 is enabled. Neuron σ_0 will fire at one of the next b steps, sending a spike to neuron σ_1. Neuron σ_1 receives the spike and will send a spike (at one of the next b steps) to neurons σ_2 and σ_{out}, respectively. Neuron σ_{out} receives this spike and will transfer this spike to the environment (possibly after some steps, but as the rule is always enabled it has to fire eventually). By the spiking rules ($a \to a; 0$ or $a^2 \to a; 0$) in each neuron σ_i $(0 \leq i \leq n)$ and $\sigma_{i_v^{(1)}}$ $(1 \leq v \leq j)$, the spike from neuron σ_0 moves along the path $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ and the cycle $(\sigma_n, \sigma_q, \sigma_{q+1}, \ldots, \sigma_n)$. During the journey of the spike from neuron σ_0, when neuron σ_{i_v} (associated to a final state i_v in automation A) and its additional neuron $\sigma_{i_v^{(1)}}$ simultaneously receive a spike from neuron σ_{i_v-1}, then neuron $\sigma_{i_v^{(1)}}$ will send a spike to neurons σ_{out} and σ_{i_v} at one of the next b steps. For neuron σ_{i_v} —first receiving a spike from neuron σ_{i_v-1} then within the next b steps getting the second spike from $\sigma_{i_v^{(1)}}$— there are following two possible cases.

(1) If neuron σ_{i_v} receives the first spike from neuron σ_{i_v-1}, and the enabled rule $a \to \lambda$ is not applied before the second spike from neuron $\sigma_{i_v^{(1)}}$ arrives in neuron σ_{i_v}, then neuron σ_{i_v} will accumulate two spikes inside and rule $a^2 \to a; 0$ is enabled, sending a spike to neuron σ_{i_v+1} (it is possible that neuron σ_{i_v+1} is associated to the next final state i_{v+1}, in this way, neuron σ_{i_v} will send a spike to neurons $\sigma_{i_{(v+1)}}$ and $\sigma_{i_{(v+1)}^{(1)}}$, respectively). In this case, the spike in neuron σ_{i_v+1} continues to move, and neuron σ_{out} receives a spike from neuron $\sigma_{i_v^{(1)}}$ and will send a spike the environment.

(2) If neuron σ_{i_v} receives the first spike from neuron σ_{i_v-1}, and the enabled rule $a \to \lambda$ is applied before the second spike from neurons $\sigma_{i_v^{(1)}}$ arrives in neuron σ_{i_v}, then the second spike will also be removed by rule $a \to \lambda$, which means that the spike can not continue to move. So, neuron σ_{out} receives one spike from neuron σ_{i_v} and will send the last spike to the environment. In this way, the system has chosen to halt in the final state of the automaton A

Note that in the both above cases neuron σ_{out} will send a spike (received from neuron $\sigma_{i_v^{(1)}}$) to the environment, which records that there is a transition from state $i_v - 1$ to i_v. Therefore, the number of spikes produced by neuron σ_{out} equals the length of the path $(\sigma_0, \sigma_1, \ldots, \sigma_{i_v})$ and several possible copies of the

cycle $(\sigma_{i_v},\ldots,\sigma_n,\sigma_q,\ldots,\sigma_{i_v})$ (for the case of $i_v \geq q$), which corresponds to the length of a string generated by automaton A. In general, the length set of strings generated by automaton A is a subset of $N_{gen}^{lasyn}(\Pi)$. ∎

Lemma 4
$N_{gen}^{lasyn}SNP(bound_*) \subseteq SLIN$.

Proof
Take a limited asynchronous SN P system Π with a bound on the number of spikes in each neuron. The number of neurons is fixed, the state of each neuron (open/closed, the time since rules became applicable) has a bounded number of values, and moreover the number of spikes in each neuron is bounded, hence the number of configurations reached by Π is finite. Let $\mathcal{C}$ be the set of configurations of Π, and let C_0 be the initial configuration of Π.

We simply construct the right-linear grammar $G = (\mathcal{C}, \{a\}, C_0, P)$, where P contains the following rules:

(1) $C \to C'$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Rightarrow C'$ in Π during which the output neuron does not spike;

(2) $C \to aC'$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Rightarrow C'$ in Π during which the output neuron spikes;

(3) $C \to \lambda$, for any $C \in \mathcal{C}$ in which all neurons are open and no rule is applicable.

Clearly the construction N ensures the fact that $N_{gen}^{lasyn}(\Pi)$ is the length set of the regular language $L(G)$, hence it is semilinear. Therefore, $N_{gen}^{lasyn}SNP(bound_*) \subseteq SLIN$. ∎

From the preceding two results we conclude the characterization of N_{gen}^{lasyn} SNP $(bound_*)$ as semi-linear sets (over a single symbol).

Theorem 5
$N_{gen}^{lasyn}SNP(bound_*) = SLIN$.

2.8 Conclusions and Remarks

With a biological motivation, we have considered a new working mode of SN P systems, called limited asynchronous mode. In an SN P system working in limited asynchronous mode, a common bound on the time interval is associated with all spiking rules. If a rule is enabled at some step, this rule is not obligatorily used immediately. However, if from this step on, the unused rule can be used in each moment of the time interval, it should be used within the given time interval. If further spikes make the rule non-applicable, then the computation continues in the new circumstances. We have proved that limited asynchronous SN P systems with standard spiking rules are universal.

In the proof of the universality result in this work, the neurons are allowed to hold arbitrarily many spikes. If the number of spikes present in each neuron of a limited asynchronous SN P system with standard spiking rules is bounded during a computation, then the power of a limited asynchronous SN P system with standard spiking rules falls drastically, and we get a characterization of semilinear sets of numbers.

In the proof of Theorem 1, delays are used in the spiking rules with the values such as $2b$, $9b$, $14b$. The functioning of these delays is to ensure that a spike should arrive later than another spike at the target neuron. It is possible to remove all delays in the spiking rules. The basic idea is that a delay k is replaced by k neurons with the spiking rule $a \to a$ (the functioning of these k neurons is to transmit the spike). If we consider the time bound b as a measure of asynchronous degree, then each neuron has asynchronous degree b. However, k neurons have asynchronous degree kb in the sense that the spike passing along the k neurons with spiking rule $a \to a$ arrives at the target neuron at a moment in the time interval of length kb. That is, the replacement of delays with neurons increases the asynchronous degree. So, the above idea might work, but the replacement is not trivial, the technical details should be carefully checked.

In this work, a time bound of application of rules is given for all rules. Of course, we can consider that a different time bound is associated with each rule instead of a uniform time bound for all rules. It will be interesting to check whether in this case simpler ADD, SUB, and FIN modules can be obtained than these modules presented in this paper. It remains open whether forgetting rules can be removed in a universal limited asynchronous SN P system if a different time bound is associated with each rule. Possible normal forms of limited asynchronous SN P systems are also worth to be investigated (e.g., refer to [13]).

As usual, it is worth to investigate many other computational properties of limited asynchronous SN P systems under different computational modes. An interesting problem among them is to design a homogeneous universal limited asynchronous SN P system, where "homogeneous" means that each neuron has the same set of spiking rules [41].

Acknowledgements

The work of L. Pan and J. Wang was supported by National Natural Science Foundation of China (61033003 and 30870826), Ph.D. Programs Foundation of Ministry of Education of China (20100142110 072), Fundamental Research Funds for the Central Universities (2010ZD001), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The authors would like to thank the three anonymous referees of the paper for their valuable comments and suggestions.