

CONTENTS

Special Issue — Natural Computing: Theory and Applications	
Preface	1
<i>R. Freund, M. Gheorghe, S. Marcus, V. Mitrană and M. J. Pérez-Jiménez</i>	
On Strong Reversibility in P Systems and Related Problems	7
<i>O. H. Ibarra</i>	
On String Languages Generated by Spiking Neural P Systems with Anti-Spikes	15
<i>K. Krithivasan, V. P. Metta and D. Garg</i>	
Computation of Ramsey Numbers by P Systems with Active Membranes	29
<i>L. Pan, D. Díaz-Pernil and M. J. Pérez-Jiménez</i>	
P Systems with Proteins on Membranes: A Survey	39
<i>A. Păun, M. Păun, A. Rodríguez-Patón and M. Sidoroff</i>	
On a Partial Affirmative Answer for A Păun's Conjecture	55
<i>I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez, M. A. Gutiérrez-Naranjo and M. Rius-Font</i>	
P Systems with Active Membranes Working in Polynomial Space	65
<i>A. E. Porreca, A. Leporati, G. Mauri and C. Zandron</i>	
On the Power of Families of Recognizer Spiking Neural P Systems	75
<i>P. Sosík, A. Rodríguez-Patón and L. Cienciala</i>	
Modeling Diffusion in a Signal Transduction Pathway: The use of Virtual Volumes in P Systems	89
<i>D. Besozzi, P. Cazzaniga, S. Cocolo, G. Mauri and D. Pescini</i>	
Log-Gain Stoichiometric Stepwise Regression for MP Systems	97
<i>V. Manca and L. Marchetti</i>	
A Simulation Algorithm for Multienvironment Probabilistic P Systems: A Formal Verification	107
<i>M. A. Martínez-del-Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez and F. Sancho-Caparrini</i>	
An Overview on Operational Semantics in Membrane Computing	119
<i>R. Barbuti, A. Maggiolo-Schettini, P. Milazzo and S. Tini</i>	
Formal Verification of P Systems Using Spin	133
<i>F. Ipate, R. Lefticaru and C. Tudose</i>	
Small Universal TVDH and Test Tube Systems	143
<i>A. Alhazov, M. Kogler, M. Margenstern, Y. Rogozhin and S. Verlan</i>	
Filter Position in Networks of Substitution Processors Does Not Matter	155
<i>F. A. Montoro, J. Castellanos, V. Mitrană, E. Santos and J. M. Sempere</i>	
Functions Defined by Reaction Systems	167
<i>A. Ehrenfeucht, M. Main and G. Rozenberg</i>	
P Systems and Topology: Some Suggestions for Research	179
<i>P. Frisco and H. J. Hoogeboom</i>	
An Observer-Based De-Quantisation of Deutsch's Algorithm	191
<i>C. S. Calude, M. Cavaliere and R. Mardare</i>	
PC Grammar Systems with Clusters of Components	203
<i>E. Csuhaj-Varjú, M. Oswald and Gy. Vaszil</i>	
Orthogonal Shuffle on Trajectories	213
<i>M. Daley, L. Kari, S. Seki and P. Sosík</i>	
On the Number of Active Symbols in Lindenmayer Systems	223
<i>J. Dassow and Gy. Vaszil</i>	
Positioned Agents in Eco-Grammar Systems	237
<i>M. Langer and A. Kelemenová</i>	
Morphic Characterizations of Language Families in Terms of Insertion Systems and Star Languages	247
<i>F. Okubo and T. Yokomori</i>	
Power Sums Associated with Certain Recursive Procedures on Words	261
<i>A. Salomaa</i>	
Erratum	273

FORMAL VERIFICATION OF P SYSTEMS USING SPIN

FLORENTIN IPATE, RALUCA LEFTICARU and CRISTINA TUDOSE

*Department of Computer Science, University of Pitesti
Str. Targu din Vale nr. 1, 110040, Pitesti, Romania
florentin.ipate@ifsoft.ro*

Received 21 June 2010

Accepted 29 September 2010

Communicated by Mario J. Pérez-Jiménez

This paper presents an approach to P system verification using the SPIN model checker. It proposes a P system implementation in PROMELA, the modeling language accepted by SPIN. It also provides the theoretical background for transforming the temporal logic properties expressed for the P system into properties of the executable implementation. Furthermore, a comparison between P systems verification using SPIN and NUSMV is realized. The results obtained show that the PROMELA implementation is more adequate, especially for verifying more complex models, such as P systems that model ecosystems.

