Formal Verification of P Systems with Active Membranes through Model Checking

Florentin Ipate[1], Raluca Lefticaru[1], Ignacio Pérez-Hurtado[2], Mario J. Pérez-Jiménez[2], and Cristina Tudose[1]

[1] Department of Computer Science, University of Pitesti
Str. Targu din Vale 1, 110040, Pitesti, Romania
{florentin.ipate,raluca.lefticaru,cristina.tudose}@upit.ro
[2] Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n. 41012, Sevilla, Spain
{perezh,marper}@us.es

Abstract. Formal verification of P systems using model checking has attracted a significant amount of research in recent years. However, up to now only P systems with static structure have been considered. This paper makes significant advances in this area by considering P systems with active membranes, in particular P systems with division rules. The paper presents a theoretical framework for addressing this problem and reports on a complex case study involving a well-known NP-complete problem solved using P systems with membrane division rules. This is implemented in Promela and non trivial properties are verified using Spin.

1 Introduction

Inspired by the behaviour and functioning of a living cell, *membrane computing* has emerged in recent years as a powerful modelling tool; various applications have been reported [4], especially in biology and bio-medicine, but also in many other areas, such as economics, approximate optimization and computer graphics [21]. Furthermore, software tools, such as P-Lingua [14], for simulating P systems, have been developed and used in real life problems. Naturally, such modelling and simulation tools must be accompanied by appropriate means of formally verifying that the model satisfies the required properties.

One of the most widely used approaches to formal verification is through *model checking*. This uses a model of the implementation, given as an operational specification, and a specification, given as a temporal logic formula, and verifies, on the entire state space of the model, whether the property holds or not. If the property fails, then a counterexample is also returned.

Formal verification of P systems using model-checking has attracted a significant amount of research in recent years, using tools such as Maude [1], PRISM [22], NuSMV [13], Spin [7], [12] or ProB [10]. However, up to now only P systems

with static structure have been considered. Some of the aforementioned investigations consider cell dissolution rules, but in this case a simple flag, indicating whether the membrane exists or not in the current configuration, can be used to address the change in the membrane structure. This is, indeed, a clear weakness of these approaches since, in biology, the membrane structure is not static, but it evolves and changes in time. Furthermore, P systems with active membranes have a wide range of applications; in particular, P systems with division rules are used to devise efficient solutions to computationally hard problems [21].

This paper makes significant advances in the area of model checking based verification of P systems by considering P systems with active membranes. Firstly, it devises a theoretical framework for addressing this problem: it describes the Kripke structure associated with a P system with active membranes and how this can be translated into an executable implementation; it also shows how properties that can be formulated for the P system can be translated into properties of the executable implementation. Secondly, it reports on a complex case study involving a well-known NP-complete problem solved in linear time and in an uniform way using P systems with membrane division rules [17]. The solution was improved in [8], where the total cost is logarithmic in one variable and linear in the rest, but this improvement is not essential in the context of this paper. For this example, a number of non-trivial properties of the model are formulated and verified using the Spin model checker [2].

The paper is structured as follows. We start by presenting in Section 2 the notation and main concepts to be used in the paper. Section 3 presents the theoretical background for our approach. The case study and empirical results are given in the next two sections, while Section 6 discusses related work. Finally, conclusions are drawn in Section 7.

2 Background

2.1 P Systems

Before presenting our approach to P system verification, let us establish the notation used and define the class of cell-like P systems addressed in the paper. Basically, a P system is defined as a hierarchical arrangement of membranes, identifying corresponding regions of the system. Each region has an associated finite multiset of objects and a finite set of rules; both may be empty. Given a finite alphabet $V = \{a_1, ..., a_p\}$, a *multiset* is either denoted by a string $u \in V^*$ (in which the order is not important, the string notation is only used as a convention), or by an associated vector of non-negative integers, $\Psi_V(u) = (|u|_{a_1}, ..., |u|_{a_p})$, where $|u|_{a_i}$ denotes the number of a_i occurrences in u, for each $1 \leq i \leq p$.

