Uniform solutions to SAT and Subset Sum by spiking neural P systems

Alberto Leporati · Giancarlo Mauri · Claudio Zandron · Gheorghe Păun · Mario J. Pérez-Jiménez

Abstract We continue the investigations concerning the possibility of using spiking neural P systems as a framework for solving computationally hard problems, addressing two problems which were already recently considered in this respect: `Subset Sum` and `SAT`. For both of them we provide uniform constructions of standard spiking neural P systems (i.e., not using extended rules or parallel use of rules) which solve these problems in a constant number of steps, working in a non-deterministic way. This improves known results of this type where the construction was non-uniform, and/or was using various ingredients added to the initial definition of spiking neural P systems (the SN P systems as defined initially are called here "standard"). However, in the `Subset Sum` case, a price to pay for this improvement is that the solution is obtained either in a time which depends on the value of the numbers involved in the problem, or by using a system whose size depends on the same values, or again by using complicated regular expressions. A uniform solution to 3-`SAT` is also provided, that works in constant time.

A. Leporati · G. Mauri · C. Zandron
Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano – Bicocca, Viale Sarca 336, 20126 Milano, Italy
e-mail: leporati@disco.unimib.it

G. Mauri
e-mail: mauri@disco.unimib.it

C. Zandron
e-mail: zandron@disco.unimib.it

G. Păun (✉)
Institute of Mathematics of the Romanian Academy, P. O. Box 1-764, 014700 Bucharest, Romania
e-mail: george.paun@imar.ro; gpaun@us.es

G. Păun · M. J. Pérez-Jiménez
Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda Reina Mercedes s/n, 41012 Sevilla, Spain
e-mail: marper@us.es

Keywords Membrane computing · Spiking neural P system · SAT problem · Subset sum problem · Complexity

1 Introduction

Spiking neural P systems (in short, SN P systems) were introduced in Ionescu et al. (2006) as a class of P systems which incorporate into membrane computing specific ideas from the way biological neurons communicate through electrical impulses of identical form (spikes). We refer to Ionescu et al. (2006) and to other papers which can be found at the Web site of membrane computing (The P Systems Web Page: http://psystems.disco.unimib.it.) for motivation and basic definitions.

In short, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along the arcs of the graph (called *synapses*). The spikes evolve by means of rules which, in the first papers reported in this area, were of two types: (i) *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 such that $c \geq 1$, $d \geq 0$, and (ii) *forgetting rules*, of the form $a^s \to \lambda$, where $s \geq 1$ is a natural number. Using a rule of the former type means that if a neuron contains k spikes, $k \geq c$, and $a^k \in L(E)$, 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. Using a forgetting rule means that s spikes are removed, provided that the neuron contains exactly s spikes. If two spiking rules can be used at the same time in a neuron (i.e., both their regular expressions describe the contents of that neuron), then one of them is non-deterministically chosen, but, by definition, in Ionescu et al. (2006) it is forbidden to have a spiking rule $E/a^c \to a; d$ and a forgetting rule $a^s \to \lambda$ such that $a^s \in L(E)$.

A common generalization of these types of rules was introduced in Chen et al. (2006b) and Păun and Păun (2007) under the name of *extended rules*. These rules are of the form $E/a^c \to a^p; d$, with the meaning that when using the rule, c spikes are consumed and p spikes are produced. Because p can be 0 or greater than 0, we obtain a generalization of both standard spiking and forgetting rules, with the additional feature of having the forgetting rules controlled by regular expressions. Moreover, forgetting rules are now allowed to compete in a non-deterministic way with firing rules.

In each time unit (a common clock is assumed to exist, marking the time for all neurons), each neuron which can use a rule, of any type, has to do it (each neuron can use at most one rule, but the neurons work synchronously, evolving in parallel). 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 (at least) a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* produced by the system—it is infinite if the computation does not halt.

With a spike train we can associate various numbers, which can be considered as *computed* (we also say *generated*) by an SN P system. For instance, in Ionescu et al. (2006) only the distance between the first two spikes of a spike train was considered, while in Păun et al. (2002) several extensions were examined which we do not mention here.

An SN P system can also work in the accepting mode: a neuron is designated as the *input neuron* and a spike train is introduced in it; this spike train is accepted if the computation halts. In particular, we can introduce a spike train with only two spikes, coming at an interval of n steps, and then we say that the number n is accepted if the computation halts.

Two main types of results on the computational power of SN P systems have been obtained in the literature: computational completeness in the case when no bound was imposed on the number of spikes present in the system, and a characterization of semi-linear sets of numbers in the case when a bound was imposed. Improvements in the form of the regular expressions, removing the delay, or the forgetting rules can be found in Ibarra et al. (2007). The result is true both for the generative and the accepting cases.

SN P systems can be also used for solving decision problems: Given a problem Q, for example, with instances $Q(n, m)$ characterized by the size parameters n and m (as it is the case of SAT, the satisfiability of propositional formulas in the conjunctive normal form, where n is the number of variables and m is the number of clauses in a given formula), we construct an SN P system Π_Q such that, when having an instance $Q(n, m)$, we introduce a polynomial number of spikes in a designated input neuron of Π_Q and the computation halts if and only if $Q(n, m)$ has a solution. Alternately, we can assume that $Q(n, m)$ has a solution if and only if the system sends at least a spike to the environment (we will see below that, in certain circumstances, the two modes, halting or spiking at least once, are equivalent). When the system is constructed in such a way to depend directly on the instance $Q(n, m)$, then we say that the construction is non-uniform (it is semi-uniform if it is done in a polynomial time); then we do not need an input neuron, as the instance is embedded in the structure (number of spikes, graph of neurons, rules) from the very beginning. In the case of uniform constructions, a useful extension is to consider 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.

Besides the sequential use of rules in each neuron, also a parallel way was considered, in two forms: the *exhaustive* mode of (Ionescu et al. 2007) (when a rule is enabled, then it is used as many times as possible in that neuron), and the *maximally parallel* mode of (Leporati et al. 2007b) (as many rules are used as enabled in a neuron).

The present paper considers SN P systems for solving decision problems, continuing the papers of Leporati et al. (2007a, b). The first of these papers deals with the Subset Sum problem, the latter with the SAT problem. For both these problems, constant time solutions were provided in these papers by using SN P systems constructed in a semi-uniform way, working in a non-deterministic way, and also using a series of ingredients added to SN P systems of the standard form: extended rules, the possibility to have a choice between spiking rules and forgetting rules, etc. We improve here the constructions from Leporati et al. (2007a, b), by using only standard SN P systems, avoiding the use of delay, providing *uniform* constructions (a uniform construction for solving SAT was also presented in Leporati et al. (2007b), but the idea of the construction given here is different and more transparent).

It is worth stressing that we are mainly interested here in having *uniform* constructions (in the sense of devising systems associated with the problems, not with specific instances; however, we do not take care of the time needed for the construction), using *standard* features of SN P systems, and in getting *constant* time solutions of the considered decision problems, working in a *non-deterministic* way. Our systems are in general of an exponential size with respect to the size of the problems (e.g., the number of neurons depends on the size of the numbers involved in the Subset Sum problem and are polynomial in n and m for a SAT(n, m) problem).

The paper is organized as follows. In the next section we formally introduce the SN P systems, in Sect. 3 we deal with the Subset Sum problem, in Sect. 5 we provide solutions to SAT, and in Sect. 6 we give a solution to the 3-SAT problem. Section 4 gives an auxiliary result: in the case of extended systems, halting is the same as spiking, hence we do not have to pay attention to this aspect of the definition. We close with some concluding remarks and open problems in Sect. 7.

2 SN P systems

We introduce SN P systems in the standard form, in the computing version (i.e., able to take an input and provide an output).

A computing *spiking neural P system* of degree $m \geq 1$ is a construct of the form

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

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ 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 $c \geq 1$, $d \geq 0$ are natural numbers;
      (2) $a^s \rightarrow \lambda$, where $s \geq 1$ is a natural number, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for all $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons);
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neuron, respectively.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, those of type (2) are called *forgetting* rules. The firing 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 $E/a^c \rightarrow a; d \in R_i$ can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in σ_i); the neuron is fired, and it produces a spike after d time units. 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 $t, t + 1, t + 2, \ldots, t + d - 1$ 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 again open, 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 spike reaches immediately all neurons σ_j such that $(i, j) \in syn$ and which are open, that is, the spike is replicated and each target neuron receives a copy of it; spikes sent to a closed neuron are "lost".

The forgetting rules are applied as follows: if neuron σ_i contains exactly s spikes, then the rule $a^s \rightarrow \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \rightarrow a; d$ of type (1) has $E = a^c$, then we write it in the simplified form $a^c \rightarrow a; d$. If all rules of a system have $d = 0$, i.e., no delay is involved, then the parameter d is omitted and the rules are written in the form $E/a^c \rightarrow a$.