Keywords: P systems; Kripke structures; model checking; SPIN; PROMELA; NUSMV.

1991 Mathematics Subject Classification: 68Q10, 68Q60

1. Introduction

In the last ten years, a natural computing paradigm, namely *membrane computing*, has emerged as a powerful computational tool [9]. Its models, called *P systems*, have been intensively studied for their theoretical aspects as well as for various applications in biology, concurrency, graphics, and with respect to many interactions with other computational models - brane, ambient and π calculi, Petri nets, cellular automata, grammar systems [4]. Many variants have been introduced and studied, covering deterministic, nondeterministic or probabilistic phenomena. A recent handbook summarizes the most important developments in the membrane computing field [11].

Model checking is an automated technique for verifying if a model meets a given specification, see [5]. It has been applied for checking concurrent systems, models of hardware and software designs. It starts from a model of the implementation, given as an operational specification; it also takes a temporal logic formula and verifies, through the entire state space, whether the property holds or fails. If a property violation is discovered then a counterexample is returned.

SPIN is a model checker widely used in industries that build critical systems and is considered one of the most powerful model checkers available [2]. It is primarily

used for modeling and verifying concurrent systems specified in PROMELA (Process or Protocol Meta Language), a verification modeling language [2].

The decidability of model-checking properties for P systems has been studied [6] and model checking tools have been used to formally verify P systems: NuSMV [7], the MAUDE LTL model checker [1] (using rewriting logic) and PRISM [13] (for stochastic systems). An approach on building test cases for P systems using model checking was also proposed in [8].

This paper makes further advances in the area of model checking based verification of P systems. Firstly, many model checking tools cannot directly implement the transitions of a P system working in maximally parallel mode and so the implementation (on which the model checker operates) is not functionally equivalent to the P system. For this reason, it is important to see how properties that can be formulated for the P system can be translated into properties of the executable implementation. This issue is addressed for properties expressed as LTL (Linear Temporal Logic) formulae. Secondly, the paper reports on a number of P system model checking case studies using SPIN [2], which show significant performance improvements over similar case studies which use another main stream model checker, NuSMV [3].

2. Background

In the rest of the paper, we will use the following notations: V^* for the set of all strings over the alphabet $V = \{a_1, \dots, a_p\}$ and λ to denote the empty string. For a string $u \in V^*$, $|u|_{a_i}$ denotes the number of a_i occurrences in u . Each string u has an associated vector of non-negative integers $(|u|_{a_1}, \dots, |u|_{a_p})$. This is denoted by $\Psi_V(u)$.

2.1. P systems

A basic cell-like P system is defined as a hierarchical arrangement of membranes identifying corresponding regions of the system. Each region has associated a finite multiset of objects and a finite set of rules; both may be empty. A multiset is either denoted by a string $u \in V^*$, where the order is not considered, or by $\Psi_V(u)$. The following definition refers to one of the many variants of P systems, namely cell-like P systems, which uses transformation and communication rules [10]. We will call these processing rules. Since now onwards we will call this model P system.

Definition 1. A P system is a tuple $\Pi = (V, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$, where V is a finite set, called alphabet; μ defines the membrane structure, which is a hierarchical arrangement of n compartments called regions delimited by membranes - these membranes and regions are identified by integers 1 to n ; w_i , $1 \leq i \leq n$, represents the initial multiset occurring in region i ; R_i , $1 \leq i \leq n$, denotes the set of processing rules applied in region i .

The membrane structure, μ , is denoted by a string of left and right brackets ($[_i$, and $]_i$), each with the label of the membrane i , it points to; μ also describes the position of each membrane in the hierarchy. The rules in each region have the form $u \rightarrow (a_1, t_1) \dots (a_m, t_m)$, where u is a multiset of symbols from V , $a_i \in V$, $t_i \in \{in, out, here\}$, $1 \leq i \leq m$. When such a rule is applied to a multiset u in the current region, u is replaced by the symbols a_i with $t_i = here$; symbols a_i with $t_i = out$ are sent to the outer region or outside the system when the current region is the external compartment and symbols a_i with $t_i = in$ are sent into one of the regions contained in the current one, arbitrarily chosen. In the following definitions and examples when the target indication is *here*, the pair $(a_i, here)$ will be replaced by a_i . 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 [9].

