Deterministic Solutions to QSAT and Q3SAT by Spiking Neural P Systems with Pre-Computed Resources

Tseren-Onolt Ishdorj[1], Alberto Leporati[2], Linqiang Pan[3⋆],
Xiangxiang Zeng[3], Xingyi Zhang[3]

[1] Computational Biomodelling Laboratory
Department of Information Technologies
Åbo Akademi University, Turku 20520, Finland
`tishdorj@abo.fi`

[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
`alberto.leporati@unimib.it`

[3] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, People's Republic of China
`lqpan@mail.hust.edu.cn, zxxhust@gmail.com, xyzhanghust@gmail.com`

Summary. In this paper we continue previous studies on the computational efficiency of spiking neural P systems, under the assumption that some pre-computed resources of exponential size are given in advance. Specifically, we give a deterministic solution for each of two well known **PSPACE**-complete problems: QSAT and Q3SAT. In the case of QSAT, the answer to any instance of the problem is computed in a time which is linear with respect to both the number n of Boolean variables and the number m of clauses that compose the instance. As for Q3SAT, the answer is computed in a time which is at most cubic in the number n of Boolean variables.

1 Introduction

Spiking neural P systems (in short, SN P systems) were introduced in [7] in the framework of Membrane Computing [15] as a new class of computing devices which are inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons to other neurons. Since then, many computational properties of SN P systems have been studied; for example, it has been proved

⋆ Corresponding author. Tel.: +86-27-87556070. Fax: +86-27-87543130.

that they are Turing-complete when considered as number computing devices [7], when used as language generators [4, 2] and also when computing functions [13].

Investigations related to the possibility to solve computationally hard problems by using SN P systems were first proposed in [3]. The idea was to encode the instances of decision problems in a number of spikes, to be placed in an (arbitrarily large) pre-computed system at the beginning of the computation. It was shown that the resulting SN P systems are able to solve the **NP**-complete problem `SAT` (the satisfiability of propositional formulas expressed in conjunctive normal form) in a constant time. Slightly different solutions to `SAT` and `3-SAT` by using SN P systems with pre-computed resources were considered in [8]; here the encoding of an instance of the given problem is introduced into the pre-computed resources in a polynomial number of steps, while the truth values are assigned to the Boolean variables of the formula and the satisfiability of the clauses is checked. The answer associated to the instance of the problem is thus computed in a polynomial time. Finally, very simple semi-uniform and uniform solutions to the numerical **NP**-complete problem `Subset Sum` — by using SN P systems with exponential size pre-computed resources — have been presented in [9]. All the systems constructed above work in a deterministic way.

A different idea of constructing SN P systems for solving **NP**-complete problems was given in [11, 12], where the `Subset Sum` and `SAT` problems were considered. In these papers, the solutions are obtained in a semi-uniform or uniform way by using nondeterministic devices but without pre-computed resources. However, several ingredients are also added to SN P systems such as extended rules, the possibility to have a choice between spiking rules and forgetting rules, etc. An alternative to the constructions of [11, 12] was given in [10], where only standard SN P systems without delaying rules, and having a uniform construction, are used. However, it should be noted that the systems described in [10] either have an exponential size, or their computations last an exponential number of steps. Indeed, it has been proved in [12] that a deterministic SN P system of polynomial size cannot solve an **NP**-complete problem in a polynomial time unless **P**=**NP**. Hence, under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency of the system (pre-computed resources, ways to exponentially grow the workspace during the computation, nondeterminism, and so on).

The present paper deals with `QSAT` (the satisfiability of fully quantified propositional formulas expressed in conjunctive normal form) and with `Q3SAT` (where the clauses that compose the propositional formulas have exactly three literals), two well known **PSPACE**-complete problems. For `QSAT` we provide a family $\{\Pi_{QSAT}(2n,m)\}_{n,m\in\mathbb{N}}$ of SN P systems with pre-computed resources such that for all $n, m \in \mathbb{N}$ the system $\Pi_{QSAT}(2n,m)$ solves all the instances of `QSAT` which are built using $2n$ Boolean variables and m clauses. Each system $\Pi_{QSAT}(2n,m)$ is deterministic, and computes the solution in a time which is linear with respect to both n and m; however, the size of $\Pi_{QSAT}(2n,m)$ is exponential with respect to the size of the instances of the problem. As for `Q3SAT`, we provide a family

$\{\Pi_{Q3SAT}(2n)\}_{n\in\mathbb{N}}$ of SN P systems with pre-computed resources, such that for all $n \in \mathbb{N}$ the system $\Pi_{Q3SAT}(2n)$ solves all possible instances of Q3SAT which can be built using $2n$ Boolean variables. Also in this case the systems $\Pi_{Q3SAT}(2n)$ are deterministic and have an exponential size with respect to n. Given an instance of Q3SAT, the corresponding answer is computed in a time which is at most cubic in n.

An important observation is that we will not specify how our pre-computed systems could be built. However, we require that such systems have a *regular* structure, and that they do not contain neither "hidden information" that simplify the solution of specific instances, nor an encoding of all possible solutions (that is, an exponential amount of information that allows to cheat while solving the instances of the problem). These requirements were inspired by open problem Q27 in [15]. Let us note in passing that the regularity of the structure of the system is related to the concept of *uniformity*, that in some sense measures the difficulty of constructing the systems. Usually, when considering families $\{C(n)\}_{n\in\mathbb{N}}$ of Boolean circuits, or other computing devices whose number of inputs depends upon an integer parameter $n \geq 1$, it is required that for each $n \in \mathbb{N}$ a "reasonable" description (see [1] for a discussion on the meaning of the term "reasonable" in this context) of $C(n)$, the circuit of the family which has n inputs, can be produced in polynomial time and logarithmic space (with respect to n) by a deterministic Turing machine whose input is 1^n, the unary representation of n. In this paper we will not delve further into the details concerning uniformity; we just rely on reader's intuition, by stating that it should be possible to build the entire structure of the system in a polynomial time, using only a polynomial amount of information and a controlled replication mechanism, as it already happens in P systems with cell division. We will thus say that our solutions are *exp-uniform* (instead of *uniform*), since the systems $\Pi_{QSAT}(2n, m)$ and $\Pi_{Q3SAT}(2n)$ have an exponential size.

The paper is organized as follows. In section 2 we recall the formal definition of SN P systems, as well as some mathematical preliminaries that will be used in the following. In section 3 we provide an exp-uniform family $\{\Pi_{QSAT}(2n, m)\}_{n,m\in\mathbb{N}}$ of SN P systems with pre-computed resources such that for all $n, m \in \mathbb{N}$ the system $\Pi_{QSAT}(2n, m)$ solves all possible instances of QSAT containing $2n$ Boolean variables and m clauses. In section 4 we present an exp-uniform family $\{\Pi_{Q3SAT}(2n)\}_{n\in\mathbb{N}}$ of SN P systems with pre-computed resources such that for all $n \in \mathbb{N}$ the system $\Pi_{Q3SAT}(2n)$ solves all the instances of Q3SAT which can be built using $2n$ Boolean variables. Section 5 concludes the paper and suggests some possible directions for future work.

2 Preliminaries

We assume the reader to be familiar with formal language theory [16], computational complexity theory [5] as well as membrane computing [15]. We mention here only a few notions and notations which are used throughout the paper.

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 simply write a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$.

A regular expression over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over V. Non-necessary parentheses can be omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

For a string $str = y_1 y_2 \dots y_{2n}$, where $y_{2k-1} \in \{0, 1\}$, $y_{2k} \in \{0, 1, x_{2k}\}$, $1 \leq k \leq n$, we denote by $str|_i$ the ith symbol of the string str, $1 \leq i \leq 2n$. For given $1 \leq i \leq n$, if str such that the $2j$th symbol is x_{2j} for all $j \leq i$ and the $2j'$th symbol equals to 1 or 0 for all $j' \geq i$, then we denote by $str|_{2i} \leftarrow x$ a string which is obtained by replacing the $2i$th symbol of str with x_{2i}. In particular, for a binary string $bin \in \{0, 1\}^{2n}$, $bin|_i$ and $bin|_2 \leftarrow x_2$ are defined as the ith bit of bin and the string obtained by replacing the second bit of bin with x_2, respectively.

`QSAT` is a well known **PSPACE**-complete decision problem (see for example [5, pages 261–262], where some variants of the problem `Quantified Boolean Formulas` are defined). It asks whether or not a given fully quantified Boolean formula, expressed in the conjunctive normal form (CNF), evaluates to *true* or *false*. Formally, an instance of `QSAT` with n variables and m clauses is a formula $\gamma_{n,m} = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n (C_1 \wedge C_2 \wedge \ldots \wedge C_m)$ where each Q_i, $1 \leq i \leq n$, is either $\forall$ or $\exists$, and each clause C_j, $1 \leq j \leq m$, is a disjunction of the form $C_j = y_1 \vee y_2 \vee \ldots \vee y_{r_j}$, with each literal y_k, $1 \leq k \leq r_j$, being either a propositional variable x_s or its negation $\neg x_s$, $1 \leq s \leq n$. For example, the propositional formula $\beta = Q_1 x_1 Q_2 x_2 [(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)]$ is *true* when $Q_1 = \forall$ and $Q_2 = \exists$, whereas it is *false* when $Q_1 = \exists$ and $Q_2 = \forall$. The decision problem `Q3SAT` is defined exactly as `QSAT`, the only difference being that all the clauses now contain *exactly* three literals. It is known that even under this restriction the problem remains **PSPACE**-complete (see, for example, [5, page 262]).

In what follows we require that no repetitions of the same literal may occur in any clause. Without loss of generality we can also avoid the clauses in which both the literals x_s and $\neg x_s$, for any $1 \leq s \leq n$, occur. Further, we will focus our attention on the instances of `QSAT` and `Q3SAT` in which all the variables having an even index (that is, $x_2, x_4, \ldots$) are *universally* quantified, and all the variables with an odd index $(x_1, x_3, \ldots)$ are *existentially* quantified. We will say that such instances are expressed in *normal form*. This may be done without loss of generality. In fact, for any instance $\gamma_{n,m} = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n (C_1 \wedge C_2 \wedge \ldots \wedge C_m)$ of `QSAT` having n variables and m clauses there exists an equivalent instance $\gamma'_{2n,m} = \exists x'_1 \forall x'_2 \ldots \exists x'_{2n-1} \forall x'_{2n} (C'_1 \wedge C'_2 \wedge \ldots \wedge C'_m)$ with $2n$ variables, where each clause C'_j is obtained from C_j by replacing every variable x_i by x'_{2i-1} if $Q_i = \exists$, or by x'_{2i} if $Q_i = \forall$. Note that this transformation may require to introduce some "dummy" variables, that is, variables which are quantified in $\gamma'_{2n,m}$ to guarantee the alternance of even-numbered and odd-numbered variables, but that nonetheless do not appear in any clause. For example, for the propositional formula $\beta_1 = \forall x_1 \exists x_2[(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)]$ the normal form is $\beta'_1 = \exists x'_1 \forall x'_2 \exists x'_3 \forall x'_4[(x'_2 \vee x'_3) \wedge (\neg x'_2 \vee \neg x'_3)]$; for the propositional formula $\beta_2 = \exists x_1 \forall x_2[(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)]$ we have the normal form $\beta'_2 = \exists x'_1 \forall x'_2 \exists x'_3 \forall x'_4[(x'_1 \vee x'_4) \wedge (\neg x'_1 \vee \neg x'_4)]$. The same transformation may be applied on any instance of Q3SAT defined on n Boolean variables; in this case the result will be another instance of Q3SAT, defined on $2n$ variables. From now on we will denote by $QSAT(2n, m)$ the set of all possible instances of QSAT, expressed in the above normal form, which are built using $2n$ Boolean variables and m clauses. Similarly, we will denote by $Q3SAT(2n)$ the set of all possible instances of Q3SAT, expressed in normal form, which can be built using $2n$ Boolean variables.