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, $E_1/a^{c_1} \rightarrow a; d_1$ and $E_2/a^{c_2} \rightarrow a; d_2$, 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, and in that case, only one of them is chosen non-deterministically. Note however that, by definition, 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 notice 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, …, 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 $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$.

A computation in a system as above starts in the initial configuration. In order to compute a function $f : \mathbf{N}^k \longrightarrow \mathbf{N}$, we introduce k natural numbers $n_1, \ldots, n_k$ in the system by "reading" from the environment a binary sequence $z = 10^{n_1-1}10^{n_2-1}1 \ldots 10^{n_k-1}1$. This means that the input neuron of Π receives a spike in each step corresponding to a digit 1 from the string z and no spike otherwise. (Another possibility is to consider k input neurons and to introduce each n_i, $1 \leq i \leq k$, as the distance between two spikes which enter the ith input neuron.) The result of the computation is also encoded in the distance between two spikes: we impose the restriction that the system outputs exactly two spikes and halts (sometimes after the second spike), hence it produces a spike train of the form $0^{b_1}10^{r-1}1b^{b_2}$, for some $b_1, b_2 \geq 0$ and with $r = f(n_1, \ldots, n_k)$ (the system outputs no spike a non-specified number of steps from the beginning of the computation until the first spike).

The previous definition covers many types of systems/behaviors. If the neuron σ_{in} is not specified, then we have a generative system: we start from the initial configuration and we collect all results of computations. Alternatively, we can compute a function which relates the input to the output.

A further possibility is to use an SN P system in the accepting mode: an input is introduced in the system and it is accepted if and only if the computation halts, or if and only if the output neuron spikes at least once. Precisely, an *accepting* SN P system of degree (m, ℓ), with $m \geq 1$ and $0 \leq \ell \leq m$, is defined like a standard SN P system of degree m, the only difference being that now there are ℓ input neurons denoted by $in_1, \ldots, in_\ell$. As a particular case, when $\ell = 0$ we have SN P systems without any input neuron. A *valid input* of an accepting SN P system of degree (m, ℓ) is a set of ℓ binary sequences, that collectively encode an instance of a decision problem.

A computation in an accepting SN P system of degree (m, ℓ) starts in the initial configuration. The valid input (encoding an instance of a decision problem) is processed, by making each input neuron in_i read from the environment a binary sequence $z_i = 10^{n_{i,1}-1}10^{n_{i,2}-1}1 \ldots 10^{n_{i,k_i}-1}1$ as described above. In this way, neuron in_i reads the sequence $n_{i,1}, n_{i,2}, \ldots, n_{i,k_i}$ of natural numbers. Then, the computation proceeds as usual, and we say that it is an *accepting computation* if and only if the system halts and the output neuron spikes exactly once. Let us note that the time spent by an SN P system working with a valid input is, at least, the maximum length of the binary sequences provided by the input.

We do not give here further details, but we refer the reader to the bibliography. In particular, we do not give examples of SN P systems, but several explicit constructions of SN P systems will be provided in the next sections.

2.1 Complexity classes in SN P systems

Since in this paper we will be dealing with the `Subset Sum`, `SAT` and `3−SAT` **NP**-complete decision problems, we recall here some basic definitions concerning the solution of decision problems by means of SN P systems.

Definition 1 Let $X = (I_X, \theta_X)$ be a decision problem, and $g : \mathbf{N} \to \mathbf{N}$ a computable function. We say that X is solvable by a family $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ of SN P systems, in time bounded by g, in a non-deterministic and uniform way (we denote it by $X \in \mathbf{NSN}(g)$), if the following holds:

- The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine working in polynomial-time which constructs the SN P system $\Pi(n)$ from $n \in \mathbf{N}$.
- There exist polynomial time computable functions, *cod* and *s*, over I_X, such that
  - For each instance $w \in I_X$, $s(w)$ is a natural number, and $cod(w)$ is a valid input of the SN P system $\Pi(s(w))$.
  - The family $\mathbf{\Pi}$ is g-bounded with respect to (X, *cod*, *s*); that is, for each instance $w \in I_X$, the minimum length of an accepting computation of $\Pi(s(w))$ with input $cod(w)$ is bounded by $g(|w|)$.
  - The family $\mathbf{\Pi}$ is sound with respect to (X, *cod*, *s*); that is, for every $w \in I_X$, if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$, then $\theta_X(w) = 1$.
  - The family $\mathbf{\Pi}$ is X-complete with respect to (X, *cod*, *s*); that is, for every $w \in I_X$, if $\theta_X(w) = 1$ then there exists a computation of $\Pi(s(w))$ with input $cod(w)$ which is an accepting computation.

Definition 2 We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ of SN P systems, in a non-deterministic and uniform way (we denote it by $X \in \mathbf{NPSN}$), if there exists $k \in \mathbf{N}$ such that X is solvable by the family $\mathbf{\Pi}$ in time bounded by a polynomial, in a non-deterministic and uniform way.

Definition 3 Let $X = (I_X, \theta_X)$ be a decision problem, and g: $\mathbf{N} \to \mathbf{N}$ a computable function. We say that X is solvable by a family $\mathbf{\Pi} = (\Pi(w))_{w\in I_X}$ of SN P systems, in time bounded by g, in a non-deterministic and semi-uniform way (we denote it by $X \in \mathbf{NSN}^*(g)$), if the following holds:

- The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine working in polynomial-time which constructs the SN P system $\Pi(w)$ from the instance $w \in I_X$.
- The family $\mathbf{\Pi}$ is g-bounded with respect to X; that is, for each $w \in I_X$, the minimum length of an accepting computations of $\Pi(w)$ is bounded by $g(|w|)$.
- The family $\mathbf{\Pi}$ is sound with respect to X; that is, for every $w \in I_X$, if there exists an accepting computation of $\Pi(w)$, then $\theta_X(w) = 1$.
- The family $\mathbf{\Pi}$ is complete with respect to X; that is, for every $w \in I_X$, if $\theta_X(w) = 1$ then there exists a computation of $\Pi(w)$ which is an accepting computation.

Definition 4 We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(w))_{w\in I_X}$ of SN P systems, in a non-deterministic and semi-uniform way (we denote it by $X \in \mathbf{NPSN}^*$), if there exists $k \in \mathbf{N}$ such that X is solvable by the family $\mathbf{\Pi}$ in time bounded by a polynomial, in a non-deterministic and semi-uniform way.

Let us only comment upon the *soundness* and the *completeness* properties. The *soundness* property requires that if we obtain an acceptance response from the system (associated with an instance) through some computation, then the answer of the problem (for that instance) is affirmative. On the other hand, the *completeness* property means that if we obtain an affirmative response to an instance of the problem, then there exists an accepting computation of the system (associated with that instance).

Let us finally note that the complexity classes defined above are closed under polynomial time reduction, in the classical sense, but in general they are not closed under complementation.

3 Solving Subset Sum

Let us start by recalling the **NP**-complete Subset Sum problem, here reformulated in an equivalent form with respect to Garey and Johnson (1979, p. 223)

Problem 1 NAME: Subset Sum

- INSTANCE: a (multi)set $V = \{v_1, v_2, \ldots, v_n\}$ of positive integer numbers, and a positive integer number S.
- QUESTION: is there a sub(multi)set $B \subseteq V$ such that $\sum_{b \in B} b = S$?

In Leporati et al. (2007a) an SN P system (recalled in Fig. 1 in a slightly modified form; note that we omit the delay when it is 0 for all rules) was given for solving the Subset Sum problem in a non-deterministic way, in two steps: neurons $\sigma_1, \ldots, \sigma_n$ choose non-deterministically some numbers among $v_1, \ldots, v_n$, and neuron σ_{out} checks whether the sum of the chosen numbers equals S; if this is the case, a spike is sent to the environment. Hence, the instance of the problem encoded in the system, by means of the initial spikes present in neurons $\sigma_1, \ldots, \sigma_n$ as well as of the rules of these neurons (and also of σ_{out}, which "knows" the value of S) has a solution if and only if there is a computation which spikes in step 2. It is important to note that the system is rather simple, but it is non-uniformly constructed, it uses extended rules, and is allowed to contain forgetting rules which are used in competition with spiking rules. This competition provides the choice opportunity, necessary for the non-deterministic behavior of the system. We will improve this construction below from several of these points of view.