A configuration of the P system Π , is a tuple $c = (u_1, \dots, u_n)$, where $u_i \in V^*$, is the multiset associated with region i , $1 \leq i \leq n$. A computation of a configuration c_2 from c_1 using the maximal parallelism mode is denoted by $c_1 \Longrightarrow c_2$. In the set of all configurations we will distinguish terminal configurations; $c = (u_1, \dots, u_n)$ is a *terminal configuration* if there is no region i such that u_i can be further developed.

We say that a rule is *cooperative* if it has at least two objects in its left hand side, e.g. $ab \rightarrow (c, in)(d, out)$. Otherwise, the rule is *non-cooperative*, e.g. $a \rightarrow (c, in)(d, out)$.

2.2. Kripke structures

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; $I \subseteq S$ is a set of initial 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 $(s, s') \in H$; $L : S \rightarrow 2^{AP}$ 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, \dots, var_n are the system variables, Val_i denotes the set of values for var_i and val_i is a value from Val_i , $1 \leq i \leq n$. Then the states of the system are $S = \{(val_1, \dots, val_n) \mid val_1 \in Val_1, \dots, val_n \in Val_n\}$, and the set of atomic predicates are $AP = \{(var_i = val_i) \mid 1 \leq i \leq n, val_i \in Val_i\}$. 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.

A path in a Kripke structure $M = (S, H, I, L)$ from a state $s \in S$ is an infinite sequence of states $\pi = s_0 s_1 \dots$, such that $s_0 = s$ and $(s_i, s_{i+1}) \in H$ for every $i \geq 0$.

2.3. Transforming a P system into a Kripke structure

In this section we outline the transformation of a P system into a Kripke structure; full details may be found in [8].

Consider the one-membrane P system $\Pi = (V, \mu, w, R)$, where $R = \{r_1, \dots, r_m\}$; each rule r_i , $1 \leq i \leq m$, is of the form $u_i \longrightarrow v_i$, where u_i and v_i are multisets over the alphabet V . For simplicity, the discussion is made for one-membrane P systems; a multi-membrane P system can be transformed into a one-membrane P system using an adequate codification.

The states of the associated Kripke structure correspond to the configurations of the P system (plus two special states, as explained later). Essentially, in order to define a transition from configuration u to configuration v we need to be able to specify two properties [6]:

- a computation from u develops in maximally parallel mode by applying rules $r_1, \dots, r_m, n_1, \dots, n_m$ times, respectively;
- v is obtained from u by applying rules $r_1, \dots, r_m, n_1, \dots, n_m$ times, respectively.

In [8] the two properties are formalised using predicates $MaxPar(u, u_1, v_1, n_1, \dots, u_m, v_m, n_m)$ and $Apply(u, v, u_1, v_1, n_1, \dots, u_m, v_m, n_m)$. Then the associated Kripke structure has a transition from u to v if there exist $n_1, \dots, n_m, n_1 + \dots + n_m > 0$ such that predicates $MaxPar$ and $Apply$ hold simultaneously.

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 . Whenever (at least) one of these upper bounds is exceeded, extra transitions to a special state, $Crash$ are added. 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, $Halt$.

3. Kripke Structure Implementation and LTL Formulae

One additional problem arises when implementing the Kripke structure defined above: most modeling languages do not support the 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. an imperative language loop construct) and so additional (intermediary) states are introduced into the model. More details about the PROMELA implementation of the Kripke structure associated with a P system are given later, in Section 4.

Let $M = (S, H, I, L)$ be the Kripke structure (over AP) associated to a P system, as described above. Then its implementation is a Kripke structure $M' = (S', H', I, L')$ (over AP' , $AP \subseteq AP'$) with $S \subseteq S'$ and H' such that for every $s, s' \in S$, $(s, s') \in H$ if and only if there exists s_0, \dots, s_n , $n \geq 1$, with $s_0 = s$ and $s_n = s'$, for which $(s_i, s_{i+1}) \in H'$, $1 \leq i \leq n$. Furthermore, we require that every

path in M' will contain infinitely often states from S (i.e. the intermediary states do not form infinite loops). With this restriction in place, it follows that, for every path π in M there is at least one corresponding path in M' (obtained by suitably adding intermediary states to π) and, vice versa, every path π' in M' gives rise to a path in M (obtained by removing the intermediary states).