The following definition refers to cell-like P systems with active membranes (see [19, 20] or [21] page 284, for details).

Definition 1. *A* P system *is a tuple* $\Pi = (V, H, \mu, w_1, \ldots, w_n, R)$*, where* V *is a finite set, called* alphabet*;* H *is a finite set of labels for membranes;* μ *is a*

membrane structure (a rooted tree), consisting of n membranes injectively labelled with elements of H; w_i, $1 \leq i \leq n$, are strings over V, describing the multisets of objects initially placed in the n regions of μ; R is a finite set of rules, where each rule is of one of the following forms:

(a) $[a \to v]_h^\alpha$, *where* $h \in H$, $\alpha \in \{+, -, 0\}$ *(electrical charges),* $a \in V$ *and* v *is a string over* V *describing a multiset of objects associated with membranes and depending on the label and the charge of the membranes (evolution rules).*

(b) $a[]_h^\alpha \to [b]_h^\beta$, *where* $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in V$ *(send-in communication rules). An object is introduced in the membrane, possibly modified, and the initial charge* α *is changed to* β.

(c) $[a]_h^\alpha \to []_h^\beta b$, *where* $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in V$ *(send-out communication rules). An object is sent out of the membrane, possibly modified, and the initial charge* α *is changed to* β.

(d) $[a]_h^\alpha \to b$, *where* $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in V$ *(dissolution rules). A membrane with a specific charge is dissolved in reaction with an object a (possibly modified).*

(e) $[a]_h^\alpha \to [b]_h^\beta [c]_h^\gamma$, *where* $h \in H$, $\alpha, \beta, \gamma \in \{+, -, 0\}$, $a, b, c \in V$ *(division rules). A membrane is divided into two membranes. The objects inside the membrane are replicated, except for a, that may be modified in each membrane.*

The membrane structure, μ, is denoted by a string of left and right brackets ($[_l$ and $]_l^e$), each with the label l of the membrane; the electrical charge e of each membrane is also given. The environment is only used to send the answer to and we do not capture its structure into this definition.

The rules are applied in maximally parallel mode, which means that they are used in all the regions at the same time and in each region all the objects to which a rule *can* be applied *must* be the subject of a rule application [19]. However, any membrane can be subject of only one rule of types (b) – (e) in one computation step. In type (e) (membrane division) rules, all the contents present before the division, except for object a, can be the subject of rules in parallel with the division. In this case we consider that in a single step two processes take place: first the contents are affected by the rules applied to them, and after that the results are replicated into the two new membranes. If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external membrane which has not been dissolved at that computation step. The skin is never dissolved neither divided. The behaviour of this system is precisely the same as the behaviour mentioned in [15–17], [19–21].

A *configuration* of the P system Π is uniquely identified by the current membrane structure μ' and the contents of each region in μ'. A transition step from a configuration c_1 to a configuration c_2 is realized if the P system can evolve from c_1 to c_2 by using the maximal parallelism mode (as well as the rule restrictions stated above); this is denoted by $c_1 \Longrightarrow c_2$. In the set of all configurations, we will distinguish halting configurations; c is a *halting configuration* if there is no region i such that its contents can be further developed.

2.2 Linear Temporal Logic

The Linear Temporal Logic (LTL) was introduced by Amir Pnueli in 1977 [18] for the verification of computer programs. Compared to CTL (Computation Tree Logic) [5], LTL does not have an existential path quantifier (the **E** of CTL). An LTL formula has to be true over all paths, having the form $\mathbf{A}f$, where f is a path formula in which the only state subformulas permitted are atomic propositions. Given a set of atomic propositions AP, an LTL path formula [5] is either:

- If $p \in AP$, then p is a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$ and $f\mathbf{R}g$ are path formulas, where:
  - The **X** operator ("neXt time", also written $\bigcirc$) requires that a property holds in the next state of the path.
  - The **F** operator ("eventually" or "in the future", also written $\Diamond$) is used to assert that a property will hold at some state on the path.
  - $\mathbf{G}f$ ("always" or "globally", also written $\Box$) specifies that a property, f, holds at every state on the path.
  - $f\mathbf{U}g$ operator (**U** means "until") holds if there is a state on the path where g holds, and at every preceding state on the path, f holds. This operator requires that f has to hold *at least* until g, which holds at the current or a future position.
  - **R** ("release") is the logical dual of the **U** operator. It requires that the second property holds along the path up to and including the first state where the first property holds. However, the first property is not required to hold eventually: if f never becomes true, g must remain true forever.

3 Theoretical Basis for Model Checking of P Systems with Active Membranes

In this section we describe the transformation of a P system with active membranes into a Kripke structure, by extending the approach given in [7] and [11] for P systems with static structure.

Definition 2. *A Kripke structure over a set of atomic propositions AP is a four tuple $M = (S, H, I, L)$, where S is a finite set of states; $H \subseteq S \times S$ is a transition relation that must be left-total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $(s, s') \in H$; $I \subseteq S$ is a set of initial states; $L : S \longrightarrow 2^{AP}$ is an interpretation function, that labels each state with the set of atomic propositions true in that state.*

Usually, the Kripke structure representation of a system results by giving values to every variable in each configuration of the system. Suppose $var_1, \ldots, var_k$ are the system variables, Val_i denotes the set of values for var_i and val_i is a value from Val_i, $1 \leq i \leq k$. Then the states of the system are $S = \{(val_1, \ldots, val_k) \mid val_1 \in Val_1, \ldots, val_k \in Val_k\}$, and the set of atomic predicates are

$AP = \{(var_i = val_i) \mid 1 \leq i \leq k, val_i \in Val_i\}$. Naturally, L will map each state (given by the values of variables) onto the corresponding set of atomic propositions. For convenience, in the sequel the expressions of AP and L will not be explicitly given, the implication being that they are defined as above.

Obviously, in a P system with division rules the number of membranes can grow infinitely. For practical reasons, we will allow only a finite number of membranes and will assume that the upper bound k on this number is known beforehand. However, this is not a limitation because, given a specific P system, this upper bound can be estimated. Then, a configuration of a P system can be represented by a tuple $c = (var_1, \ldots, var_k)$, where var_i holds the current contents u_i of region i (a special symbol not in V will be used when the ith membrane does not exist), the membrane label, its electrical charge and the number of the parent membrane.

Consider a P system $\Pi = (V, H, \mu, w_1, \ldots, w_n, R)$ with $R = \{r_1, \ldots, r_m\}$. The states of the Kripke structure associated with Π will correspond to the configurations of the P system (plus two special states, as explained later). Given two configurations, c and d, there is a transition from c to d if for every membrane i in c there exist $n_1^i, \ldots, n_m^i$, such that the following conditions hold simultaneously:

- At least one rule is applied, i.e. there exists i such that $n_1^i + \ldots + n_m^i > 0$;
- d is obtained from c by applying rules $r_1, \ldots, r_m$, $n_1^i, \ldots, n_m^i$ times, respectively, for every membrane i;
- Any membrane can be subject of only one rule of type (b) – (e), i.e. for every i, there exists at most one j, $1 \leq j \leq m$, such that $n_j^i > 0$ and rule r_j is of type (b) – (e); furthermore, in this case $n_j^i = 1$;
- A computation from c develops in maximally parallel mode, i.e. for every membrane i and every j, $1 \leq j \leq m$, if rules $r_1, \ldots, r_m$ can be applied $n_1^i, \ldots, n_{j-1}^i, (n_j^i + 1), n_{j+1}^i \ldots, n_m^i$ times, respectively, in c then: (1) r_j is of type (b)–(e) and (2) there exists l, $1 \leq l \leq m$ such that r_l is of type (b)–(e) and $n_l^i = 1$.