2.1 Spiking neural P systems

As stated in the Introduction, SN P systems have been introduced in [7], in the framework of Membrane Computing. They can be considered as an evolution of P systems, corresponding to a shift from *cell-like* to *neural-like* architectures.

In SN P systems the cells (also called *neurons*) are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consists of a number of copies of a single object type, called the *spike*. Every cell may also contain a number of *firing* and *forgetting* rules. Firing rules allow a neuron to send information to other neurons in the form of electrical impulses (also called *spikes*) which are accumulated at the target cell. The applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use one of its rules, then one of such rules must be used. If two or more rules could be applied, then only one of them is nondeterministically chosen. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other. Note that, as usually happens in Membrane Computing, a global clock is assumed, marking the time for the whole system, and hence the functioning of the system is synchronized. When a cell sends out spikes it becomes "closed" (inactive) for a specified period of time, that reflects the refractory period of biological neurons. During this period, the neuron does not accept new inputs and cannot "fire" (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike arrives at the target. In SN P systems this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, then there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

Formally, a *spiking neural membrane system* (SN P system, for short) of degree $m \geq 1$, as defined in [7], is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \dots, \sigma_m$ 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 form $E/a^c \to a^p; d$, where E is a regular expression over a, and $c \geq 1$, $p \geq 0$, $d \geq 0$, with the restriction $c \geq p$;
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \dots, m\}$ indicate the *input* and the *output* neurons of Π.

A rule $E/a^c \to a^p; d$ with $p \geq 1$ is an *extended firing* (we also say spiking) rule; a rule $E/a^c \to a^p$ with $p = 0$ is written in the form $E/a^c \to \lambda$ and is called an *extended forgetting* rule. Rules of the types $E/a^c \to a; d$ and $a^c \to \lambda$ are said to be *standard*.

If a rule $E/a^c \to a^p; d$ has $E = a^c$, then we will write it in the simplified form $a^c \to a^p; d$; similarly, if a rule $E/a^c \to a^p; d$ has $d = 0$, then we can simply write it as $E/a^c \to a^p$; hence, if a rule $E/a^c \to a^p; d$ has $E = a^c$ and $d = 0$, then we can write $a^c \to a^p$.

The rules are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a^p; d$ is enabled and can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in neuron σ_i); the neuron is fired, and it produces p spikes after d time units. If $d = 0$, then the spikes are emitted immediately; if $d = 1$, then the spikes are emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \dots, t+d-1$ the neuron is closed (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost). In the step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can again apply rules). Once emitted from neuron σ_i, the p spikes reach immediately all neurons σ_j such that $(i, j) \in syn$ and which are open, that is, the p spikes are replicated and each target neuron receives p spikes; as stated above, spikes sent to a closed neuron are "lost", that is, they are removed from the system. In the case of the output neuron, p spikes are also sent to the environment. Of course, if neuron σ_i has no synapse leaving from it, then the produced spikes are lost. If the rule is a forgetting one of the form $E/a^c \to \lambda$, then, when it is applied, $c \geq 1$ spikes are removed.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. Since two firing rules $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$ can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron; in such a case, only one of them is chosen in a nondeterministic way. However it is assumed that if a firing rule is applicable then no forgetting rule is applicable, and vice versa. Thus, the rules are used in the sequential manner in each neuron (at most one in each step), but neurons work in parallel with each other. It is important to note that the applicability of a rule is established depending on the total number of spikes contained in the neuron.

The *initial configuration* of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron, with all neurons being open. During the computation, a *configuration* is described by both the number of spikes present in each neuron and the *state* of the neuron, that is, the number of steps to count down until it becomes open again (this number is zero if the neuron is already open). Thus, $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ is the configuration where neuron σ_i contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps, for $i = 1, 2, \ldots, m$; with this notation, the initial configuration of the system is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$. Using the rules as described above, one can define *transitions* among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation *halts* if it reaches a configuration where all neurons are open and no rule can be used.

Since in SN P systems the alphabet contains only one symbol (denoted by a), the input information is sometimes encoded as a sequence of "virtual" symbols, λ or a^i, $i \geq 1$, where λ represents no spike and a^i represents a multiset of i spikes. The input sequence is then introduced in the input neuron one virtual symbol at one time unite, starting from the leftmost symbol of the sequence. For instance, the sequence $a^2 \lambda a^3$ is composed of three virtual symbols: a^2, λ and a^3. When providing this sequence as input to an SN P system, the virtual symbol a^2 (that is, two spikes) is introduced at the first computation step, followed by λ (0 spikes) at the next step, and finally by a^3 (three spikes) at the third step.

Another useful extension to the model defined above, already considered in [10, 12, 11, 8], is to use several input neurons, so that the introduction of the encoding of an instance of the problem to be solved can be done in a faster way, introducing parts of the code in parallel in various input neurons. Formally, we can define an SN P system of degree (m, ℓ), with $m \geq 1$ and $0 \leq \ell \leq m$, just like a standard SN P system of degree m, the only difference being that now there are ℓ input neurons denoted by $in_1, \ldots, in_\ell$. A *valid input* for an SN P system of degree (m, ℓ) is a set of ℓ binary sequences (where each element of the sequence denotes the presence or the absence of a spike), that collectively encode an instance of a problem.

Spiking neural P systems can be used to solve decision problems, both in a *semi–uniform* and in a *uniform* way. When solving a problem $\mathcal{Q}$ in the *semi–uniform* setting, for each specified instance $\mathcal{I}$ of $\mathcal{Q}$ we build in a polynomial time (with respect to the size of $\mathcal{I}$) an SN P system $\Pi_{\mathcal{Q},\mathcal{I}}$, whose structure and initial configuration depend upon $\mathcal{I}$, that halts (or emits a specified number of spikes in a given interval of time) if and only if $\mathcal{I}$ is a positive instance of $\mathcal{Q}$. On the other hand, a *uniform* solution of $\mathcal{Q}$ consists of a family $\{\Pi_{\mathcal{Q}}(n)\}_{n \in \mathbb{N}}$ of SN P systems such that, when having an instance $\mathcal{I} \in \mathcal{Q}$ of size n, we introduce a polynomial (in n) number of spikes in a designated (set of) input neuron(s) of $\Pi_{\mathcal{Q}}(n)$ and the computation halts (or, alternatively, a specified number of spikes is emitted in a given interval of time) if and only if $\mathcal{I}$ is a positive instance. The preference for uniform solutions over semi–uniform ones is given by the fact that they are more strictly related to the structure of the problem, rather than to specific instances. Indeed, in the semi–uniform setting we do not even need any input neuron, as the instance of the problem is embedded into the structure (number of spikes, graph of neurons and synapses, rules) from the very beginning. If the instances of a problem $\mathcal{Q}$ depend upon two parameters (as is the case of `QSAT`, where n is the number of variables and m the number of clauses in a given formula), then we will denote the family of SN P systems that solves $\mathcal{Q}$ by $\{\Pi_{\mathcal{Q}}(n,m)\}_{n,m\in\mathbb{N}}$. Alternatively, if one does not want to make the family of SN P systems depend upon two parameters, it is possible to define it as $\{\Pi_{\mathcal{Q}}(\langle n,m\rangle)\}_{n,m\in\mathbb{N}}$, where $\langle n,m\rangle$ indicates the positive integer number obtained by applying an appropriate bijection (for example, Cantor's pairing) from $\mathbb{N}^2$ to $\mathbb{N}$.

In the above definitions it is assumed that the uniform (resp., semi-uniform) construction of $\Pi_{\mathcal{Q}}(n)$ (resp., $\Pi_{\mathcal{Q},\mathcal{I}}$) is performed by using a deterministic Turing machine, working in a polynomial time. As stated in the Introduction, the SN P systems we will describe will solve all the instances of `QSAT` and `Q3SAT` of a given size, just like in the uniform setting. However, such systems will have an exponential size. Since a deterministic Turing machine cannot produce (the description of) an exponential size object in a polynomial time, we will say that our solutions are *exp-uniform*.

3 An exp-Uniform Solution to `QSAT`

In this section we build an exp-uniform family $\{\Pi_{QSAT}(2n,m)\}_{n,m\in\mathbb{N}}$ of SN P systems such that for all $n,m\in\mathbb{N}$ the system $\Pi_{QSAT}(2n,m)$ solves all the instances of $QSAT(2n,m)$ in a polynomial number of steps with respect to n and m, in a deterministic way.

The instances of $QSAT(2n,m)$ to be given as input to the system $\Pi_{QSAT}(2n,m)$ are encoded as sequences of virtual symbols, as follows. For any given instance $\gamma_{2n,m} = \exists x_1 \forall x_2 \ldots \exists x_{2n-1} \forall x_{2n}(C_1 \wedge C_2 \wedge \ldots \wedge C_m)$ of $QSAT(2n,m)$, let $code(\gamma_{2n,m}) = \alpha_{11}\alpha_{12}\cdots\alpha_{12n}\alpha_{21}\alpha_{22}\cdots\alpha_{22n}\cdots\alpha_{m1}\alpha_{m2}\cdots\alpha_{m2n}$, where each α_{ij}, for $1\le i\le m$ and $1\le j\le 2n$, is a *spike variable* whose value is an amount of spikes (a virtual symbol), assigned as follows:

$$\alpha_{ij} = \begin{cases} a, & \text{if } x_j \text{ occurs in } C_i; \\ a^2, & \text{if } \neg x_j \text{ occurs in } C_i; \\ \lambda, & \text{otherwise.} \end{cases}$$

In this way the sequence $\alpha_{i1}\alpha_{i2}\cdots\alpha_{i2n}$ of spike variables represents the clause C_i, and the representation of $\gamma_{2n,m}$ is just the concatenation of the representations of the single clauses. As an example, the representation of $\gamma_{2,2} = \exists x_1 \forall x_2[(x_1 \vee \neg x_2) \wedge (\neg x_2)]$ is $aa^2\lambda a^2$. The set of all the encoding sequences of all possible instances of $QSAT(2n,m)$ is denoted by $code(QSAT(2n,m))$. For instance, $QSAT(2,1)$ contains the following nine formulas (the existential and the universal quantifiers are here omitted for the sake of readability): $\gamma^1_{2,1}$ = no variable appears in the clause, $\gamma^2_{2,1} = x_2$, $\gamma^3_{2,1} = \neg x_2$, $\gamma^4_{2,1} = x_1$, $\gamma^5_{2,1} = x_1 \vee x_2$, $\gamma^6_{2,1} = x_1 \vee \neg x_2$, $\gamma^7_{2,1} = \neg x_1$, $\gamma^8_{2,1} = \neg x_1 \vee x_2$, $\gamma^9_{2,1} = \neg x_1 \vee \neg x_2$. Therefore, $code(QSAT(2,1)) = \{\lambda\lambda, \lambda a, \lambda a^2, a\lambda, aa, aa^2, a^2\lambda, a^2a, a^2a^2\}$.