An alternative way to solve Subset Sum is to use the system depicted in Fig. 2, which is inpired by Proposition 3.1 in Leporati et al. (2007b). Given the instance ($V = \{v_1, v_2, \ldots, v_n\}$, S) of Subset Sum, let

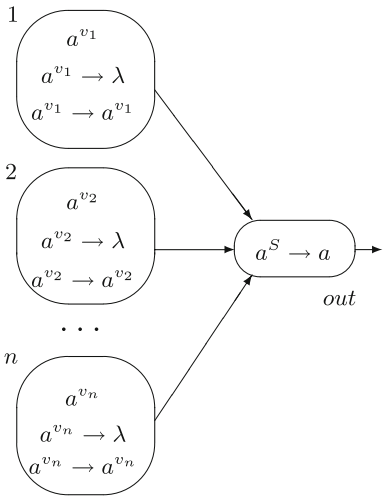


Fig. 1 The SN P system given in Leporati et al. (2007a) for solving the Subset Sum problem

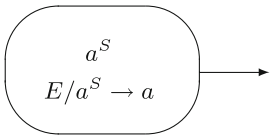


Fig. 2 A simple SN P system that solves one instance of `Subset Sum`

$$E = (a^{v_1} \cup \lambda)(a^{v_2} \cup \lambda)\ldots(a^{v_n} \cup \lambda)$$

be the corresponding regular expression built using the values $v_1, v_2, \ldots, v_n$ from V. The neuron that composes the system depicted in Fig. 2 contains only the firing rule $E/a^S \to a$ (the omitted delay is 0, and E is the above regular expression), and is initialized with S spikes. According to the standard definition of SN P systems, the rule can fire (thus sending a single spike to the environment) if and only if $a^S \in L(E)$, that is, if and only if there exists a subset of V whose elements sum up to S. Also this solution uses a system which is non-uniformly constructed, but it does not use neither extended rules nor forgetting rules in competition which spiking rules. However it uses complicated regular expressions, that involve strings whose length is proportional to the values v_i contained into V (and thus, possibly exponential with respect to the usually agreed instance size of `Subset Sum`: $\Theta(n \log K)$, where $K = \max\{v_1, \ldots, v_n, S\}$).

Let us now consider the SN P system depicted in Fig. 3. It uses only standard rules, and solves the `Subset Sum` problem in four steps, working non-deterministically. The non-determinism is now provided by the rules $a \to a; 0$ and $a \to a; 1$ from neurons $\sigma_{c_1}, \ldots, \sigma_{c_n}$. If the first rule is chosen in σ_{c_i}, then neuron σ_{d_i} receives immediately two spikes (one is coming from neuron σ_{c_0}), fires, and sends a spike to neurons $\sigma_{e_{i,j}}, 1 \leq j \leq v_i$; each of these neurons spikes and sends one spike to neuron σ_{out}. If the rule $a \to a; 1$ is chosen in σ_{c_i}, then σ_{d_i} receives one spike from σ_{c_0} and forgets it in the next step, when also a spike from σ_{c_i} arrives, and it is also removed by the forgetting rule. Thus, no spike is sent to neurons $\sigma_{e_{i,j}}, 1 \leq j \leq v_i$, hence to σ_{out}. This means that some of the numbers v_i, $1 \leq i \leq n$, are non-deterministically chosen, and the output neuron accumulates their sum. The output neuron spikes if and only if this sum equals S; this spike is sent to the environment in step 4 of the computation.

Note that this system is constructed in a non-uniform way, it consists of $\sum_{i=1}^{n} v_i + 2n + 2$ neurons, and initially contains $n + 1$ spikes.

At the price of getting the result after 5 steps and of slightly increasing the number of neurons and of initial spikes, the delay feature can be avoided. To this aim, the neurons $\sigma_{c_i}, \sigma_{d_i}$—as well as σ_{c_0}—are replaced as suggested in Fig. 4. This time, the choice is between using the rule $a^2/a \to a$ or the rule $a^2 \to a$ in the first step of the computation. In the first case, one spike remains to be used in the second step, when the rule $a \to a$ sends it to neuron σ_{d_i}. In this way, during the first step of computation σ_{d_i} either receives one spike or two from σ_{c_i}. In step 2, σ_{d_i} also receives two spikes from neuron σ_{c_0} through neurons $\sigma_{c'_0}, \sigma_{c''_0}$. From now on, the computation proceeds as in the system from Fig. 3.

Let us now pass to constructing the system which solves the `Subset Sum` problem in a uniform way. This means that the system can "know" only the number n, while $v_1, \ldots, v_n$ and S should be introduced in the system (in a specified form) at the beginning of the computation. We choose to introduce the instance of the problem in the system as follows: we consider $n + 1$ input nodes, and we introduce $2v_i$, $1 \leq i \leq n$, spikes in the first n of them and $2S$ spikes in the $(n + 1)$th input neuron. The system is presented in Fig. 5, with these spikes already present in the input neurons σ_{in_i}, $1 \leq i \leq n + 1$.

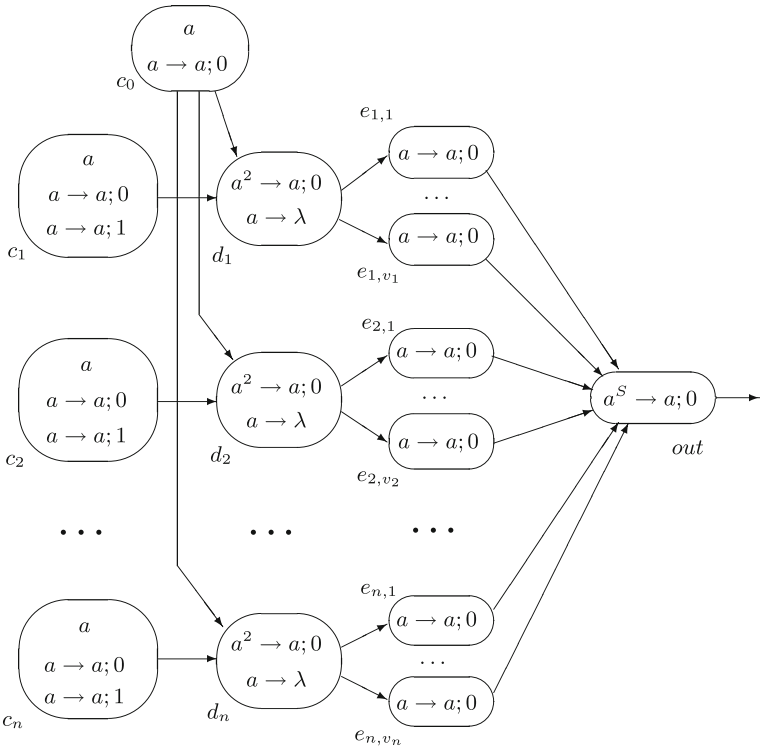


Fig. 3 A standard, non-uniform solution to `Subset Sum`

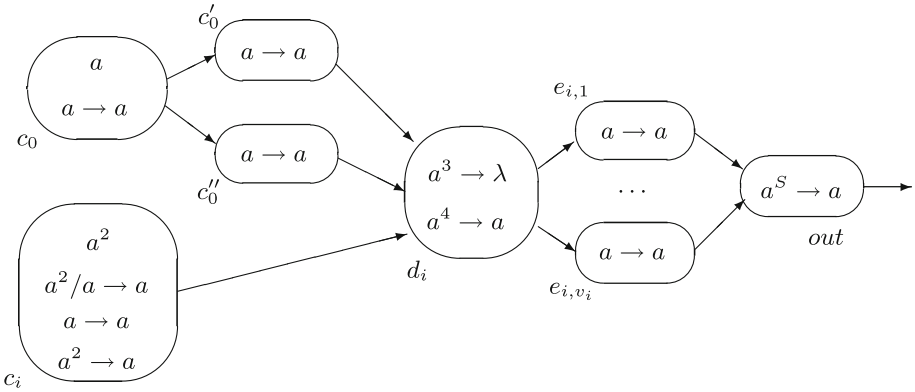


Fig. 4 Removing the use of delays

These spikes cannot be used until receiving a spike from σ_{d_i}, in step 2. This happens non-deterministically, in the same way as in the system from Fig. 3. When firing, each neuron σ_{in_i} sends two spikes (duplicated by $\sigma_{e_{i,1}}, \sigma_{e_{i,2}}$) both to the "accumulator" σ_{acc} and to the "comparison trigger" σ_{t_1}. In each step, starting with the fourth one, neuron σ_{t_1} also receives a spike from σ_{h_4}. Having an odd number of spikes inside, greater than or equal to 3, σ_{t_1} forgets its spikes. When all spikes from neurons σ_{in_i} activated by neurons $\sigma_{c_i}, \sigma_{d_i}$