In order to keep the number of states finite, for each configuration u we will assume that each component of $\Psi_V(u)$ has an established upper bound, denoted Max, and each rule can only be applied for at most a given number of times, denoted Sup (therefore $n_1^i, \ldots, n_m^i \leq Sup$ in the above definition). Whenever (at least) one of these upper bounds is exceeded, extra transitions to a special state, $Crash$, are added. Consequently, as long as the verification is performed for "specific" P systems (and not for a class of P systems), then this upper bound is not a limiting factor. However, the upper bounds and the $Crash$ state are added just to prevent an unexpected behaviour, caused by a P system with a misleading computation.

The halting configurations of the P system (i.e. in which no rule can be applied) are also represented by extra transitions, to another special state, $Halt$. In order for the transition relation to be left-total, loop-back transitions in $Crash$ and $Halt$ are also added.

One additional problem arises when implementing the Kripke structure defined above: most modelling languages (Promela included) do not support the

Table 1. Reformulating the basic LTL operators for the Promela specification of a P system

Property	**LTL specification**
G p	[] (p || !$pInS$)
F p	<> (p && $pInS$)
p U q	(p || !$pInS$) U (q && $pInS$)
X p	X (!$pInS$ U (p && $pInS$))
p R q	(p && $pInS$) V (q || !$pInS$)

existential (or the universal) quantifier. Consequently, a transition involving $\exists$ (as in the Kripke structure representation of a P system) is normally implemented as a sequence of transitions (e.g. a "do - od" loop in Promela) and so additional (intermediary) states are introduced into the model. Naturally, the intermediary states cannot be allowed to form infinite loops and so every possible path in the Promela executable model will contain infinitely often states corresponding to the P system configurations. These assumptions ensure that every path in the P system has at least one corresponding path in the Promela model and vice versa.

Naturally, the properties to be verified (and which refer to the given P system) will need to be reformulated as equivalent formulas for the associated Promela model. Table 1 summarizes the transformations of basic LTL formulas for the Promela implementation, as formally proven in [12] ($pInS$ is a predicate which holds in the original (non-intermediary) states). For example, 'always $b > 0$' (the number of occurrences of b objects is always greater than 0), will become, for the Promela model, 'Globally $b > 0$ or not $pInS$' (we expect $b > 0$ only for configurations corresponding to the P system, but not for the intermediary states).

4 Case Study: The Subset Sum Problem

The Subset Sum problem has been used with some classes of P systems with active membranes in order to illustrate the efficiency and power of these mechanisms for solving NP-complete problems in an uniform way ([8, 9, 15–17]). Most of these solutions follow a similar scenario with a number of stages: generation, calculation, checking and output. This is important for the approach we suggest as we can identify specific properties for some of these stages as well as global ones.

The Subset Sum problem can be formulated as follows:

Given a finite set $A = \{a_1, \ldots, a_n\}$, of n elements, where each element a_i has an associated weight, w_i, and a constant $k \in N$, it is requested to determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$, where $w(B) = \sum_{a_i \in B} w_i$.

For given n and k, a P system with active membranes $\Pi(\langle n,k\rangle)$ is constructed to solve the Subset Sum problem - for details see [17]. The P system is given by

$$\Pi(\langle n,k\rangle) = (\Gamma(\langle n,k\rangle), \{e,s\}, \mu, w_e, w_s, R, i(n,k)), \text{where}$$