The structure of the pre-computed SN P system that solves all possible instances of $QSAT(2n,m)$ is illustrated in a schematic way in Figures 1 and 2. The system is a structure of the form $\Pi^{(2n,m)}_{QSAT} = (\Pi_{QSAT}(2n,m), code(QSAT(2n,m)))$ with:

- $\Pi_{QSAT}(2n,m) = (O, \mu, in, out)$, where:
  1. $O = \{a\}$ is the singleton alphabet;
  2. $\mu = (H, \bigcup_{i\in H}\{m_i\}, \bigcup_{j\in H} R_j, syn)$ is the structure of the SN P system, where:
  - $H = H_0 \cup H_1 \cup H_2 \cup H_3$ is a finite set of neuron labels, with
    $H_0 = \{in, out, d\} \cup \{d_i \mid 0 \le i \le 2n\}$,
    $H_1 = \{Cx_i, Cx_i1, Cx_i0 \mid 1 \le i \le 2n\}$,
    $H_2 = \{bin, Cbin \mid bin \in \{0,1\}^{2n}\}$,
    $H_3 = \{y_1y_2\ldots y_{2n-1}y_{2n} \mid y_i \in \{0,1\}$ when i is odd and $y_i \in \{0,1,x_i\}$ when i is even, and there exists at least one $k \in \{1,2,\ldots,n\}$ such that $y_{2k} = x_{2k}\}$ (we recall that even values of i correspond to universally quantified variables).
    All the neurons are injectively labeled with elements from H;
  - $m_{d_0} = 2$, $m_d = 1$ and $m_i = 0$ ($i \in H$, $i \ne d_0, d$) are the numbers of spikes that occur in the initial configuration of the system;
  - R_k, $k \in H$, is a finite set of rules associated with neuron σ_k, where:
    $R_{in} = \{a \to a, a^2 \to a^2\}$, $R_d = \{a \to a; 2mn + n + 6\}$,
    $R_{d_i} = \{a^2 \to a^2\}$, for $i \in \{0,1,\ldots,2n-1\}$, and $R_{d_{2n}} = \{a^2 \to a^2, a^3 \to \lambda\}$,
    $R_{Cx_i} = \{a \to \lambda, a^2 \to \lambda, a^3 \to a^3; 2n-i, a^4 \to a^4; 2n-i\}$, for $i \in \{1,2,\ldots,2n\}$,
    $R_{Cx_i1} = \{a^3 \to a^2, a^4 \to \lambda\}$ and $R_{Cx_i0} = \{a^3 \to \lambda, a^4 \to a^2\}$, for $i \in \{1,2,\ldots,2n\}$,
    $R_{Cbin} = \{(a^2)^+/a \to a\} \cup \{a^{2k-1} \to \lambda \mid k = 1,2,\ldots,2n\}$, for $bin \in \{0,1\}^{2n}$,
    $R_{bin} = \{a^m \to a\}$, for $bin \in \{0,1\}^{2n}$,
    $R_{str} = \{a^2 \to a\}$, where $str \in H_3$ and there exists at least one $i \in \{1,2,\ldots,n\}$ such that $str|_{2i} \ne x_{2i}$,
    $R_{str'} = \{a^2 \to a^2\}$, where $str' \in H_3$ and $str'|_{2k} = x_{2k}$, for all $1 \le k \le n$,
    $R_{out} = \{(a^2)^+/a \to a\}$;
  - syn is the set of all the synapses between the neurons. The following synapses are used in the input module (see Figure 1): (in, Cx_i), (d_{i-1}, d_i), (d_i, Cx_i), (Cx_i, Cx_i1) and (Cx_i, Cx_i0), for all $1 \le i \le 2n$, as well as (d_{2n}, d_1) and (d, d_{2n}).
    The synapses connecting the other neurons are illustrated in Figure 2:

$(Cx_ij, Cbin)$, where $bin \in \{0,1\}^{2n}$ and $bin|_i = j, 1 \le i \le 2n$, $j \in \{0,1\}$,
$(Cbin, bin)$, where $bin \in \{0,1\}^{2n}$,
(bin, str), where $bin \in \{0,1\}^{2n}$, $str \in H_3$, and $str = (bin|_i \leftarrow x_2)$,
(str_{j_1}, str_{j_2}), where $str_{j_1}, str_{j_2} \in H_3$ and $str_{j_2} = (str_{j_1}|_{2i} \leftarrow x_{2i})$, $2 \le i \le n$,
(str, out), where $str \in H_3$ and $str|_{2k} = x_{2k}$, for all $1 \le k \le n$;

3. in, out indicate the *input* and *output* neurons, respectively;

- $code(QSAT(2n,m))$ is the set of all the encoding sequences for all the possible instances of $QSAT(2n,m)$, as defined above.

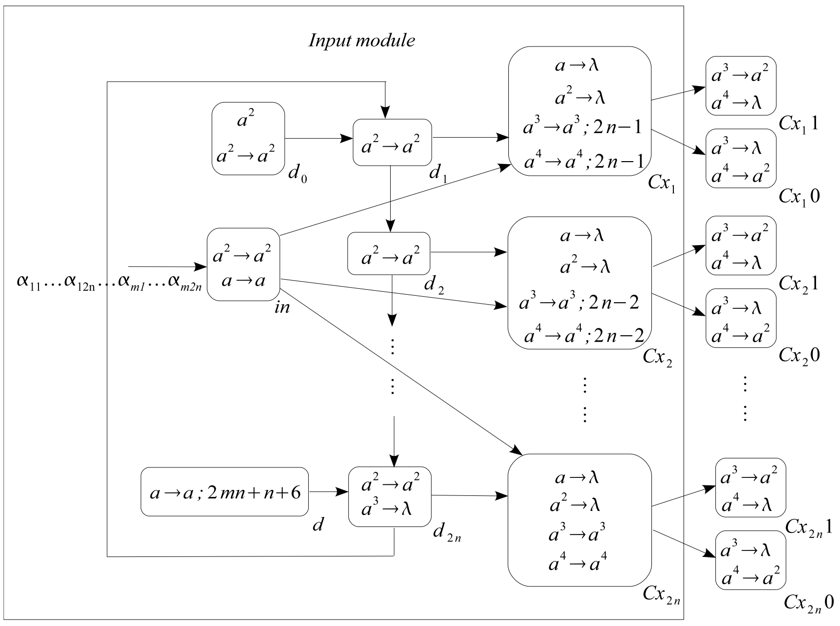


Fig. 1. The input module of $\Pi_{QSAT}(2n,m)$

The system is composed of four modules: *input, satisfiability checking, quantifier checking*, and *output*. To simplify the description of the system and its working, the neurons in the system are arranged in $n+7$ layers in Figures 1 and 2. The input module has three layers (the first layer contains three neurons σ_{d_0}, σ_d and σ_{in}; the second layer contains $2n$ neurons σ_{d_i}, $1 \le i \le 2n$; the third layer contains $2n$ neurons σ_{Cx_i}, $1 \le i \le 2n$). The satisfiability checking module has also three layers (the fourth layer contains $4n$ neurons σ_{Cx_ij}, $1 \le i \le 2n$, $j = 0, 1$; the fifth layer contains 2^{2n} neurons σ_{Cbin}, $bin \in \{0,1\}^{2n}$; the sixth layer contains 2^{2n} neurons σ_{bin}, $bin \in \{0,1\}^{2n}$). The quantifier checking module is composed of n layers, from the 7th to the $(n+6)$th layer, where a total of $2^{2n} - 2^n$ neurons are used.