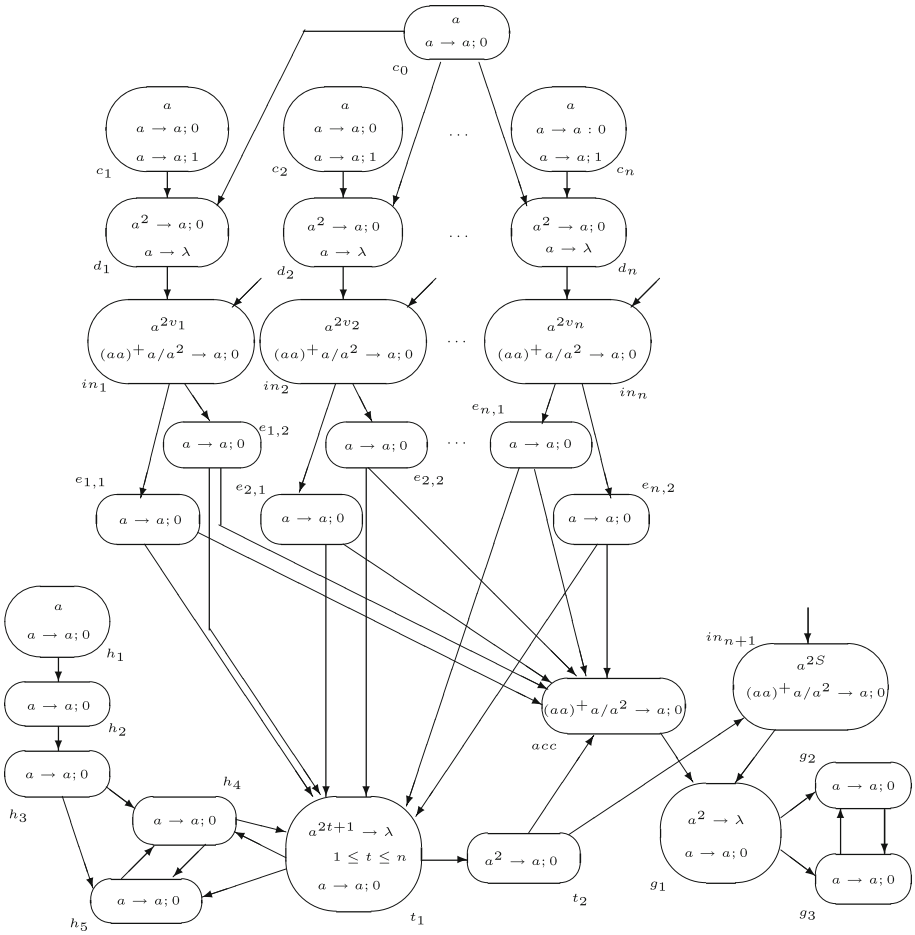


Fig. 5 A uniform SN P system solving the Subset Sum problem

were moved to σ_{acc}, no spike comes to σ_{t_1}, except the spike produced by σ_{h_4}. In that moment, σ_{t_1} can use the rule $a \to a; 0$. The produced spike is sent both to σ_{t_2} (which just waits with this spike inside) and to σ_{h_4} and σ_{h_5}, which from now on contain (at least) two spikes and stop firing. At the same step, σ_{t_1} receives one more spike from σ_{h_4}, fires, and in this way σ_{t_2} gets two spikes. It fires, and sends a spike to both σ_{acc} and $\sigma_{in_{n+1}}$. With an odd number of spikes, these neurons start to fire, each of them sending one spike to σ_{g_1}. With two spikes inside, this neuron forgets them. If the number of spikes accumulated in σ_{acc} equals the number of spikes from $\sigma_{in_{n+1}}$, then the computation will halt. If this is not the case, after exhausting the spikes from σ_{acc} or from $\sigma_{in_{n+1}}$, neuron σ_{g_1} will receive only one spike and will fire. The produced spike will "feed" neurons σ_{g_2} and σ_{g_3} which will work forever, thus preventing the halting of the computation.

Consequently, the computation halts if and only if the non-deterministic choice made by neurons σ_{c_i} provides a solution to the given instance of the Subset Sum problem.

The system from Fig. 5 contains $5n + 13$ neurons, and the computation stops after at most $3\sum_{i=1}^{n} v_i + 6$ steps (we have two initial steps, at most $2\max\{v_i \mid 1 \le i \le n\}$ steps for

moving the spikes from neurons σ_{in_i}, to σ_{acc}, 4 steps to send a spike from σ_{t_1} to σ_{acc} and $\sigma_{in_{n+1}}$, then at most $\Sigma_{i=1}^{n} v_i$ steps for the comparison; since $\max\{v_i \mid 1 \leq i \leq n\} \leq \Sigma_{i=1}^{n} v_i$, we obtain the stated upper bound on the number of computation steps; of course, we may assume that $S \leq \Sigma_{i=1}^{n} v_i$, otherwise it is obvious that the problem has no solution).

The non-determinism is again ensured by the choice between a rule with delay 0 and a rule with delay 1; as we did in Fig. 4, we can avoid using a non-zero delay, by adding one further step to the computation. However, the computation is no longer performed in a constant number of steps, but in a number which is bounded by the values of input numbers v_i, $1 \leq i \leq n$, which can be exponential with respect to n (precisely, with respect to the size of the instance, measured in bits). It is an open problem whether the construction can be improved from this point of view.

Then, we have another aspect here: the problem has a solution if and only if there is a halting computation, and this gives us the opportunity to remark that, in certain circumstances, there is no difference between defining successful computations by halting or by having a spike emitted to the environment.

4 Halting versus Spiking

The following two observations can be considered as auxiliary lemmas with respect to the other results of this paper.

First of all, *given a system* Π, *with standard or extended rules, with or without delays, we can construct a system* Π', *with rules of the same kinds as those of* Π, *which spikes if and only if* Π *halts*.

Indeed, let us consider an SN P system Π of degree n. For each neuron σ_i of $\Pi, 1 \leq i \leq n$, let $P_i = \bigcup\{p|E/a^c \rightarrow a^p; d \text{ is a rule of } \sigma_i\}$ be the set of all possible numbers of spikes which can be produced by σ_i using one of its rules. Let us now define:

$$D = \max\{d \mid E/a^c \rightarrow a^p; d \text{ is a rule of } \Pi\} + 1$$

as the maximal delay occurring in rules, plus 1, and

$$K = \left\{ \sum_{i=1}^{n} p_i | p_i \in P_i \right\} - \{0\}$$

as the set of all possible numbers of spikes which can be produced by Π in a computation step, excluding number 0.

Clearly, the system Π halts if and only if for D consecutive steps no spike is produced in the system (if the system does not use the delay feature, then it stops if in one step no rule is used): if a neuron uses a forgetting rule, then it remains empty, hence it can use again a rule, of any type, only if it will receive further spikes; such spikes arrive at the same time with the use of the forgetting rule or in the next at most $D - 1$ steps; thus, if no spike is generated in the system Π for D steps, then no further rule can be used.

We construct the system Π' as suggested in Fig. 6, where five neurons are added to the system Π. The components of Π are kept unchanged, and from each neuron of Π there is a synapse to σ_{c_1}. In each step where at least one spike is produced in Π, some spikes (whose number belongs to the set K) are sent to neuron σ_{c_1}, which spikes immediately and sends a spike to σ_{out}. Note that K may contain an exponential (in n) number of elements; hence, neuron σ_{c_1} may contain an exponential number of rules as well. In each step, starting with the second one, this neuron also receives two spikes from σ_{c_3} and σ_{c_4}. Any odd number of

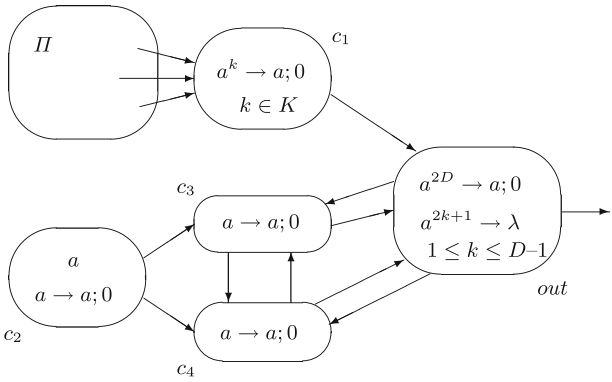


Fig. 6 Passing from halting to spiking

spikes smaller than $2D$ are removed from σ_{out}. If $2D$ spikes are accumulated in σ_{out}, this means that the system Π has produced no spike for D consecutive steps, that is, it halted; in that case, the output neuron of Π' spikes (and stops, because neurons $\sigma_{c_3}, \sigma_{c_4}$ cannot work further, but this is just a by-product of our construction, it is not requested in the statement of our result). Thus, Π' spikes if and only if Π halts.