- $\Gamma(\langle n,k\rangle) = \{x_0, x_1, \ldots, x_n\} \cup \{\bar{a}_0, \bar{a}, a_0, a, d_1, e_0, \ldots, e_n, q, q_0, \ldots, q_{2k+1}, z_0, \ldots, z_{2n+2k+2}, Yes, No_0, No, \#\}$ is the alphabet;
- $\mu = [_s[_e\,]^0_e]^0_s$ is the membrane structure;
- $w_s = z_0$, $w_e = e_0 a^k$ are the initial multisets;
- $i(n,k) = e$ and contains the code $x_1^{w_1} \ldots x_n^{w_n}$;
- the set of evolution rules, R, consists of
  (1) $[e_i]^0_e \to [q]^-_e [e_i]^+_e$, $0 \le i \le n$;
  $[e_i]^+_e \to [e_{i+1}]^0_e [e_{i+1}]^+_e$, $0 \le i \le n-1$. For each subset of A a membrane is generated.
  (2) $[x_0 \to \bar{a}_0]^0_e$; $[x_0 \to \lambda]^+_e$; $[x_i \to x_{i-1}]^+_e$, for $1 \le i \le n$.
  The code from the input membrane is built in such a way that the multiplicity of x_j represents the weight of $a_j \in A$. These three rules calculates in $\bar{a}_0$ the weight of a subset.
  (3) $[q \to q_0]^-_e$; $[\bar{a}_0 \to a_0]^-_e$; $[\bar{a} \to a]^-_e$.
  These rules mark the beginning of the checking stage; the weight of the subset is now coded by the multiplicity of a_0.
  (4) $[a_0]^-_e \to []^0_e \#$; $[a]^0_e \to []^-_e \#$.
  The number of occurrences of a_0 and a are compared in a checking loop.
  (5) $[q_{2j} \to q_{2j+1}]^-_e, 0 \le j \le k$; $[q_{2j+1} \to q_{2j+2}]^0_e, 0 \le j \le k-1$.
  Objects q_i are utilised as counters of the checking loop.
  (6) $[q_{2k+1}]^-_e \to []^0_e Yes$; $[q_{2k+1}]^0_e \to []^0_e \#$; $[q_{2j+1}]^-_e \to []^-_e \#$; $0 \le j \le k-1$.
  These rules provide an answer to the checking loop given that there are the same number of a_0 and a, more a_0 objects, or more a objects, respectively.
  (7) $[z_i \to z_{i+1}]^0_s, 0 \le i \le 2n+2k+1$; $[z_{2n+2k+2} \to d_1 No_0]^0_s$.
  Objects z_i control the checking stage in all membranes. When the checking stage is over d_1 and No_0 are released into the skin membrane.
  (8) $[d_1]^0_s \to []^+_s d_1$; $[No_0 \to No]^+_s$; $[Yes]^+_s \to []^0_s Yes$; $[No]^+_s \to []^0_s No$.
  In the final stage either Yes or No is sent out into the environment.

5 Experimental Results

In the sequel we will show how different properties of the above P system, providing an answer to an instance of the Subset Sum problem, can be checked. We start with some simpler properties and then will focus on some more complex ones that have been revealed by the current literature [17]. Before starting the presentation of these properties it is important to mention that many of the properties require some transformations or adaptations in order to be checked and validated - as discussed in Section 3.

We have considered different instances of the Subset Sum problem. We have considered a set A with two and three elements, weights between one and three,

Table 2. Some properties verified for the Promela specifications of two P systems, providing an answer to the Subset Sum problem

Property	**LTL specification**	Result 1	Result 2
Generally, there is not YES in the environment.	`[](env.Yes == 0 || !pInS)`	false	true
Generally, there is not NO in the environment.	`[](env.No == 0 || !pInS)`	true	false
Eventually, there is YES in the environment.	`<>(env.Yes == 1 && pInS)`	true	false
Eventually, there is NO in the environment.	`<>(env.No == 1 && pInS)`	false	true
When e_1 appears in a specific membrane 1, with electrical charge 0, then a q will appear with a negative electrical charge.	`[]( ( membr[1].e[1]==1 && membr[1].charge==0 -> <> (membr[1].q==1 && membr[1].charge==-1 && pInS)) || !pInS)`	true	true
When e_1 appears in a specific membrane 1, with electrical charge 0, then a q will appear with a positive electrical charge.	`[]( ( membr[1].e[1]==1 && membr[1].charge==0 -> <> (membr[1].q==1 && membr[1].charge==1 && pInS)) || !pInS)`	false	false
For all i, j, if e_i appears in membrane j, with electrical charge 0, then q will appear in the same membrane with a negative electrical charge.	`[]((i>=0 && i<3 && j>=0 && j<10 && (membr[j].e[i]==1 && membr[j].charge==0 -> <> (membr[j].q==1 && membr[j].charge==-1 && pInS))) ||!pInS)`	true	true