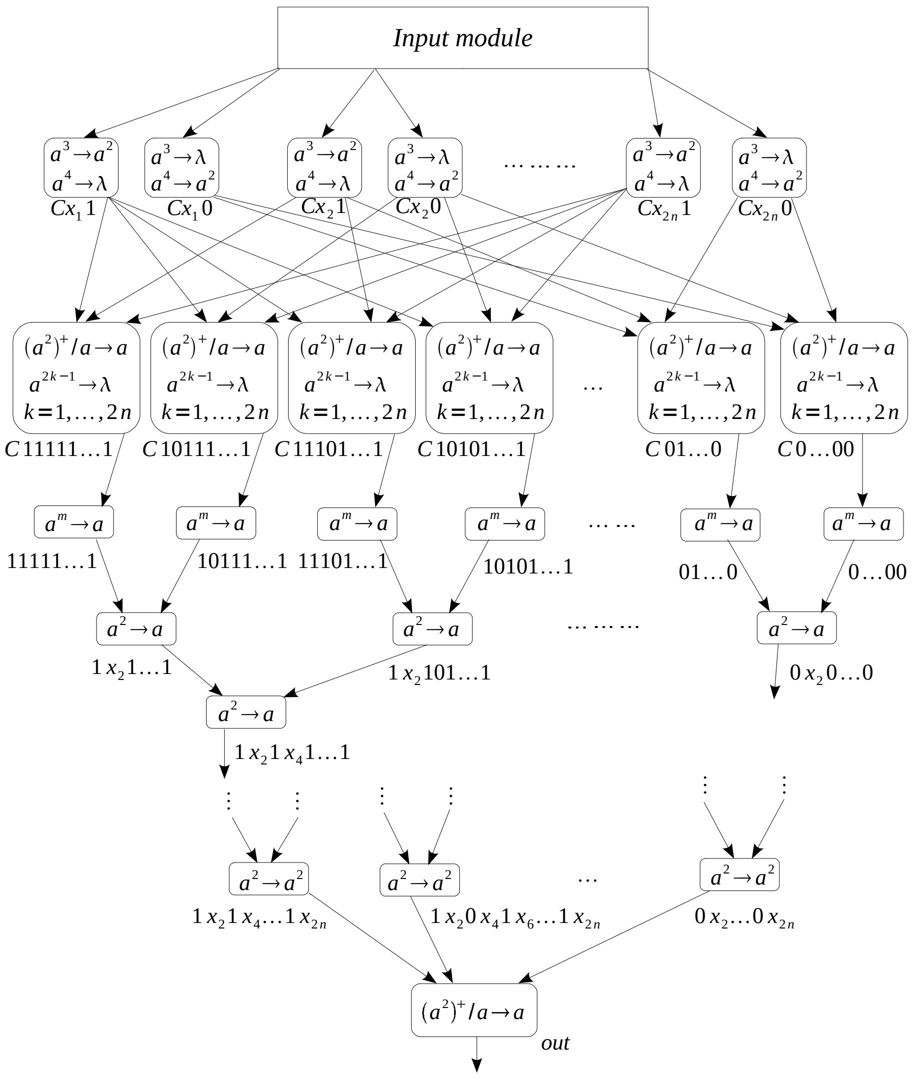

Input module
$a^3 \to a^2$ $a^4 \to \lambda$ $Cx_1 1$
$a^3 \to \lambda$ $a^4 \to a^2$ $Cx_1 0$
$a^3 \to a^2$ $a^4 \to \lambda$ $Cx_2 1$
$a^3 \to \lambda$ $a^4 \to a^2$ $Cx_2 0$
$a^3 \to a^2$ $a^4 \to \lambda$ $Cx_{2n} 1$
$a^3 \to \lambda$ $a^4 \to a^2$ $Cx_{2n} 0$
$(a^2)^+/a \to a$ $a^{2k-1} \to \lambda$ $k=1,\dots,2n$
$C11111\dots1$
$C10111\dots1$
$C11101\dots1$
$C10101\dots1$
$C01\dots0$
$C0\dots00$
$a^m \to a$
$11111\dots1$
$10111\dots1$
$11101\dots1$
$10101\dots1$
$01\dots0$
$0\dots00$
$a^2 \to a$
$1x_21\dots1$
$1x_2101\dots1$
$0x_20\dots0$
$1x_21x_41\dots1$
$a^2 \to a^2$
$1x_21x_4\dots1x_{2n}$
$1x_20x_41x_6\dots1x_{2n}$
$0x_2\dots0x_{2n}$
$(a^2)^+/a \to a$
out

Fig. 2. Structure of the SN P system $\Pi_{QSAT}(2n, m)$

The output module only contains one neuron σ_{out}, which appears in the last layer. In what follows we provide a more detailed description of each module, as well as its working when solving a given instance $\gamma_{2n,m} \in QSAT(2n, m)$.

- *Input*: The input module consists of $4n + 3$ neurons, contained in the layers 1 – 3 as illustrated in Figure 1; σ_{in} is the unique input neuron. The values of the spike variables of the encoding sequence $code(\gamma_{2n,m})$ are introduced into σ_{in}

one by one, starting from the beginning of the computation. At the first step of the computation, the value of the first spike variable α_{11}, which is the virtual symbol that represents the occurrence of the first variable in the first clause, enters into neuron σ_{in}; in the meanwhile, neuron σ_{d_1} receives two auxiliary spikes from neuron σ_{d_0}. At this step, the firing rule in neuron σ_d is applied; as a result, neuron σ_d will send one spike to neuron $\sigma_{d_{2n}}$ after $2mn+n+6$ steps (this is done in order to halt the computation after the answer to the instance given as input has been determined). In the next step, the value of the spike variable α_{11} is replicated and sent to neurons σ_{Cx_i}, for all $i \in \{1,2,\dots,2n\}$; the two auxiliary spikes contained in σ_{d_1} are also sent to neurons σ_{Cx_1} and σ_{d_2}. Hence, neuron σ_{Cx_1} will contain 2, 3 or 4 spikes: if x_1 occurs in C_1, then neuron σ_{Cx_1} collects 3 spikes; if $\neg x_1$ occurs in C_1, then it collects 4 spikes; if neither x_1 nor $\neg x_1$ occur in C_1, then it collects two spikes. Moreover, if neuron σ_{Cx_1} has received 3 or 4 spikes, then it will be closed for $2n-1$ steps, according to the delay associated with the rules in it; on the other hand, if 2 spikes are received, then they are deleted and the neuron remains open. At the third step, the value of the second spike variable α_{12} from neuron σ_{in} is distributed to neurons σ_{Cx_i}, $2 \le i \le 2n$, where the spikes corresponding to α_{11} are deleted. At the same time, the two auxiliary spikes are duplicated and one copy of them enters into neurons σ_{Cx_2} and σ_{d_3}, respectively. The neuron σ_{Cx_2} will be closed for $2n-2$ steps only if it contains 3 or 4 spikes, which means that this neuron will not receive any spike during this period. In neurons σ_{Cx_i}, $3 \le i \le 2n$, the spikes represented by α_{12} are forgotten in the next step.
In this way, the values of the spike variables are introduced and delayed in the corresponding neurons until the value of the spike variable α_{12n} of the first clause and the two auxiliary spikes enter together into neuron $\sigma_{Cx_{2n}}$ at step $2n+1$. At that moment, the representation of the first clause of $\gamma_{2n,m}$ has been entirely introduced in the system, and the second clause starts to enter into the input module. The entire sequence $code(\gamma_{2n,m})$ is introduced in the system in $2mn+1$ steps.

- *Satisfiability checking*: Once all the values of spike variables α_{1i} ($1 \le i \le 2n$), representing the first clause, have appeared in their corresponding neurons σ_{Cx_i} in layer 3, together with a copy of the two auxiliary spikes, all the spikes contained in σ_{Cx_i} are duplicated and sent simultaneously to the pair of neurons σ_{Cx_i1} and σ_{Cx_i0}, for $i \in \{1,2,\dots,2n\}$, at the $(2n+2)$nd computation step. In this way, each neuron σ_{Cx_i1} and σ_{Cx_i0} receives 3 or 4 spikes when x_i or $\neg x_i$ occurs in C_1, respectively, whereas it receives no spikes when neither x_i or $\neg x_i$ occurs in C_1. In general, if neuron σ_{Cx_i1} receives 3 spikes, then the literal x_i occurs in the current clause (say C_j), and thus the clause is satisfied by all those assignments in which x_i is true. Neuron σ_{Cx_i0} will also receive 3 spikes, but it will delete them during the next computation step. On the other hand, if neuron σ_{Cx_i1} receives 4 spikes, then the literal $\neg x_i$ occurs in C_j, and the clause is satisfied by those assignments in which x_i is false. Since neuron σ_{Cx_i1} is designed to process the case in which x_i occurs in C_j, it will

delete its 4 spikes. However, also neuron σ_{Cx_i0} will have received 4 spikes, and this time it will send two spikes to those neurons which are bijectively associated with the assignments for which x_i is false. Note that all possible 2^{2n} truth assignments to $x_1, x_2, \ldots, x_{2n}$ are represented by the neurons' labels $Cbin$ in layer 5, where bin is generated from $\{0,1\}^{2n}$; precisely, we read bin, where $bin|_i = j$, $j \in \{0,1\}$, as a truth assignment whose value for x_i is j. In the next step, those neurons σ_{Cbin} that received at least two spikes send one spike to the corresponding neurons σ_{bin} in layer 6 (the rest of the spikes will be forgotten), with the meaning that the clause is satisfied by the assignment bin. This occurs in the $(2n+4)$th computation step. Thus, the check for the satisfiability of the first clause has been performed; in a similar way, the check for the remaining clauses can proceed. All the clauses can thus be checked to see whether there exist assignments that satisfy all of them.
If there exist some assignments that satisfy all the clauses of $\gamma_{2n,m}$, then the neurons labeled with the values of $bin \in \{0,1\}^{2n}$ that correspond to these assignments succeed to accumulate m spikes. Thus, the rule $a^m \rightarrow a$ can be applied in these neurons. The satisfiability checking module completes its process in $2mn+5$ steps.

- *Quantifier checking*: The universal and existential quantifiers of the fully quantified formula $\gamma_{2n,m}$ are checked starting from step $2mn+6$.
Since all the instances of $QSAT(2n,m)$ are in the normal form, it is not difficult to see that we need only to check the universal quantifiers associated to even-numbered variables $(x_2, x_4, \ldots)$. These universal quantifiers are checked one by one, and thus the quantifier checking module needs n steps to complete its process. The module starts by checking the universal quantifier associated with x_2, which is performed as follows. For any two binary sequences bin_1 and bin_2 with $bin_1|_i = bin_2|_i$ for all $i \neq 2$ and $bin_1|_2 = 1$, $bin_2|_2 = 0$, if both neurons σ_{bin_1} and σ_{bin_2} contain m spikes, then neuron σ_{str} will receive two spikes from them at step $2mn+5$, where $str = (bin_1|_2 \leftarrow x_2)$. This implies that, no matter whether $x_2 = 1$ or $x_2 = 0$, if we assign each variable x_j with the value $str|_j$, $j \neq 2$, $1 \leq j \leq 2n$, then all the clauses of $\gamma_{2n,m}$ are satisfied. As shown in Figure 2, in this way the system can check, in the 7th layer, the satisfiability of the universal quantifier associated to variable x_2. The system is then ready to check the universal quantifier associated with variable x_4, which is performed in a similar way as follows. For any two sequences str_1 and str_2 with $str_1|_i = str_2|_i \in \{0,1\}$, for all $i \neq 2,4$, and $str_1|_2 = str_2|_2 = x_2$, $str_1|_4 = 1$, $str_2|_4 = 0$, if both neurons σ_{str_1} and σ_{str_2} contain two spikes, then σ_{str_3} will receive two spikes, where str_3 is obtained by replacing the fourth symbol of str_1 with x_4 (i.e., $str_3 = (str_1|_4 \leftarrow x_4)$). In this way, we check the (simultaneous) satisfiability of the universal quantifiers associated to the two variables x_2 and x_4. Similarly, the system can check the satisfiability of the universal quantifier associated with variable x_6 by using the neurons in the ninth layer. Therefore, after n steps (in the $(n+6)$th layer) the system has checked the satisfiability of all the universal quantifiers associated with

the variables $x_2, x_4, \ldots, x_{2n}$. If a neuron σ_{str} accumulates two spikes, where $str|_{2k} = x_{2k}$ for all $1 \leq k \leq n$, then we conclude that this assignment not only makes all the clauses satisfied, but also satifies all the quantifiers. Therefore, the neurons which accumulate two spikes will send two spikes to the output neuron, thus indicating that the instance of the problem given as input is positive.

- *Output*: From the construction of the system, it is not difficult to see that the output neuron sends exactly one spike to the environment at the $(2mn+n+6)$th computation step if and only if $\gamma_{2n,m}$ is *true*. At this moment, neuron σ_d will also send a spike to the auxiliary neuron $\sigma_{d_{2n}}$ (the rule is applied in the first computation step). This spike stays in neuron $\sigma_{d_{2n}}$ until two further spikes arrive from neuron $\sigma_{d_{2n-1}}$; when this happens, all three spikes are forgotten by using the rule $a^3 \to \lambda$ in neuron $\sigma_{d_{2n}}$. Hence, the system eventually halts after a few steps since the output neuron fires.