From a theoretical point of view, it is important to show that formal verification properties, which are valid on the model M , can be suitably reformulated for its implementation. In what follows we show how LTL formulae on M can be translated into LTL formulae on M' . Let us denote by inS a predicate which holds if and only if M' is in a state from S (normally, inS will be implemented using a boolean variable, and, in any case, inS can be obtained from the atomic propositions AP' using the usual logic connectives, \neg , \vee , \wedge). We assume that the reader is familiar with LTL, for details see for example [12].

Lemma 3. *For every LTL formula f over AP , $M \models Xf$ if and only if $M' \models X(\neg inS \cup (f \wedge inS))$.*

Proof. $' \implies$: Assume $M \models Xf$. Let $\pi' = ws'_1 \dots$ be a path in M' and let $k \geq 1$ be the smallest integer such that $s'_1 \dots s'_{k-1} \in S' \setminus S$ and $s'_k \in S$. Then the corresponding path in M is $\pi = ws'_k \dots$ and since $M \models Xf$ it follows that f holds in s'_k . Thus, since π' is arbitrarily chosen, $M' \models X(\neg inS \cup (f \wedge inS))$. $' \impliedby$: Assume $M' \models X(\neg inS \cup (f \wedge inS))$. Let $\pi = ws_1 \dots$ be a path in M and $\pi' = ws'_1 \dots s'_k s_1 \dots$ a corresponding path in M' . Then f holds in s_1 . Thus, since π is arbitrarily chosen, $M \models Xf$. \square

Lemma 4. *For every LTL formulae f, g over AP , $M \models f \cup g$ if and only if $M' \models (f \vee \neg inS) \cup (g \wedge inS)$.*

Proof. Follows the lines of the proof of Lemma 3. \square

Theorem 5. *For every LTL formula f over AP there exists an LTL formula f' over AP' such that $M \models f$ if and only if $M' \models f'$.*

Proof. Follows from Lemmas 3 and 4 since any LTL formula over AP can be constructed using the usual logic connectives and the temporal operators X and \mathbb{U} .

The transformations of LTL formulae involving the (derived) operators F, G, R are given below. The proofs are left, as a simple exercise, to the reader. For every LTL formulae f, g over AP the following hold:

- $M \models Ff$ if and only if $M' \models F(f \wedge inS)$;
- $M \models f R g$ if and only if $M' \models (f \wedge inS) R (g \vee \neg inS)$;
- $M \models Gf$ if and only if $M' \models G(f \vee \neg inS)$.

4. Case Studies

In this section, we present the main technical details involved in the P system verification using the SPIN model-checker [2]. Furthermore, a comparison with NuSMV is performed, taking into account the verification of different types of LTL specifications, with increasing state spaces.

4.1. Specifying a P system in PROMELA

Consider the one-membrane P system $\Pi = (V, \mu, w, R)$, with the alphabet $V = \{a_1, \dots, a_k\}$ and the set of rules $R = \{r_1, \dots, r_m\}$ (each rule r_i has the form $u_i \rightarrow v_i$, $u_i, v_i \in V^*$). Let $M = (S, H, I, L)$ and $M' = (S', H', I', L')$ be the Kripke structures associated to the P system, as described in Section 3. The PROMELA implementation of the P system will contain:

- k variables, labeled exactly like the objects from V , each one showing the number of occurrences of each object in the membrane, $a_i \in V$, $1 \leq i \leq k$;
- at most k auxiliary variables, labeled like the objects from the alphabet V plus a suffix p , each one showing the number of occurrences of each object a_i , produced in the current computation step;
- m variables n_i , $1 \leq i \leq m$, each one showing the number of applications of each rule $r_i \in R$, $1 \leq i \leq m$;
- one boolean variable $bStateInS$ expressing if the current configuration of M' is a state in S ; the predicate inS from Section 3 will have a corresponding atomic proposition $pInS$ which evaluates whether $bStateInS$ holds;
- one variable $state$ showing the current state of the model, $state \in \{running, halt, crash\}$;
- two constants, Max , the upper bound for the number of occurrences of each object $a_i \in V$, $1 \leq i \leq k$, and Sup , the upper bound for the number of applications of each rule r_i , $1 \leq i \leq m$ (see Section 2.3);
- a set of atomic propositions, which will be used in LTL specifications, introduced by `#define` and named arbitrarily, to increase the readability of the code. For example, `#define pn1 (n1 > 0)` is used to check if the rule r_1 has been applied at least once, `#define pa1 (a == 1)` checks if the number of objects of type a is exactly 1.