and values of k between two and four. We have tested both cases, with and without solution. In Table 2 we present some properties verified for the Promela specification of the P system. The first two columns express the properties to be verified and the last two columns the results obtained for two instances of the Subset Sum P systems. The first example is $n = 3, k = 4, w = [1, 2, 2]$ and has the associated results in the third column, while the second example is $n = 3, k = 3, w = [2, 2, 2]$ and the model checker answers are given in the last column of Table 2.

We first checked whether there is a solution for a given instance of Subset Sum ("Generally, there is not NO in the environment"). As said above we had to slightly change the query due to the current codification of the P system into Promela and the associated Kripke structure. The current form of the query is $[](env.No == 0 \,||\, !pInS)$, which means that we consider only states associated with the P system and not intermediary ones and check the variable No from

the environment, $env.No$, to be 0. The same behaviour of the system can be checked by "Eventually, there is YES in the environment".

Based on rule of the type "(1)" it can be shown that when e_i appears in a specific e membrane, or more generally, in any such membrane, with electrical charge 0, then in the membrane obtained from it with negative electrical charge, q will appear.

We have also made some experiments running simulations with both P-Lingua and the Promela code. The results obtained were the same (final configuration, number of steps, obtained membranes etc.) and the two simulations have produced values of the same order of magnitude regarding the elapsed time. This shows that the Kripke structure underlying the translation from P-Lingua to Promela does not introduce much overhead into the system.

For the two instances of the Subset Sum P systems considered before, the maximum number of membranes obtained during the P system computation was 16 (corresponding to $n = 3$) and the verification time of each property listed in Table 2 was $7 - 8$ seconds.

In future experiments we will test the limit for which Spin can produce a response in a reasonable time interval and will investigate how our implementation can be improved in order to cope with the well-known state explosion problem associated with model checking. We also aim to address more complex properties as well as to identify invariants of various stages.

6 Related Work

A first approach on P system verification using model checking is introduced in [1]. The authors transform P systems into executable specifications written in Maude, a programming language and the software system developed around it, that uses a rewriting semantics. Further, the LTL model checker of Maude is used to verify some properties of the P systems.

Other papers [6, 7] tackle the decidability problem of P system properties. The models used for experiments are, similarly to those in [1], simple cell-like P systems, generating for example n^2. They present membrane dissolving rules, but no membrane division. The tools used for experimentation are Omega and Spin, the authors conclusion is that Spin is preferable over Omega for model checking P systems.

Another paper that compares the experimental results obtained by two main stream model checkers is [12], which concludes that Spin achieves better performance than NuSMV for P system models. More details regarding the P system transformation into SMV, the language accepted by NuSMV, are given in [11, 13], which use the model checker counterexamples for test generation.

In [22] the probabilistic model checker Prism is employed to verify a stochastic P system. The paper presents a case study representing the cell cycle in eukaryotes, described using a P system specification which is translated into Prism. Specific questions are then formulated and run against the Prism specification of the P system. Daikon, a dynamic invariant detector, has been used in a similar context to extract P systems properties, which were later validated by Prism [3].

The ProB model checker is employed in [10] to verify P systems translated into Event-B.

7 Conclusions and Future Work

This paper makes significant advances in the area of model checking based verification of P systems by considering P systems with active membranes, in particular P systems with cell division rules, having a bounded number of produced membranes. It devises a theoretical framework for model checking of P systems, by extending the previous work on this subject to P systems with active membranes. This is implemented in Spin for a complex case study involving a well-known NP-complete problem. For this example, a number of non-trivial properties of the model are formulated and verified. The case study is very relevant for our approach since division rules may be the source of drastic change in the membrane structure.