Note that the number m of clauses appearing in a $QSAT(2n, m)$ problem may be very large (e.g., exponential) with respect to n: every variable can occur negated or non-negated in a clause, or not occur at all, and hence the number of all possible clauses is 3^{2n}. This means that the running time of the system may be exponential with respect to n, and also the rules $a^m \to a$ in some neurons of the system are required to work on a possibly very large number of spikes. As we will see in the next section, these "problems" (if one considers them as problems) do not occur when considering `Q3SAT`, since each clause in the formula contains exactly three literals, and thus the number of possible clauses is at most cubic in n.

3.1 An example

Let us present a simple example which shows how the system solves the instances of $QSAT(2n, m)$, for specified values of n and m, in an exp-uniform way. Let us consider the following fully quantified propositional formula, which has two variables and two clauses (i.e., $n = 1, m = 2$):

$$\gamma_{2,2} = \exists x_1 \forall x_2 (x_1 \vee \neg x_2) \wedge x_1$$

Such a formula is encoded as the sequence $code(\gamma_{2,2}) = aa^2a\lambda$ of virtual symbols.

The structure of the SN P system which is used to solve all the instances of $QSAT(2, 2)$ is pre-computed as illustrated in Figure 3. It is composed of 22 neurons; its computations are performed as follows.

Input: Initially, neuron σ_{d_0} contains two spikes and neuron σ_d contains one spike, whereas the other neurons in the system contain no spikes. The computation starts by inserting the leftmost symbol of the encoding sequence $code(\gamma_{2,2}) = aa^2a\lambda$ into the input neuron σ_{in}. When this symbol (a) enters into the system, neuron σ_{d_0} emits its two spikes to neuron σ_{d_1}. At this moment, the rule occurring in neuron σ_d also fires; as a result, it will send one spike to neuron σ_{d_2} after 11 steps. At the next step, two spikes from σ_{d_1} are sent to neurons σ_{d_2} and σ_{Cx_1},

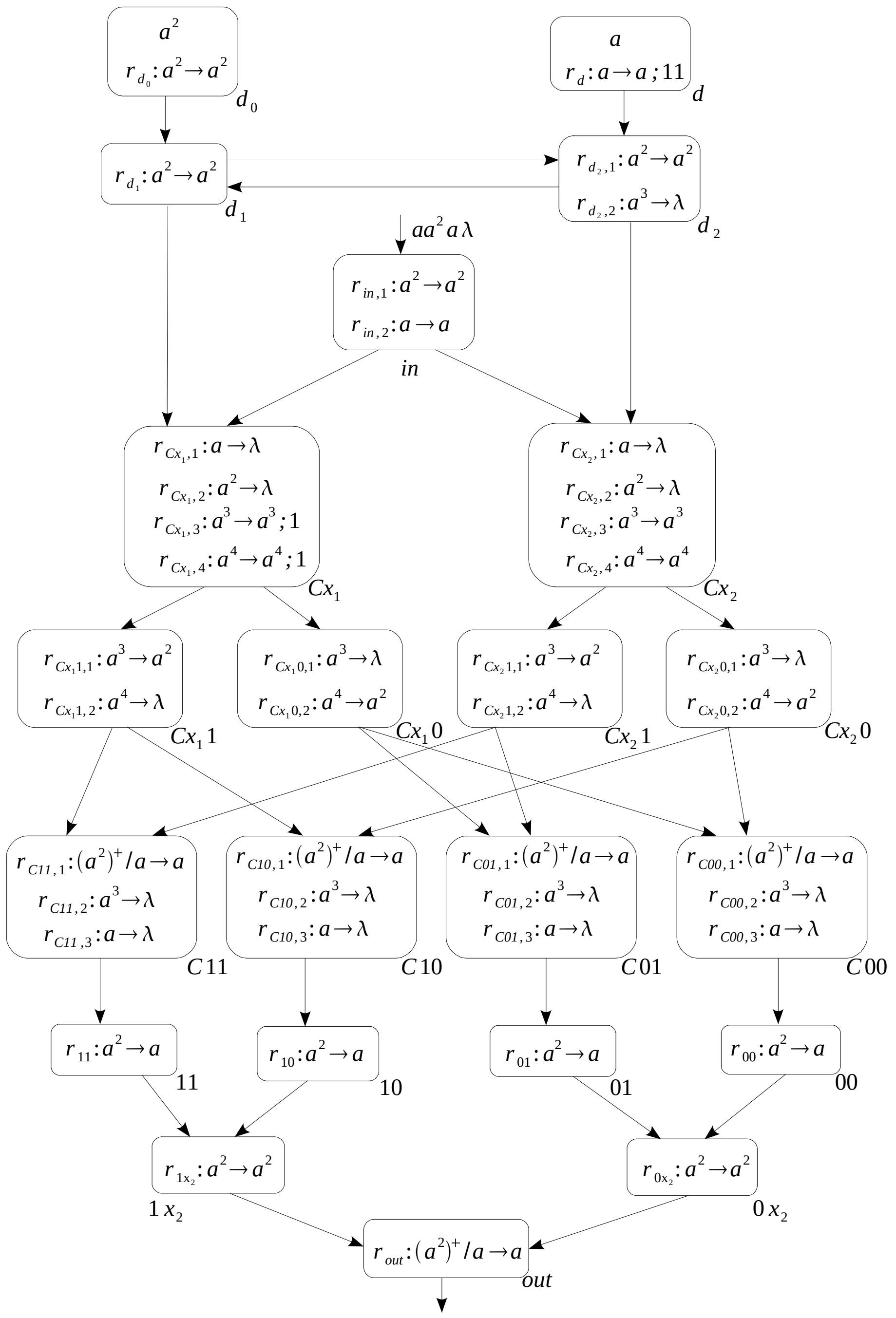

a^2
$r_{d_0}: a^2 \to a^2$
d_0
a
$r_d: a \to a; 11$
d
$r_{d_1}: a^2 \to a^2$
d_1
$r_{d_2,1}: a^2 \to a^2$
$r_{d_2,2}: a^3 \to \lambda$
d_2
$aa^2a\lambda$
$r_{in,1}: a^2 \to a^2$
$r_{in,2}: a \to a$
in
$r_{Cx_1,1}: a \to \lambda$
$r_{Cx_1,2}: a^2 \to \lambda$
$r_{Cx_1,3}: a^3 \to a^3; 1$
$r_{Cx_1,4}: a^4 \to a^4; 1$
Cx_1
$r_{Cx_2,1}: a \to \lambda$
$r_{Cx_2,2}: a^2 \to \lambda$
$r_{Cx_2,3}: a^3 \to a^3$
$r_{Cx_2,4}: a^4 \to a^4$
Cx_2
$r_{Cx_11,1}: a^3 \to a^2$
$r_{Cx_11,2}: a^4 \to \lambda$
$Cx_1 1$
$r_{Cx_10,1}: a^3 \to \lambda$
$r_{Cx_10,2}: a^4 \to a^2$
$Cx_1 0$
$r_{Cx_21,1}: a^3 \to a^2$
$r_{Cx_21,2}: a^4 \to \lambda$
$Cx_2 1$
$r_{Cx_20,1}: a^3 \to \lambda$
$r_{Cx_20,2}: a^4 \to a^2$
$Cx_2 0$
$r_{C11,1}: (a^2)^+/a \to a$
$r_{C11,2}: a^3 \to \lambda$
$r_{C11,3}: a \to \lambda$
$C11$
$r_{C10,1}: (a^2)^+/a \to a$
$r_{C10,2}: a^3 \to \lambda$
$r_{C10,3}: a \to \lambda$
$C10$
$r_{C01,1}: (a^2)^+/a \to a$
$r_{C01,2}: a^3 \to \lambda$
$r_{C01,3}: a \to \lambda$
$C01$
$r_{C00,1}: (a^2)^+/a \to a$
$r_{C00,2}: a^3 \to \lambda$
$r_{C00,3}: a \to \lambda$
$C00$
$r_{11}: a^2 \to a$
11
$r_{10}: a^2 \to a$
10
$r_{01}: a^2 \to a$
01
$r_{00}: a^2 \to a$
00
$r_{1x_2}: a^2 \to a^2$
$1x_2$
$r_{0x_2}: a^2 \to a^2$
$0x_2$
$r_{out}: (a^2)^+/a \to a$
out

Fig. 3. The pre-computed structure of the SN P system $\Pi_{QSAT}(2, 2)$

while the symbol a is sent by σ_{in} to neurons σ_{Cx_1} and σ_{Cx_2}. At the same time, the value a^2 of the second spike variable α_{12} of $code(\gamma_{2,2})$ is introduced into σ_{in}.

Neuron σ_{Cx_1} has accumulated three spikes and thus the rule $a^3 \to a^2; 1$ can be applied in it, while the spike in neuron σ_{Cx_2} is forgotten by using the rule $a \to \lambda$ at the third computation step. Simultaneously, the value a^2 of the second spike variable α_{12} from σ_{in} and two spikes from σ_{d_2} enter together into σ_{Cx_2}; neuron σ_{Cx_1} does not receive any spike, as it has been closed for this step. Thus, at the third computation step the representation of the first clause aa^2 has appeared in the input module. At this step, the value a of the first spike variable α_{21} of the second clause also enters the input neuron, while neuron σ_{d_1} receives two spikes again, which triggers the introduction of the representation of the second clause $(a\lambda)$ in the input module.

Satisfiability checking: Now, neuron σ_{Cx_1} is open and fires, sending three spikes to neurons $\sigma_{Cx_1 1}$ and $\sigma_{Cx_1 0}$. The three spikes in neuron $\sigma_{Cx_1 1}$ denote that literal x_1 occurs in the current clause (C_1), and thus the clause is satisfied by all those assignments in which $x_1 = 1$. And, in fact, $\sigma_{Cx_1 1}$ sends two spikes to neurons σ_{C11} and σ_{C10}, to indicate that the first clause is satisfied by the assignments *bin* whose first value is 1. The three spikes in neuron $\sigma_{Cx_1 0}$ denote that the current clause (C_1) does not contain the literal $\neg x_1$. Hence, no spike is emitted from neuron $\sigma_{Cx_1 0}$; its three spikes are forgotten instead. Similarly, the presence of four spikes in neuron $\sigma_{Cx_2 1}$ (resp., in $\sigma_{Cx_2 0}$) denotes the fact that literal x_2 (resp., $\neg x_2$) does not occur (resp., occurs) in clause C_1. Hence, the spikes in neuron $\sigma_{Cx_2 1}$ are forgotten, whereas neuron $\sigma_{Cx_2 0}$ sends two spikes to neurons σ_{C10} and σ_{C00} to denote that clause C_1 is satisfied by those assignments in which $x_2 = 0$.