We illustrate this approach on the following two one-membrane P systems: $\Pi_1 = (V_1, \mu, w_1, R_1)$, with $V_1 = \{s, a, b, c\}$, $\mu = [1]_1$, $w_1 = s$, $R_1 = \{r_1 : s \rightarrow ab; r_2 : a \rightarrow c; r_3 : b \rightarrow bc; r_4 : b \rightarrow c\}$ and $\Pi_2 = (V_2, \mu, w_2, R_2)$, with $V_2 = \{s, a, b, c, x\}$, $\mu = [1]_1$, $w_2 = s$, $R_2 = \{r_1 : s \rightarrow abcx; r_2 : a \rightarrow ab; r_3 : b \rightarrow bcc; r_4 : x \rightarrow xc\}$. The SMV code for Π_1 is provided in [8] and for comparison the NuSMV and PROMELA code for Π_1, Π_2 and other P systems can be downloaded from http://fmi.upit.ro/evomt/psys/psys_spin.html.

To describe in PROMELA one step of the P system computation, a set of operations, additional variables and intermediary states are needed. As an example,

consider the following code excerpt corresponding to Π_1 , when the current state is *running*:

```

bStateInS = false; n1 = 0; n2 = 0; n3 = 0; n4 = 0; ap = 0; bp = 0; cp = 0;
do
  :: s > 0 -> s = s - 1; n1 = n1 + 1; ap = ap + 1; bp = bp + 1
  :: a > 0 -> a = a - 1; n2 = n2 + 1; cp = cp + 1
  :: b > 0 -> b = b - 1; n3 = n3 + 1; bp = bp + 1; cp = cp + 1
  :: b > 0 -> b = b - 1; n4 = n4 + 1; cp = cp + 1
  :: else -> break
od;
a = a + ap; b = b + bp; c = c + cp;
if
  :: ( a > Max || b > Max || c > Max || s > Max ||
      n1 > Sup || n2 > Sup || n3 > Sup || n4 > Sup ) ->
      state = crash; bStateInS = true
  :: else ->
    if
      :: s == 0 && a == 0 && b == 0 ->
        state = halt; bStateInS = true
      :: else ->
        state = running; bStateInS = true
    fi
fi

```

Inside the *do-od* construct (loop) the four rules are applied non-deterministically, if there are enough objects to be consumed (e.g. $s > 0$ for r_1 , or $b > 0$ for r_3, r_4). The number of objects s, a, b is decreasing as they are consumed each time a rule is applied; the number of objects produced ap, bp, cp and the number of rules applied n_1, \dots, n_4 is increasing. These intermediary steps will terminate when there are no more objects which can be consumed by any rule, in this way ensuring that the computation develops in a maximal parallel manner. The number of objects of each type is updated (thus producing the next configuration) and the next state (*running, halt* or *crash*) is decided.

The parallel application of rules, characteristic to P systems, is simulated sequentially in the proposed implementation, which contains some extra variables and statements. As described in Section 3, to differentiate between states from S and S' , a predicate *inS* is used. It simply evaluates the value of the flag variable $bStateInS \in \{true, false\}$, which is set to false while the intermediary statements are executed and then updated to *true* after successfully finishing one computation step.

4.2. Comparing P system verification with SPIN and NuSMV

Model checking tools face a combinatorial blow up of the state-space, known as the *state explosion problem*. This can occur if the system being verified has many components which can make transitions in parallel [5]. This is the case of P systems,

Table 1. List of LTL properties verified.