In future experiments we aim to address more complex properties of the example considered here and to identify invariants of various stages. More complex instances (for larger n and k) will be tested and improvements of the Promela implementation will be sought in order to deal with the state explosion problem. Future work will also involve automating the process of transforming the P system specification, given as a P-Lingua file, into a Promela model and applying this approach on other challenging, real life examples.

Acknowledgment. The work of FI and RL was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 643/2008. The authors IPH and MPJ acknowledge the support of the project TIN200913192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with Investigador de Reconocida Valía of the Junta de Andalucía, grant P08-TIC-04200. The authors would like to thank the anonymous reviewers for their comments and suggestions that allowed us to improve the paper.

References

1. Andrei, O., Ciobanu, G., Lucanu, D.: Executable Specifications of P Systems. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 126–145. Springer, Heidelberg (2005)
2. Ben-Ari, M.: Principles of the Spin Model Checker. Springer, London (2008)
3. Bernardini, F., Gheorghe, M., Romero-Campero, F.J., Walkinshaw, N.: A Hybrid Approach to Modeling Biological Systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 138–159. Springer, Heidelberg (2007)
4. Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.): Applications of Membrane Computing. Natural Computing Series. Springer, Heidelberg (2006)
5. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (1999)

6. Dang, Z., Ibarra, O.H., Li, C., Xie, G.: On Model-Checking of P Systems. In: Calude, C.S., Dinneen, M.J., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G. (eds.) UC 2005. LNCS, vol. 3699, pp. 82–93. Springer, Heidelberg (2005)
7. Dang, Z., Ibarra, O.H., Li, C., Xie, G.: On the decidability of model-checking for P systems. Journal of Automata, Languages and Combinatorics 11(3), 279–298 (2006)
8. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A Logarithmic Bound for Solving Subset Sum with P Systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 257–270. Springer, Heidelberg (2007)
9. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A fast P system for finding a balanced 2-partition. Soft Computing 9(9), 673–678 (2005)
10. Ipate, F., Ţurcanu, A.: Modelling, verification and testing of P systems using Rodin and ProB. In: Ninth Brainstorming Week on Membrane Computing (BWMC 2011), pp. 209–220 (2011)
11. Ipate, F., Gheorghe, M., Lefticaru, R.: Test generation from P systems using model checking. Journal of Logic and Algebraic Programming 79(6), 350–362 (2010)
12. Ipate, F., Lefticaru, R., Tudose, C.: Formal verification of P systems using Spin. International Journal of Foundations of Computer Science 22(1), 133–142 (2011)
13. Lefticaru, R., Ipate, F., Gheorghe, M.: Model checking based test generation from P systems using P-Lingua. Romanian Journal of Information Science and Technology 13(2), 153–168 (2010)
14. P-Lingua website, `http://www.p-lingua.org` (last visited, November 2011)
15. Pérez-Jiménez, M.J., Jiménez, Á.R., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–285 (2003)
16. Jesús Pérez-Jímenez, M., Riscos-Núñez, A.: A Linear-time Solution to the Knapsack Problem Using P Systems with Active Membranes. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 250–268. Springer, Heidelberg (2004)
17. Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving the Subset-Sum problem by P systems with active membranes. New Generation Computing 23(4), 339–356 (2005)
18. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE (1977)
19. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
20. Păun, G.: Membrane Computing: An Introduction. Springer, Heidelberg (2002)
21. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
22. Romero-Campero, F.J., Gheorghe, M., Bianco, L., Pescini, D., Jesús Pérez-Jímenez, M., Ceterchi, R.: Towards Probabilistic Model Checking on P Systems Using PRISM. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 477–495. Springer, Heidelberg (2006)