At step 5, the configuration of the system is as follows. Three spikes occur in neuron σ_{Cx_1}, since literal x_1 occurs in the second clause; no spikes occur in σ_{Cx_2}, as the clause does not contain variable x_2; the two auxiliary spikes appear alternately in neurons σ_{d_1} and σ_{d_2} in the input module. At the same time, neurons σ_{C11} and σ_{C00} contain two spikes each, whereas neuron σ_{C10} contains four spikes.

In the next step, neuron σ_{Cx_1} sends three spikes to its two target neurons $\sigma_{Cx_1 1}$ and $\sigma_{Cx_1 0}$, while each of the neurons σ_{C11}, σ_{C00} and σ_{C10} sends one spike towards their related neurons in the next layer, thus confirming that the first clause is satisfied by the corresponding assignments. The rest of spikes in these neurons will be forgotten in the following step. At step 7, neuron $\sigma_{Cx_1 1}$ sends two spikes to neurons σ_{C11} and σ_{C10} by using the rule $a^3 \to a^2$, whereas the three spikes in neuron $\sigma_{Cx_1 0}$ are forgotten. Note that the spike in neurons σ_{11}, σ_{10} and σ_{00}, which is received from their related neurons σ_{C11}, σ_{C00} and σ_{C10}, remains unused until one more spike is received. At step 8, neuron σ_{C11} sends one spike to its related neuron σ_{11} and neuron σ_{C10} sends one spike to its related neuron σ_{10}, while the rest of spikes are forgotten. In this way, neurons σ_{11} and σ_{10} succeed to accumulate a sufficient number (two) of spikes to fire. On the other hand, neuron σ_{00} fails to accumulate the desired number of spikes (it obtains only one spike), thus the rule in it cannot be activated.

Quantifier checking: We now pass to the module which checks the universal and existential quantifiers associated to the variables. At step 9 neuron σ_{1x_2} receives two spikes, one from σ_{11} and another one from σ_{10}, which means that the formula $\gamma_{2,2}$ is satisfied when $x_1 = 1$, no matter whether $x_2 = 0$ or $x_2 = 1$. On the other hand, neuron σ_{0x_2} does not contain any spike. At step 10 the rule $a^2 \to a^2$ is applied in neuron σ_{1x_2}, making neuron σ_{out} receive two spikes.

Output: The rule occurring in neuron σ_{out} is activated and one spike is sent to the environment, indicating that the instance of the problem given as input is positive (that is, $\gamma_{2,2}$ is *true*). At this step, as neuron σ_{d_2} will receive a "trap" spike from neuron σ_d, the two auxiliary spikes circulating in the input module are deleted as soon as they arrive in it, because of the rule $a^3 \to \lambda$. Thus, the system halts after 13 computation steps since it has been started.

In order to illustrate how the system from Figure 3 evolves in detail, its transition diagram (generated with the software tool developed in [6]) is also given in Figure 4. In this figure, $\langle r_1/t_1, \ldots, r_{21}/t_{21}, r_{22}/t_{22}\rangle$ is the configuration in which neurons σ_d, σ_{d_0}, σ_{d_1}, σ_{d_2}, σ_{in}, σ_{Cx_1}, σ_{Cx_2}, σ_{Cx_11}, σ_{Cx_10}, σ_{Cx_21}, σ_{Cx_20}, σ_{C11}, σ_{C10}, σ_{C01}, σ_{C00}, σ_{11}, σ_{10}, σ_{01}, σ_{00}, σ_{1x_2}, σ_{0x_2} and σ_{out} contain $r_1, r_2, \ldots, r_{22}$ spikes, respectively, and will be open after $t_1, t_2, \ldots, t_{22}$ steps, respectively. Between two configurations we draw an arrow if and only if a direct transition is possible between them. For simplicity, the rules are indicated only when they are used, while unused rules are omitted. When neuron σ_k spikes after being closed for $s \geq 0$ steps, we write $r_{k,s}$. We omit to indicate s when it is zero. Finally, we have highlighted the firing of σ_{out} by writing r_{out} in bold.

4 Solving Q3SAT

As stated in section 2, the instances of Q3SAT are defined like those of QSAT, with the additional constraint that each clause contains exactly three literals. In what follows, by $Q3SAT(2n)$ we will denote the set of all instances of Q3SAT which can be built using $2n$ Boolean variables $x_1, x_2, \ldots, x_{2n}$, with the following three restrictions: (1) no repetitions of the same literal may occur in any clause, (2) no clause contains both the literals x_s and $\neg x_s$, for any $s \in \{1, 2, \ldots, 2n\}$, and (3) the instance is expressed in the normal form described in section 2 (all even-numbered and odd-numbered variables are universally and existentially quantified, respectively).

As stated in the previous section, the number m of possible clauses that may appear in a formula $\gamma_{n,m} \in QSAT(n, m)$ is exponential with respect to n. On the contrary, the number of possible 3-clauses which can be built using $2n$ Boolean variables is $4n \cdot (4n-2) \cdot (4n-4) = \Theta(n^3)$, a polynomial quantity with respect to n. This quantity, that we denote by $Cl(2n)$, is obtained by looking at a 3-clause as a triple, and observing that each component of the triple may contain one of the $4n$ possible literals, with the constraints that we do not allow the repetition of literals in the clauses, or the use of the same variable two or three times in a clause.