Note that the neurons added to Π use only standard rules, without delays, and that their functioning is deterministic.

Unfortunately, we do not have a proof for the reverse assertion for standard SN P systems, but only for the case when extended rules are used, and in all of these rules the delay is 0. Indeed, let us consider an SN P system Π with extended rules, and let σ_{out} be its output neuron. We "double" this system by doubling the number of spikes present in the initial configuration in each neuron, and replacing each rule $E/a^c \to a^p; 0$ with $2E/a^{2c} \to a^{2p}; 0$, where $2E$ is a regular expression for the set $\{2n \mid n \in L(E)\}$ (this is a regular set of numbers for each regular expression E, hence the expression $2E$ exists and can be constructed effectively). Let us denote by 2Π the obtained system. Clearly, in this way the behavior of the system is not changed, in the sense that the new system spikes if and only if Π spikes. Now, let us add a further neuron, σ_{new}, with an incoming synapse from σ_{out} and with outgoing synapses to all neurons of 2Π. Provide a rule $a^{2p} \to a; 0$ to σ_{new} for each rule $2E/a^{2c} \to a^{2p}; 0$ from σ_{out} (in the form obtained by "doubling"). Thus, if σ_{out} spikes, then also σ_{new} spikes, and its spike goes to all neurons of 2Π; because in this way, the number of spikes from each neuron of 2Π becomes odd, no rule of 2Π can be used from now on, hence the system halts. Consequently, spiking implies halting—but the use of extended rules is essential in this construction.

Whether or not the extended rules or the restriction to have no delay can be avoided remains as an open problem.

5 Solving SAT

Let us pass now to SAT, the most invoked **NP**-complete problem (Garey and Johnson 1979, p. 39). The instances of SAT depend upon two parameters: the number n of variables, and the number m of clauses. As a consequence, SAT will be uniformly solved by means of a family $\{\Pi(\langle n, m\rangle)\}_{n,m\in\mathbf{N}}$ of SN P systems, where $\Pi(\langle n, m\rangle)$ solves all the instances composed by m clauses, built using n variables.

We recall that a *clause* is a disjunction of literals, occurrences of x_i or $\neg x_i$, built on a given set $X = \{x_1, x_2, \ldots, x_n\}$ of Boolean variables. In what follows we will require that no repetitions of the same literal may occur in any clause; in this way, a clause can be seen as a *subset* of all possible literals. An *assignment* of the variables $x_1, x_2, \ldots, x_n$ is a mapping a: $X \to \{0, 1\}$ that associates to each variable a truth value. The number of all possible assignments to the variables of X is 2^n. We say that an assignment *satisfies* the clause C if, assigned the truth values to all the variables which occur in C, the evaluation of C (considered as a Boolean formula) gives 1 (*true*) as a result.

Problem 2 NAME: SAT

- INSTANCE: a set $C = \{C_1, C_2, \ldots, C_m\}$ of clauses, built on a finite set $\{x_1, x_2, \ldots, x_n\}$ of Boolean variables.
- QUESTION: is there an assignment of the variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in C?

An SN P system which solves this problem in a number of steps which is linear in the number of variables, independent of the number of clauses, working non-deterministically, can be constructed in a uniform manner, using only standard rules. The construction is given again in a pictorial way, starting with the general structure suggested in Fig. 7. Note that, differently from the number of computation steps, both the size and the structure of the system depend upon the number of clauses. Several modules appear in this figure which will be explained below.

Because the construction is uniform, we need a way to codify a given instance of SAT.

Let us consider a propositional formula in the conjunctive normal form, $\gamma = C_1 \wedge C_2 \wedge \ldots \wedge C_m$. A variable x_i, $1 \leq i \leq n$, can appear or not in a clause C_j, and in the case that it appears, it can be negated or not ($\neg x_i$ or x_i). Thus, we have to distinguish between three cases; that is why we use a code of two digits 0 and 1 for indicating the relation between x_i and C_j: 00 indicates the case when x_i does not appear in C_j, 01 (equally, we can use 10) indicates the case when x_i appears in C_j, and 11 corresponds to the case when $\neg x_i$ is present in C_j. It is important to note that this means that 0, 1, or 2 spikes are to be introduced in the input neuron of the system.

Actually, we consider m input neurons, one for each clause, and in each of them we introduce a sequence of $2n$ digits 0 and 1 (with a spike sent inside in the steps corresponding to the occurrence of 1), describing the situation of each variable $x_1, \ldots, x_n$ with respect to the corresponding clause.

For instance, for the formula

$$\gamma = (x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_3),$$

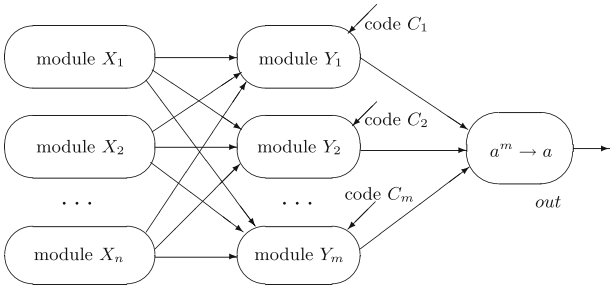


Fig. 7 The structure of the SN P systems solving SAT

we have two input neurons, the first one receiving the spike train 01000111, and the second one receiving the spike train 00110100. Note the important fact that introducing the input takes $2n$ steps, hence the computation cannot last less than $2n$ steps (and, for the construction below, we cannot separate the introduction of the data from the actual computation; later we will improve from the computation duration point of view).

A module X_i exists for each variable x_i, $1 \leq i \leq n$, and a module Y_j is associated with each clause C_j, $1 \leq j \leq m$. Each module X_i has three synapses going to each module Y_j, as precisely specified below.

The structure of every module X_i is suggested in Fig. 8. Actually, neurons $\sigma_{c_0}, \sigma_{c_0'}, \sigma_{c_0''}$, and $\sigma_{c_0'''}$ are common to all modules X_i (they appear only once and have synapses to all the neurons of modules X_i, as indicated in Fig. 8).

These modules non-deterministically produce a truth-assignment for the variables x_1, …, x_n, using the same idea as in the case of `Subset Sum` : the choice between rules $a \to a$; 0 and $a \to a$; 1. Of course, at the price of one further step, the delay can be avoided, as indicated in Fig. 4.

Neuron σ_{c_0} not only feed σ_{d_i}, but also sends—with a certain delay—two spikes to neuron $\sigma_{b_{j,1}}$ from modules Y_j associated with clauses. In turn, neurons $\sigma_{d_i}, \sigma_{e_i}$ send one or no spike to $\sigma_{b_{j,1}}$. No spike is interpreted as the value *false* assigned to x_i, and one spike is interpreted as the value *true* assigned to x_i. Therefore, $\sigma_{b_{j,1}}$ receives either two or three spikes from the module X_i, and these spikes meet in $\sigma_{b_{j,1}}$ the spikes which codify the type of presence of x_i in clause C_j (no occurrence, negated, not negated).

In order to synchronize the check performed in neurons $\sigma_{b_{j,1}}$, i.e., to bring here the truth assignment of variable x_i in the moment when the code of the presence of x_i in C_j arrives in this neuron, we use the delaying neurons labeled generically in Fig. 8 with f and g. No such neurons appear in module X_1, two neurons in each row appear in X_2, in general, $i-1$ couples of neurons appear in each row of module X_i. In this way, a delay of $2(i-1)$ steps is enforced, thus ensuring the synchronization: in steps $1+2i$, all neurons $\sigma_{b_{j,1}}$ receive both the truth assignment of x_i and the code of the way x_i is related with C_j. This last information is produced and processed as indicated in Fig. 9, where the module Y_j is represented.