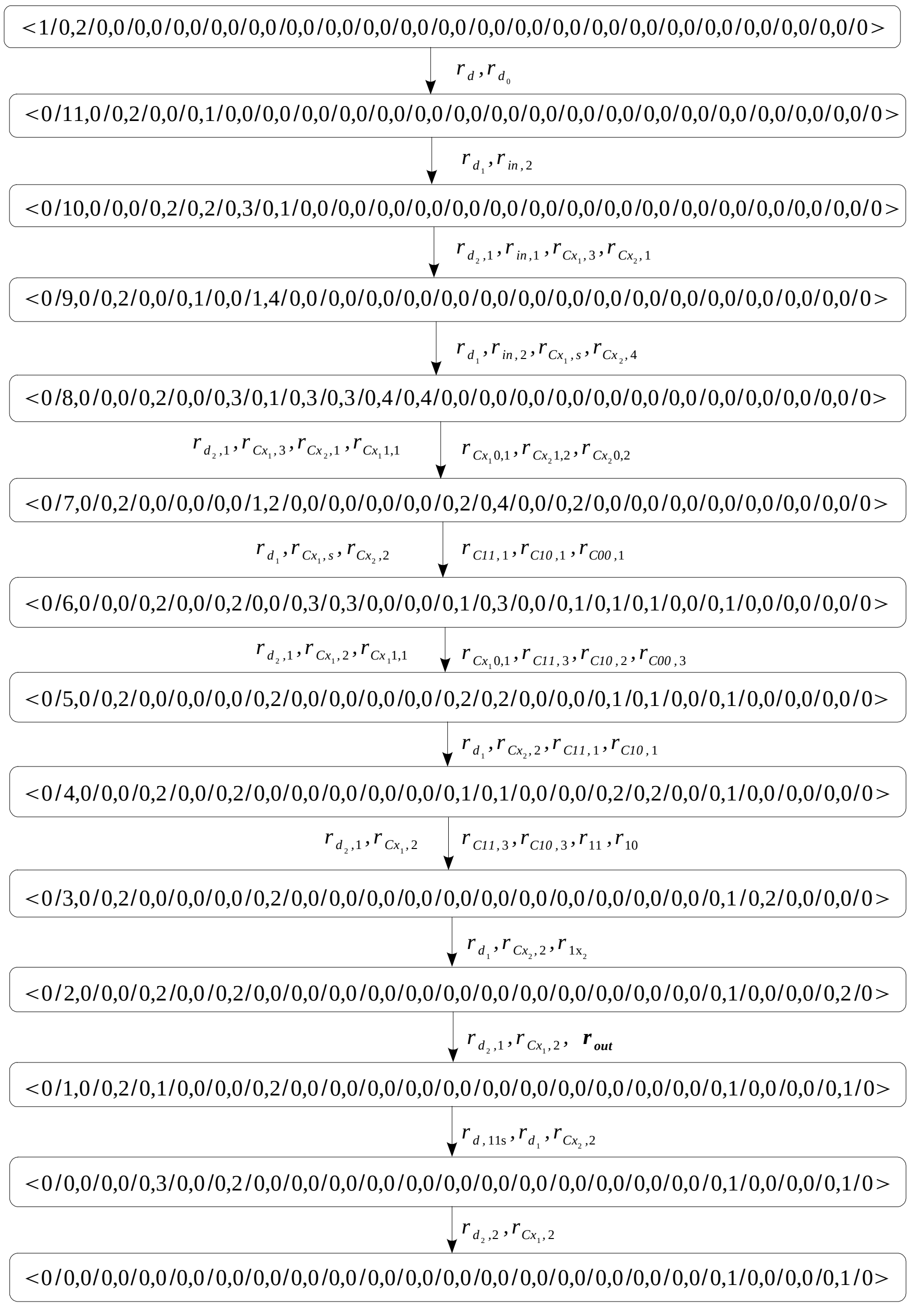


Fig. 4. The transition diagram of the system illustrated in Figure 3

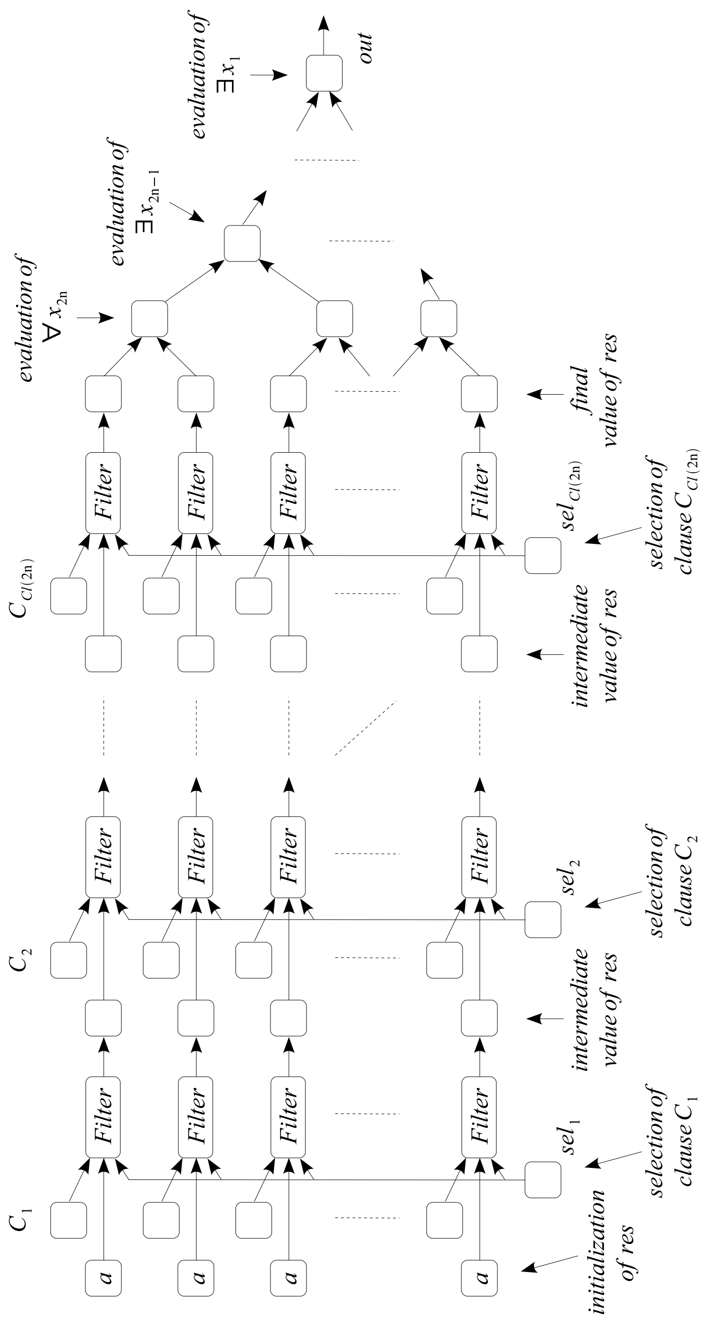

evaluation of $\forall x_{2n}$
evaluation of $\exists x_{2n-1}$
evaluation of $\exists x_1$
out
C_1
C_2
$C_{Cl(2n)}$
a
a
a
a
Filter
sel_1
sel_2
$sel_{Cl(2n)}$
initialization of res
selection of clause C_1
intermediate value of res
selection of clause C_2
intermediate value of res
selection of clause $C_{Cl(2n)}$
final value of res

Fig. 5. Sketch of a deterministic SN P system that solves all possible instances of $Q3SAT(2n)$

Figure 5 outlines an SN P system which can be used to solve any instance $\gamma_{2n,m}$ of $Q3SAT(2n)$. The input to this system is once again the instance of Q3SAT we want to solve, but this time such an instance is given by specifying — among all the possible clauses that can be built using n Boolean variables — which clauses occur in the instance. The selection is performed by putting (in parallel, in the initial configuration of the system) one spike in each of the input neurons $sel_1, sel_2, \ldots, sel_{Cl(2n)}$ that correspond to the selected clauses.

The system is a simple modification of the one used in [8] to uniformly solve all the instances of the **NP**-complete problem 3-SAT of a given size. To see how it works, let us consider the family $\{M^{(2n)}\}_{n \in \mathbb{N}}$ of Boolean matrices, where $M^{(2n)}$ has 2^{2n} rows — one for each possible assignment to the variables $x_1, x_2, \ldots, x_{2n}$ — and one column for each possible 3-clause that can be built using the same variables. As stated above, the number of columns is $Cl(2n) \in \Theta(n^3)$, a polynomial quantity in n. In order to make the construction of the matrix $M^{(2n)}$ as regular as possible, we could choose to list all the 3-clauses in a predefined order; however, our result is independent of any such particular ordering, and hence we will not bother further with this detail. For every $j \in \{1, 2, 3, \ldots, 2^{2n}\}$ and $i \in \{1, 2, \ldots, Cl(2n)\}$, the

x_1	x_2	x_3	x_4	$\cdots$	$x_1 \vee x_2 \vee \neg x_4$	$\cdots$	$\neg x_1 \vee \neg x_2 \vee x_3$	$\cdots$
0	0	0	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	0	0	1	$\cdots$	0	$\cdots$	1	$\cdots$
0	0	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	0	1	1	$\cdots$	0	$\cdots$	1	$\cdots$
0	1	0	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	1	0	1	$\cdots$	1	$\cdots$	1	$\cdots$
0	1	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	1	1	1	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	0	0	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	0	1	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	1	1	$\cdots$	1	$\cdots$	1	$\cdots$
1	1	0	0	$\cdots$	1	$\cdots$	0	$\cdots$
1	1	0	1	$\cdots$	1	$\cdots$	0	$\cdots$
1	1	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
1	1	1	1	$\cdots$	1	$\cdots$	1	$\cdots$
Assignments				Clauses				

Fig. 6. An excerpt of matrix $M^{(4)}$. On the left we can see the assignments which are associated to the corresponding rows of the matrix. Only the columns corresponding to the clauses $x_1 \vee x_2 \vee \neg x_4$ and $\neg x_1 \vee \neg x_2 \vee x_3$ are detailed

element $M_{ji}^{(2n)}$ is equal to 1 if and only if the assignment associated with row j satisfies the clause associated with column i. Figure 6 shows an excerpt of matrix $M^{(4)}$, where each row has been labelled with the corresponding clause; only the

```
Check Satisfiability($M^{(2n)}$)

$res \leftarrow [1\ 1\ \cdots\ 1]$        // $2^{2n}$ elements
for all columns $C$ in $M^{(2n)}$
    do if $C$ corresponds to a selected clause
        then $res \leftarrow res \wedge C$        // bit-wise AND
return $res$
```

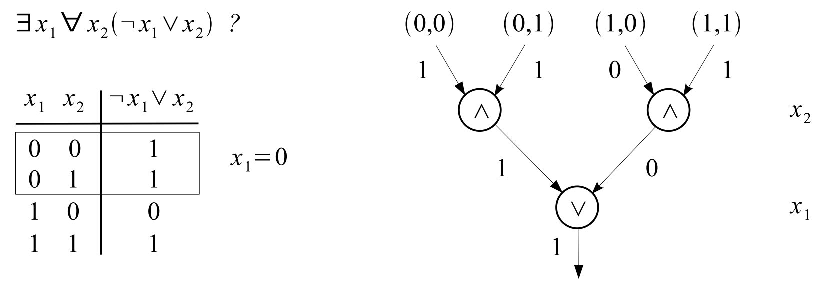
Fig. 7. Pseudocode of the algorithm used to select the assignments that satisfy all the clauses of $\gamma_{2n,m} \in Q3SAT(2n)$

columns that correspond to clauses $x_1 \vee x_2 \vee \neg x_4$ and $\neg x_1 \vee \neg x_2 \vee x_3$ are shown in details.

Let us now consider the algorithm given in pseudocode in Figure 7. The variable res is a vector of length 2^{2n}, whose components — which are initialized to 1 — are bijectively associated with all the possible assignments of $x_1, x_2, \ldots, x_{2n}$. The components of res are treated as flags: when a component is equal to 1, it indicates that the corresponding assignment satisfies all the clauses which have been examined so far. Initially we assume that all the flags are 1, since we do not have examined any clause yet. The algorithm then considers all the columns of $M^{(2n)}$, one by one. If the column under consideration does not correspond to a selected clause, then it is simply ignored. If, on the other hand, it corresponds to a clause which has been selected as part of the instance, then the components of res are updated, putting to 0 those flags that correspond to the assignments which do not satisfy the clause. At the end of this operation, which can be performed in parallel on all the components, only those assignments that satisfy all the clauses previously examined, as well as the clause currently under consideration, survive the filtering process. After the last column of $M^{(2n)}$ has been processed, we have that the components of vector res indicate those assignments that satisfy all the clauses of the instance $\gamma_{2n,m}$ of Q3SAT given as input. Stated otherwise, res is the output column of the truth table of the unquantified propositional formula contained in $\gamma_{2n,m}$.

This algorithm can be easily transformed into an exponential size Boolean circuit, that mimics the operations performed on the matrix $M^{(2n)}$, described by the pseudocode given in Figure 7. Such a circuit can then be easily simulated using the SN P system that we have outlined in Figure 5 (precisely, the left side of the system, until the column of neurons that contain the final value of vector res). This part of the system is composed of three layers for each possible 3-clause that can be built using $2n$ Boolean variables. Two of these layers are used to store the intermediate values of vector res and the values contained in the columns of $M^{(2n)}$, respectively. The third layer, represented by the boxes marked with *Filter* in Figure 5, transforms the current value of res to the value obtained by applying the corresponding iteration of the algorithm given in Figure 7. This layer is in turn composed by three layers of neurons, as we will see in a moment.

The last part of the system is used to check the satisfiability of the universal and existential quantifiers associated with the variables $x_1, x_2, \ldots, x_{2n}$. The neurons in this part of the system compose a binary tree of depth $2n$; the first layer of neurons corresponds to the bottom of the tree, and checks the satisfiability of the quantifier $\forall x_{2n}$; the second layer checks the satisfiability of $\exists x_{2n-1}$ and so on, until the last layer, whose only neuron is σ_{out} (the output neuron), that checks the satisfiability of $\exists x_1$. To see how the check is performed, let us consider the fully quantified



x_1	x_2	$\neg x_1 \vee x_2$
0	0	1
0	1	1
1	0	0
1	1	1

Fig. 8. Example of a quantified Boolean formula formed by one clause, built using two Boolean variables. On the left, its truth table is reported with an indication of the truth assignments that make the formula *true*. On the right, the tree which is used to check the satisfaction of the quantifiers $\forall$ and $\exists$ is depicted

formula $\exists x_1 \forall x_2 (\neg x_1 \vee x_2)$. This formula is composed of a single 2-clause (hence it is not a valid instance of Q3SAT), built using two Boolean variables. In Figure 8 we can see the truth table of the clause, and an AND/OR Boolean circuit that can be used to check whether the quantifiers associated with x_1 and x_2 are satisfied. This circuit is a binary tree whose nodes are either AND or OR gates. Each layer of nodes is associated with a Boolean variable: precisely, the output layer is associated to x_1, the next layer to x_2, and so on until the input layer, which is associated to x_n. If $Q_i = \forall$ then the nodes in the layer associated with x_i are AND gates; on the contrary, if $Q_i = \exists$ then the nodes in such a layer are OR gates. The input lines of the circuit are bijectively associated to the set of all possible truth assignments. It is not difficult to see that when these input lines are fed with the values contained in the output column of clause's truth table, the output of the circuit is 1 if and only if the fully quantified formula is *true*. Since in the first part of the system we have computed the output column of the truth table of the unquantified propositional formula contained in $\gamma_{2n,m}$, we just have to feed these values to an SN P system that simulates the above AND/OR Boolean circuit to see whether $\gamma_{2n,m}$ is *true* or not. Implementing such a Boolean circuit by means of an SN P system is trivial: to see how this can be done, just compare the last two layers of the circuits illustrated in Figures 10 and 11.