As one can see from the previous explanations, in steps 3, 5, …, $2n+1$, neurons $\sigma_{b_{j,1}}, 1 \leq j \leq m$, receive a number of spikes as follows:

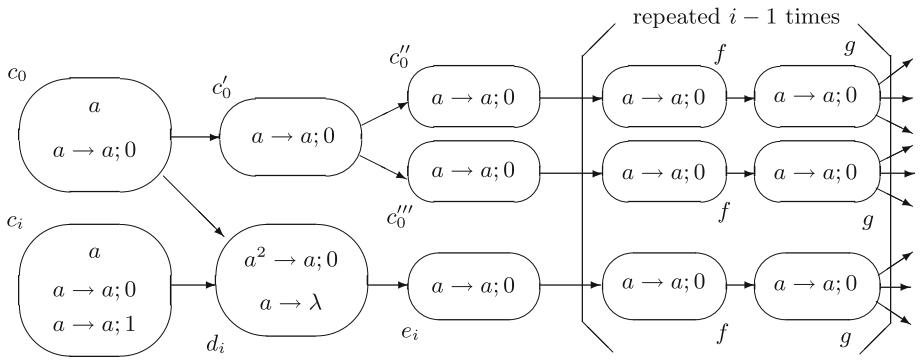
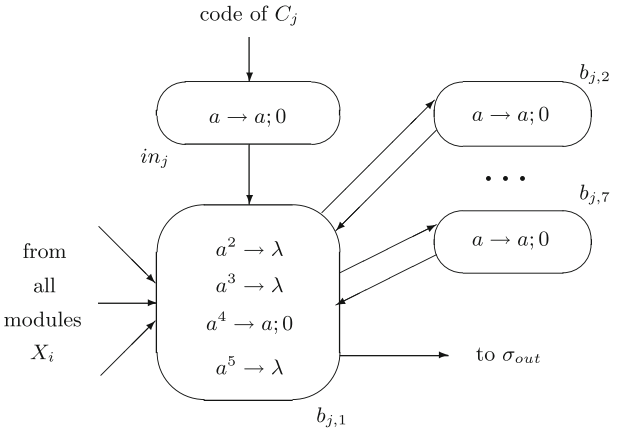


Fig. 8 Module X_i

Fig. 9 Module Y_j



2	if $x_i =$ false and x_i does not appear in C_j,
3	if $x_i =$ true and x_i does not appear in C_j,
3	if $x_i =$ false and x_i appears in C_j,
4	if $x_i =$ true and x_i appears in C_j,
4	if $x_i =$ false and $\neg x_i$ appears in C_j,
5	if $x_i =$ true and $\neg x_i$ appears in C_j.

Thus, the rules of $\sigma_{b_{j,1}}$ produce a spike only in the case when the clause C_j becomes true for the corresponding truth assignment of variable x_i. This spike reaches both the output neuron and the "flooding neurons" $\sigma_{b_{j,2}}, \ldots, \sigma_{b_{j,7}}$, which send six spikes to $\sigma_{b_{j,1}}$ and make this neuron halt. The use of these "flooding neurons" ensures the fact that σ_{out} receives at most one spike from each module Y_j, namely, only if clause C_j has been satisfied. Consequently, the whole system spikes (in step $2n + 2$) only if the truth assignment produced non-deterministically by modules X_i satisfies formula γ.

It should be noted that the number of neurons of the system constructed above is $3n^2 + 8m + 5$, and that the computation lasts a number of steps which is linear in n and independent of m.

The duration of the computation can be made constant at the price of using a larger number of input neurons: instead of introducing the whole description of clause C_j in a neuron, bit by bit, we can use n input neurons for each clause, each of them receiving the two bits which describe the relation between a variable and the clause.

Specifically, for each $j = 1, 2, \ldots, m$ we consider n neurons $\sigma_{j,i}$, $1 \leq i \leq n$; neuron $\sigma_{j,i}$ will receive the spike train 00, 01 (or 10), 11, depending on the fact whether x_i does not appear, appears non-negated in C_j, or appears negated in C_j, respectively. After receiving the spikes which correspond to these codes (0, 1, or 2 spikes), the input neurons just pass the spikes to neurons $\sigma_{b_{j,i}}$, $1 \leq i \leq n, 1 \leq j \leq m$, which behave like neurons $\sigma_{b_{j,1}}$ from Fig. 9: all these neurons contain the rules

$$a^2 \to \lambda, a^3 \to \lambda, a^4 \to a; 0, a^5 \to \lambda,$$

thus checking the truth value of clause C_j with respect to variable x_i. This check is done in parallel for all variables and all clauses, in 3 steps, because the input neurons receive the spikes at the same time. In turn, also the modules X_i are modified, all of them sending the spikes (3 for value *true* and 2 for value *false*) to all neurons $\sigma_{b_{j,i}}$, in two steps, simultaneously.

Three more steps are necessary in order to produce the output. A neuron σ_j is associated with each clause, with synapses coming from all $\sigma_{b_{j,i}}$, $1 \leq i \leq n$, to σ_j. If at least one spike comes along these synapses, this means that clause C_j was satisfied by at least one variable. That is why neuron σ_j contains all rules $a^r \to a; 0$, for $r = 1, 2, \ldots, n$.

Then, all neurons σ_j, $1 \leq j \leq m$, are linked by a synapse to the output neuron σ_{out}, where the rule $a^m \to a; 0$ is present. Therefore, the system spikes in step 6 if and only if a truth assignment was "guessed" which satisfies all the clauses of the propositional formula.

The details of this construction are left to the reader. We only mention that this time the number of neurons is equal to $2nm + 2n + m + 4$. Since in 3-SAT the number of clauses is bounded by $8n^3$ (see below), in this case we can bound the number of neurons with respect to n: it is at most $16n^4 + 8n^3 + 2n + 4$. A more economical solution from this point of view will be given in the next section, based on a different idea: encoding the instance which is introduced in the system.

6 Solving 3-SAT

In this section we focus our attention to 3-SAT, which is defined just like SAT (see Problem 2), the only difference being that now each clause contains exactly three literals. In what follows we will sometimes equivalently say that an instance of 3-SAT is a Boolean formula γ_n, built on n Boolean variables and expressed in conjunctive normal form, with each clause containing exactly three literals.

Note that the number m of clauses appearing in a $\mathtt{SAT}(n, m)$ problem may be very large (e.g., exponential) with respect to n: every variable can be used negated or non-negated, thus obtaining $2n$ literals, and every clause can be seen as the disjunction of a subset of all possible literals (recall that we avoid repetitions of the same literal). Hence the number of all possible clauses is 2^{2n}. The reason for which we are here interested into 3-SAT is that the number of possible 3-clauses which can be built by putting a negated or non-negated variable in each of the three available positions is at most $(2n)^3 = 8n^3$, a polynomial quantity with respect to n. This quantity is obtained by looking at a 3-clause as a triple, and observing that each component of the triple may contain one of the $2n$ possible literals. If we do not allow the repetition of literals in the clauses, then the resulting number of possible clauses becomes $2n \cdot (2n - 1) \cdot (2n - 2)$, which is again $\Theta(n^3)$.

As shown in Garey and Johnson (1979, p. 48), every instance γ of SAT can be transformed in polynomial time (with respect to n and m) into an instance γ' of 3-SAT, in such a way that γ is satisfiable if and only if γ' is satisfiable. However this transformation introduces a new set of variables, whose number is polynomial in m (and thus, possibly, exponential in n).

Figure 10 depicts an SN P system which can be used to solve any instance γ_n of 3-SAT built on n Boolean variables. The system is composed by four layers of neurons. Moving from right to left, in the fourth layer we have a neuron that operates just like an $8n^3$-input AND gate, since it fires if and only if the number of spikes it contains is exactly $8n^3$. These spikes come from the third layer, in which we have one neuron for every possible clause that can be built using n variables. The input to the system is given to these neurons (spikes arrive here in step 2, as explained below). Precisely, each of these neurons can be "selected" by putting in it 4 spikes, or can be ignored (i.e., no spike is inserted in it). In what follows we will also say that these ignored clauses are "unselected". An instance of 3-SAT is thus indicated by selecting the clauses it contains. Since the output neuron in the fourth layer fires if and only if exactly $8n^3$ spikes arrive during the third computation step, the neurons in the third layer will have to be designed in such a way that if they have not been selected then they fire (a somewhat

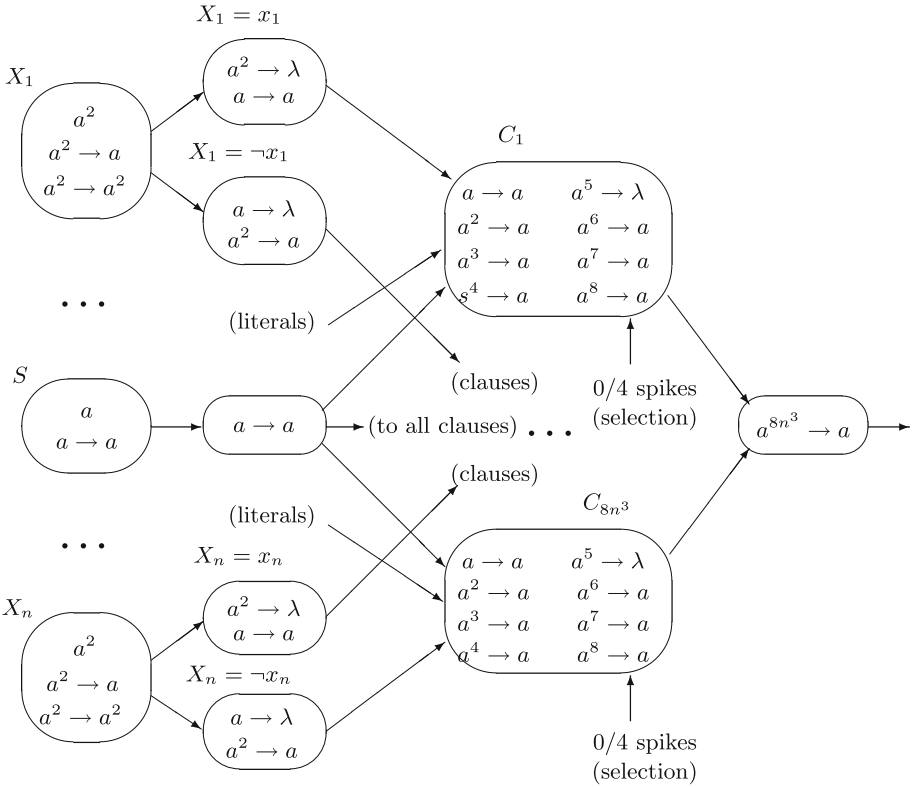


Fig. 10 A non-deterministic SN P system that uniformly solves the 3-SAT problem in constant time

counterintuitive behavior). They will not fire only if they have been selected but the chosen assignment to the variables does not satisfy the corresponding clauses.

The second and the first layer of the system are composed by modules which are similar to those used in the semi-uniform solution of 3-SAT given in Leporati et al. (2007b). Note that these modules use *extended* rules to choose, during the first step of computation, whether to emit one or two spikes; we will remove this requirement below. Every neuron σ_{X_i} in the first layer is connected with its two associated neurons in the second layer, that correspond to the non-negated and to the negated literal which can be built using the Boolean variable x_i, respectively. Instead, the neurons of the second layer are connected with those of the third layer according to what literals appear in each clause. Since we are dealing with 3-SAT, every neuron in the third layer will have exactly three input synapses coming from the second layer.

During the computation, spikes move from the first to the fourth layer, and then one spike is (eventually) expelled to the environment. In the initial configuration, every neuron in the first layer (which is bijectively associated with one of the n variables of the selected instance γ_n) contains two spikes, neuron σ_s contains one spike, and all the other neurons are empty. Note that we are not still considering the mechanism used to select the clauses: such a mechanism is composed by $8n^3$ subsystems, described below, that of course must be initialized at the beginning of the computation. In the first computation step, in each neuron of the first layer it is non-deterministically chosen whether to assign 1 or 0 to the corresponding variable, that is,

whether to assign 1 to the non-negated or to the negated literal. This choice is made by choosing between two rules: one that sends two spikes to the next layer, and one that sends a single spike. In the former case, only the neuron that corresponds to the negated literal will fire during the next computation step; in the latter case, only the neuron that corresponds to the non-negated literal will fire. Since literals are directly connected to the clauses in which they appear, every neuron associated to a clause will receive from 0 to 3 spikes, according to the number of literals of the clause which are satisfied. Thanks to the contribution of neuron σ_s, during step 2 all the neurons of the third layer will receive a further spike. Other 4 spikes will arrive during the same computation step to all those neurons of the third layer which correspond to the clauses which have been selected.

In order to allow the user to select or unselect each possible clause, one copy of the subsystem depicted in Fig. 11 is attached to every neuron of the third layer (neuron σ_{C_j} in the figure). To select the clause, the user puts one spike in the leftmost neuron of the corresponding subsystem before starting the computation of the entire system; the absence of this spike indicates that the clause has not been selected. Note that the user must provide all these spikes simultaneously, that is, in parallel. This is a crucial point to keep in mind when we will consider the execution time of our system.

The core of the system is thus composed by the neurons $\sigma_{C_1}, \ldots, \sigma_{C_{8n^3}}$ of the third layer. As stated above, an unselected neuron should emit one spike, and also a selected neuron that corresponds to a satisfied clause should emit one spike. The neuron should not fire only when it corresponds to a selected clause which is not satisfied. In order to implement this behavior, we can play with the number of spikes contained into the neuron. An unselected neuron will receive from the second layer a number of spikes comprised between 1 and 4. If we add 4 spikes to select a clause, then the corresponding neuron will contain a number of spikes ranging from 5 to 8. The only case in which the neuron does not have to fire is when it is selected but no literals are satisfied, that is, when it contains exactly 5 spikes. Figure 12 illustrates the rules that allow to correctly implement this behavior.

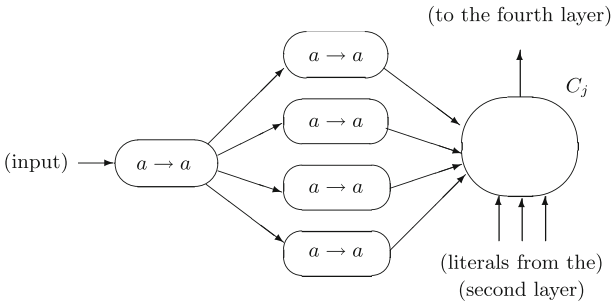


Fig. 11 The subsystem that allows to select a clause by means of a spike given in input

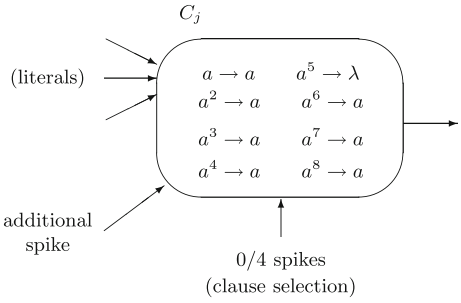


Fig. 12 A neuron that realizes the behavior of a selected/unselected and satisfied/not satisfied clause C_j

The system illustrated in Fig. 10 contains $8n^3 + 3n + 3$ neurons, hence the size of the system is polynomially bounded in n. Also the number of rules and spikes contained into the system during the computations is polynomially bounded. The system solves (in a non-deterministic way) the selected instance of 3-SAT in 4 steps. This last sentence, concerning non-determinism, should be interpreted as follows: either the system "magically" chooses the correct assignment (if it exists) that satisfies all the clauses, exploiting the power of non-determinism, or at least one of the possible computations produces the correct assignment (if it exists). Notice that we are able to keep constant the execution time of the system by forcing the user to provide its input (the set of spikes that indicates which clauses occur in the selected instance γ_n of 3-SAT) in parallel.

Stated otherwise, the user must provide one spike—in the initial configuration of the system—to every subsystem depicted in Fig. 11 that corresponds to a clause that has to be selected. If the user would like to give its input in a sequential way, for example as a bit string of length $8n^3$, where a 1 (resp., 0) in a given position indicates that the corresponding clause has to be selected (resp., ignored), then he should use a sort of sequential to parallel conversion buffer. Such a buffer would read one bit at each computation step, it would produce one spike in the corresponding position of an array when it reads a 1, and finally would release the contents of the array in parallel to the neurons of the third layer of the SN P system depicted in Fig. 10, during the appropriate computation step. This modification would clearly make the computation time of the entire system *linear* with respect to the length of the bit string provided by the user.

One drawback of the proposed system is that it uses extended rules to produce, during the first computation step, one or two spikes in a non-deterministic way. If we want to avoid the use of extended rules, we have at least two possibilities. The first one is depicted in Fig. 13; here the extended rules have been replaced with a non-deterministic choice between a firing and a forgetting rule, also a feature which is missing in the standard definition of SN P systems. A second possibility is to use the module depicted in Fig. 14, where also two clauses are represented to make clear how the connections between the second and the third layer of the system should be made. In this module, the use of extended rules is replaced by the use of delays. Note that neurons σ_1 and σ_2 of Fig. 14 occur only once in the system, and are connected with every neuron of the second and of the third layer, respectively. In particular, neuron σ_1 plays the role that was played by σ_s in Fig. 10; here, however, it also provides one spike to every neuron of the second layer.