The overall system then works as follows. During the computation, spikes move from the leftmost to the rightmost layer. One spike is expelled to the environment by neuron σ_{out} if and only if $\gamma_{2n,m}$ is *true*. In the initial configuration, every neuron in the first layer (which is bijectively associated with one of the 2^{2n} possible assignments to the Boolean variables $x_1, x_2, \ldots, x_{2n}$) contains one spike, whereas neurons $sel_1, sel_2, \ldots, sel_{Cl(2n)}$ contain one or zero spikes, depending upon whether or not the corresponding clause is part of the instance $\gamma_{2n,m}$ given as input. Stated otherwise, the user must provide one spike — in the initial configuration of the system — to every input neuron sel_i that corresponds to a clause that has to be selected. In order to deliver these spikes at the correct moment to all the filters that correspond to the ith iteration of the algorithm, every neuron sel_i contains the rule $a \to a; 4(i-1)$, whose delay is proportional to i. In order to synchronize the execution of the system, also the neurons that correspond to the ith column of $M^{(2n)}$ deliver their spikes simultaneously with those distributed by neurons sel_i, using the same rules. An alternative possibility is to provide the input to the system in a sequential way, for example as a bit string of length $Cl(2n)$, where a 1 (resp., 0) in a given position indicates that the corresponding clause has to be selected (resp., ignored). In this case we should use a sort of delaying subsystem, that delivers — every four time steps — the received spike to all the neurons that correspond to the column of $M^{(2n)}$ currently under consideration. Since the execution time of our algorithm is proportional to the number $Cl(2n)$ of all possible clauses containing $2n$ Boolean variables, this modification keeps the computation time of the entire system cubic with respect to n.

In the first computation step, all the inputs going into the first layer of filters are ready to be processed. As the name suggests, these filters put to 0 those flags which correspond to the assignments that do not satisfy the first clause (corresponding to the first column of $M^{(2n)}$). This occurs only if the clause has been selected as part of the instance $\gamma_{2n,m} \in Q3SAT(2n)$ given as input, otherwise all the flags are kept unchanged, ready to be processed by the next layer of filters. In either case, when the resulting flags have been computed they enter into the second layer of filters together with the values of the second column of $M^{(2n)}$, and the input sel_2 that indicates whether this column is selected or not as being part of the instance. The computation proceeds in this way until all the columns of $M^{(2n)}$ have been considered, and the resulting flags (corresponding to the final value of vector res in the pseudocode of Figure 7) have been computed.

Before looking at how the system checks the satisfiability of the universal and existential quantifiers $\exists x_1, \forall x_2, \ldots, \forall x_{2n}$, let us describe in detail how the filtering process works. This process is performed in parallel on all the flags: if the clause C_i has been selected then an AND is performed between the value $M_{ji}^{(2n)}$ (that indicates whether the jth assignment satisfies C_i) and the current value of the flag res_j (the jth component of res); as a result, res_j is 1 if and only if the jth assignment satisfies all the selected clauses which have been considered up to now. On the other hand, if the clause C_i has not been selected then the old value of res_j is kept unaltered. This filtering process can be summarized by the pseudocode

```
FILTER($sel_i, res_j, C_i$)

if $sel_i = 0$ then return $res_j$
           else return $res_j \wedge C_i$
```

Fig. 9. Pseudocode of the Boolean function computed by the blocks marked with FILTER in Figure 5

given in Figure 9, which is equivalent to the following Boolean function:

$$(\neg sel_i \wedge res_j) \vee (sel_i \wedge res_j \wedge C_i)$$

Such a function can be computed by the Boolean circuit depicted in Figure 10, that in turn can be simulated by the SN P system illustrated in Figure 11. Note the system represented in this latter figure is a generic module which is used many times in the whole system outlined in Figure 5, hence we have not indicated the delays which are needed in neurons sel_i and C_i. Also neuron 1, which is used to negate the value emitted by neuron sel_i, must be activated together with sel_i, that is, after $4(i-1)$ steps after the beginning of the computation. The spike it contains can be reused in the namesake neuron that occurs in the next layer of filters.

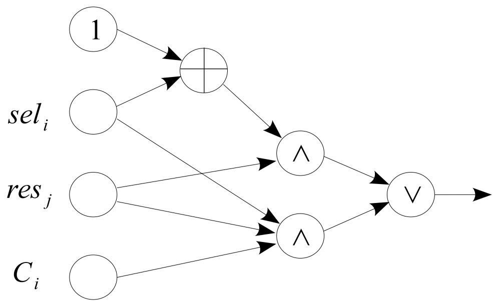


Fig. 10. The Boolean circuit that computes the function FILTER whose pseudocode is given in Figure 9

The last part of the system illustrated in Figure 5 is devoted to check the satisfiability of the universal and existential quantifiers $\exists x_1, \forall x_2, \ldots, \forall x_{2n}$ associated to the Boolean variables $x_1, x_2, \ldots, x_{2n}$. As we have seen, the final values of vector res represent the output column of the truth table of the unquantified propositional formula contained in $\gamma_{2n,m}$. Hence, to check whether all the universal and existential quantifiers are satisfied, it suffices to feed these values as input to an SN P system that simulates a depth $2n$ AND/OR Boolean circuit that operates as described in Figure 8. Each gate is simply realized as a neuron that contains

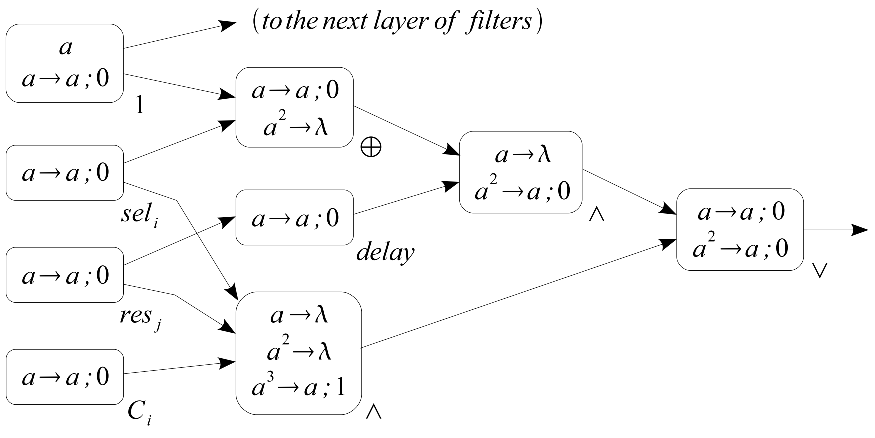


Fig. 11. An SN P system that computes the function FILTER given in Figure 9, simulating the Boolean circuit of Figure 10

two rules, as depicted in the last two layers of Figure 11. At each computation step, one quantifier is checked; when the check terminates, one spike is emitted to the environment by the output neuron σ_{out} if and only if the fully quantified formula $\gamma_{2n,m} \in Q3SAT(2n)$ given as input to the entire system is *true*. The total computation time of the system is proportional to the number $Cl(2n)$ of columns of $M^{(2n)}$, that is, $\Theta(n^3)$.

As we can see, the structure of the system that uniformly solves all the instances of $Q3SAT(2n)$ is very regular, and does not contain "hidden information". For the sake of regularity we have also omitted some possible optimizations, that we briefly mention here. The first column of neurons in Figure 5 corresponds to the initial value of vector res in the pseudocode given in Figure 7. Since this value is fixed, we can pre-compute part of the result of the first step of computation, and remove the entire column of neurons from the system. In a similar way we can also remove the subsequent columns that correspond to the intermediate values of res, and send these values directly to the next filtering layer. A further optimization concerns the values $M_{ji}^{(2n)}$, which are contained in the neurons labelled with C_i. Since these values are given as input to AND gates, when they are equal to 1 they can be removed since they do not affect the result; on the other hand, when they are equal to 0 also the result is 0, and thus we can remove the entire AND gate.

5 Conclusions and Remarks

In this paper we have shown that QSAT, a well known **PSPACE**-complete problem, can be deterministically solved in linear time with respect to the number n of variables and the number m of clauses by an exp-uniform family of SN P systems with pre-computed resources. We have also considered `Q3SAT`, a restricted (but still **PSPACE**-complete) version of `QSAT` in which all the clauses of the instances have exactly three literals; we have shown that in this case the problem can be solved in a time which is at most cubic in n, independent of m. Each pre-computed SN P system of the family can be used to solve all the instances of `QSAT` (or `Q3SAT`), expressed in a normalized form, of a given size.

Note that using pre-computed resources in spiking neural P systems is a powerful technique, that simplifies the solution of computationally hard problems. The pre-computed SN P systems presented in this paper have an exponential size with respect to n but, on the other hand, have a regular structure. It still remains open whether such pre-computed resources can be constructed in a regular way by using appropriate computation devices that, for example, use a sort of controlled duplication mechanism to produce an exponential size structure in a polynomial number of steps. A related interesting problem is to consider whether alternative features can be introduced in SN P systems to uniformly solve **PSPACE**-complete problems. Nondeterminism is the first feature that comes to mind, but this is usually considered too powerful in the Theory of Computation.

Acknowledgements

Valuable comments and suggestions from professor Mario J. Pérez-Jiménez are greatly appreciated. This work was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60803113), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The first author was partially supported by Academy of Finland, project 122426. The second author was partially supported by MIUR project "Mathematical aspects and emerging applications of automata and formal languages" (2007).

References

1. J.L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity*, Voll. I and II, Springer-Verlag, Berlin, 1988–1990.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75 (2007), 141–162.
3. H. Chen, M. Ionescu, T.-O. Ishdorj, On the efficiency of spiking neural P systems. *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh, Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), Sevilla, January 30– February 3, Sevilla University, Fénix Editora, 2006, 195–206.
4. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: universality and languages. *Natural Computing*, 7, 2 (2008), 147–166.

5. M.R. Garey, D.S. Johnson: *Computers and Intractability. A Guide to the Theory of* **NP**-*Completeness*. W.H. Freeman and Company, San Francisco, 1979.
6. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, D. Ramírez-Martínez: A software tool for verification of spiking neural P systems. *Natural Computing*, 7, 4 (2008), 485–497.
7. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2–3 (2006), 279–308.
8. T.-O. Ishdorj, A. Leporati: Uniform solutions to `SAT` and `3-SAT` by spiking neural P systems with pre-computed resources. *Natural Computing*, 7, 4 (2008), 519–534.
9. A. Leporati, M.A. Gutiérrez-Naranjo: Solving `Subset Sum` by spiking neural P systems with pre-computed resources. *Fundamenta Informaticae*, 87, 1 (2008), 61–77.
10. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez: Uniform solutions to `SAT` and `Subset Sum` by spiking neural P systems. *Natural Computing*, online version (DOI: 10.1007/s11047-008-9091-y).
11. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical **NP**-complete problem with spiking neural P systems. *Membrane Computing, International Workshop, WMC8* (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Thessaloniki, Greece, 2007, LNCS 4860, Springer-Verlag, Berlin, 2007, 336–352.
12. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *International Journal of Unconventional Computing*, to appear.
13. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90, 1 (2007), 48–60.
14. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 1, 61 (2000), 108–143. See also Turku Centre for Computer Science – TUCS Report No. 208, 1998.
15. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
16. A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.
17. The P systems Web page: `http://ppage.psystems.eu`