If desired, also the delays contained in the subsystem of Fig. 14 can be removed, by complicating a bit the module, as we have done in Fig. 4. The resulting subsystem is depicted in Fig. 15. In this subsystem one or two spikes are produced after four computation steps; these spikes are sent to the neurons that represent the literals, just like it

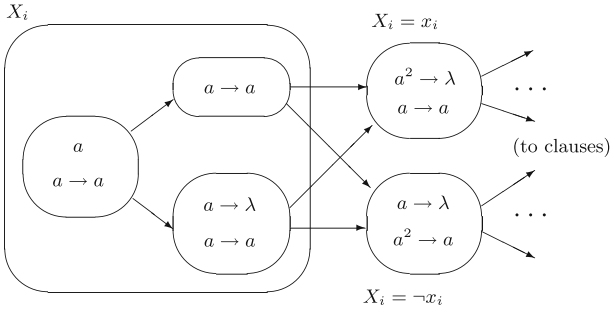


Fig. 13 An alternative way to generate an assignment

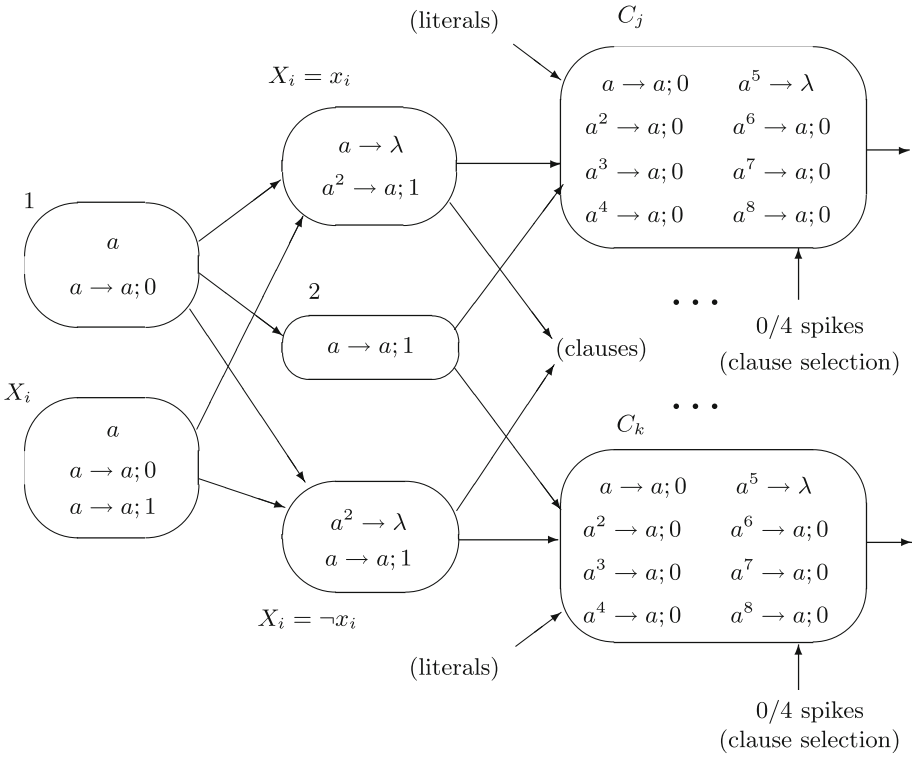


Fig. 14 Another way to generate the assignments. Neurons σ_1 and σ_2 occur only once in the system, and are connected with every neuron of the second and of the third layer, respectively

happened in the first layer of Fig. 10. All subsequent computation steps proceed like before, and thus the resulting system that uniformly solves any instance γ_n of 3-SAT, without using neither extended rules nor delays, operates in 6 steps.

7 Final remarks

Investigations related to the possibility of using SN P systems for solving problems are very recent; in particular, the use of such systems for solving computationally hard problems, an issue of a definite interest, is addressed only in a few papers (besides Leporati et al. 2007a, b), mentioned above, we also mention (Chen et al. 2006a), where a different idea is proposed: start from a pre-computed SN P system, of an arbitrarily large size, but of a rather uniform structure and without spikes inside; the problem is specified by introducing a polynomial number of spikes in certain neurons; the answer, computed in a deterministic way, can be read from the system after a predefined number of computation steps. A way to solve SAT in constant time by means of this model was discussed in Chen et al. (2006a).

The present paper is a contribution to this research direction, with results dealing with the type of the ingredients used in SN P systems which are uniformly constructed for solving, in a non-deterministic manner, Subset Sum and SAT problems. Many open problems were mentioned in the previous sections, most of them dealing with

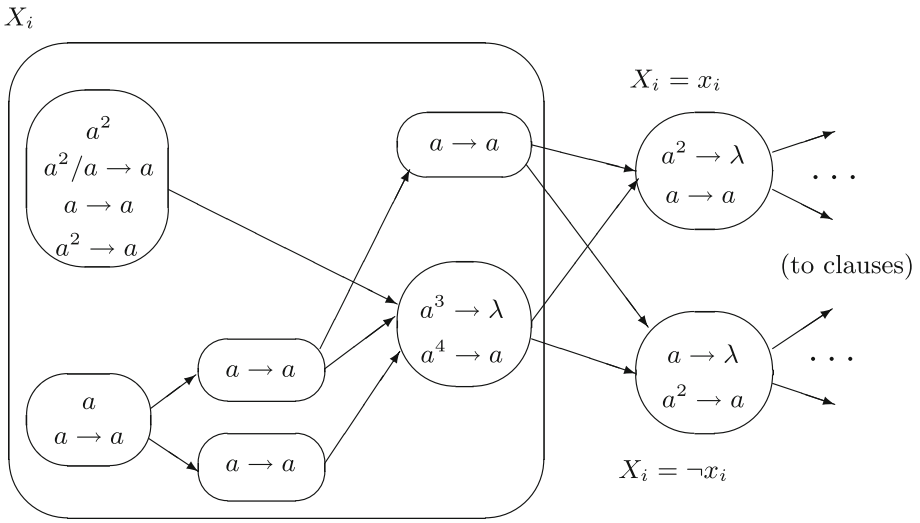


Fig. 15 Another way to generate an assignment. Here we do not use neither extended rules nor delays

improvements of the constructions we have proposed here (for instance, from the point of view of their complexity, the number of neurons or of spikes present in the initial configurations of the systems).

It is important to note that, as proved in Leporati et al. (2007b), an SN P system of polynomial size cannot solve in a deterministic way in a polynomial time an **NP**-complete problem (unless $\mathbf{P} = \mathbf{NP}$), hence efficient solutions to **NP**-complete (or harder) problems cannot be obtained without introducing features which enhance the efficiency (pre-computed resources, ways to exponentially grow the workspace during the computation, non-determinism, and so on). A more careful examination of such features—maybe in relation with the (supposed) way the brain works—is a research direction of a clear interest.

Acknowledgments The first three authors were partially supported by the project "Azioni Integrate Italia-Spagna—Theory and Practice of Membrane Computing" (Acción Integrada Hispano-Italiana HI 2005-0194). The work of the last two authors was supported by the project TIN 2006-13425 from the Ministerio de Educación y Ciencia of Spain, co-financed by FEDER funds, the Excellence project TIC-581 from the Junta de Andalucí a, and the Acción Integrada Hispano-Italiana HI 2005-0194. Gh. Păun was also partially supported by project BioMAT 2-CEx06-11-97/19.09.06.

References

Chen H, Ionescu M, Ishdorj T-O (2006a) On the efficiency of spiking neural P systems. In: *Proceedings of 8th international conference on electronics, information, and communication*, Ulanbator, Mongolia, pp 49–52

Chen H, Ishdorj T-O, Păun Gh, Pérez-Jiménez MJ (2006b) Spiking neural P systems with extended rules. In: Gutiérrez-Naranjo MA et al (eds) Proceedings of fourth brainstorming week on membrane computing, vol I. Fenix Editora, Sevilla, pp 241–265

Garey MR, Johnson DS (1979) Computers and intractability. A guide to the theory on **NP**–Completeness. W.H. Freeman and Company, CA, USA

Ibarra OH, Păun A, Păun Gh, Rodríguez-Patón A, Sosik P, Woodworth S (2007) Normal forms for spiking neural P systems. Theor Comput Sci 372(2–3):196–217

Ionescu M, Păun Gh, Yokomori T (2006) Spiking neural P systems. Fundam Inform 71(2–3):279–308
Ionescu M, Păun Gh, Yokomori T (2007) Spiking neural P systems with an exhaustive use of rules. Int J Unconvent Comput 3(2):135–154
Leporati A, Zandron C, Ferretti C, Mauri G (2007a) Solving numerical NP-complete problems with spiking neural P systems, In: Eleftherakis G, Kefalas P, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing, International Workshop, WMC8, Thessaloniki, Greece, Selected and Invited Papers, LNCS 4860, Springer-Verlag, Berlin, pp 336–352
Leporati A, Zandron C, Ferretti C, Mauri G (2007b) On the computational power of spiking neural P systems. Int J Unconvent Comput, in press
Păun Gh (2002) Membrane computing—an introduction. Springer, Berlin
Păun A, Păun Gh (2007) Small universal spiking neural P systems. BioSystems 90(1):48–60
Păun Gh, Pérez-Jiménez MJ, Rozenberg G (2002) Spike trains in spiking neural P systems. Int J Found Comp Sci 17(4):975–1002