P sys.	LTL specifications for SPIN	LTL specifications for NuSMV	T/F
Π_1	X (!pInS U (pa1 && pb1 && pInS))	X (a = 1 & b = 1)	T
Π_1	X (!pInS U (ps1 && pInS))	X (s = 1)	F
Π_1	(pc0 !pInS) U (pc2 && pInS)	(c = 0) U (c = 2)	T
Π_1	(pc0 !pInS) U (pc4 && pInS)	(c = 0) U (c = 4)	F
Π_1	[] (pa0 pa1 !pInS)	G (a = 1 a = 0)	T
Π_1	[] (pa0 !pInS)	G (a = 0)	F
Π_1	<> (pa0 && pc2 && pInS)	F (a = 0 & c = 2)	T
Π_1	<> (pc4 && pInS)	F (c = 4)	F
Π_1	(ps0 && pInS) V (pc0 !pInS)	(s = 0) V (c = 0)	T
Π_1	(ps0 && pInS) V (ps1 !pInS)	(s = 0) V (s = 1)	F
Π_2	[] ((pRunning -> pcMax) !pInS)	G (state = running -> (c <= Max))	T
Π_2	[] (!pHalt !pInS)	G !(state = halt)	T
Π_2	[] ((pb0 -> (pn1_0 && pn2_0)) !pInS)	G ((b = 0) -> (n1 = 0 & n2 = 0))	T
Π_2	[] ((pb2c && pax) !pInS)	G (b*b = c & a = x)	T
Π_2	[] (!pn2 !pInS)	G !(n2 > 0)	F
Π_2	[] ((pRunning) -> X(!pInS U (pRunning && pInS)) !pInS)	G (state = running -> X (state = running))	F
Π_2	<> (pc13 && pInS)	F (c = 13)	F
Π_2	<> (pHalt && pInS)	F (state = halt)	F

Table 2. Comparison of elapsed time for verifying the LTL properties.

P system → Model checker ↓	Π_1					Π_2				
	T_1	T_2	T_3	T_4	T_5	T_1	T_2	T_3	T_4	T_5
NuSMV	< 1	13	94	467	1177 (†)	< 1	12	81	244	691 (†)
SPIN	< 6	< 6	< 6	< 6	< 6	< 4	< 4	< 4	< 4	< 4

Note: The columns T_1, \dots, T_5 represent the time elapsed (seconds) to verify the corresponding sets of LTL specifications, when $Max = Sup \in \{5, 10, 15, 20, 25\}$. For NuSMV, the variable domain was always: $D = [0..N]$, $N = 2 \cdot Max$, e.g. $D_1 = [0..10]$, $D_5 = [0..50]$. For SPIN, the variable domain was always int : $D_1 = \dots = D_5 = [-2^{31}..2^{31} - 1]$.

† = out of memory (and no result returned)

which work in parallel and have a non-deterministic nature, causing the system states to grow exponentially.

The NuSMV model checker is limited to describing finite state machines and only finite data types are provided, such as boolean, enumerations, integer subrange and fixed length array. In our experiments with NuSMV we have used only small subranges such as $[0..10]$ or $[0..50]$. When using smaller domains, NuSMV is successful in verifying the LTL propositions. But for a wider range such as $[0..50]$, NuSMV runs out of memory and no result (true or false) is returned. On the other hand, the SPIN model checker works fine with int domains for variables ($[-2^{31}..2^{31} - 1]$).

Table 2 shows the results obtained by both model checkers for verifying the LTL properties given in Table 1. These LTL properties were written for the implementations of the P systems Π_1 and Π_2 . They were chosen such as to illustrate all the formulae obtained in Section 3, that is all the operators: X (neXt), U (Until), F or $\langle \rangle$ (Finally, eventually), G or $[\]$ (Globally), R or V (Release). Among the formulae provided, half of them were false (see the last column of Table 1) and the model checkers were expected to return a counterexample in this case. As explained in Table 2, the performance of the NUSMV model checker decreases when the domain of the variables is larger.

An extension of the presented approach to multi-membrane P systems with cooperative and transformation-communication rules can be easily obtained. The main modification is using more variables, one for each object type in each membrane (e.g. a_2 represents the number of objects of type a in the membrane labeled 2). For P systems with charges (polarizations) some extra variables are needed to model the charge of each membrane. One benefit of using the SPIN model checker is the capacity of dealing with more complex models, having many variables and larger domains.

A specification of a probabilistic P system, modeling a simplified tritrophic ecosystem (with 10 types of objects, 16 rules), has also been produced. The PROMELA model also includes constraints derived from the probability of each rule used to represent the number of carnivores, herbivores or plants in the population; the upper chosen bounds were $Sup = Max = 10^6$. Due to space constraints the code for this simplified ecosystem is not included, but an extended research report with all the P system models for SPIN and NUSMV can be downloaded from the web page: http://fmi.upit.ro/evomt/psys/psys_spin.html.

5. Conclusions

This paper makes further advances in the area of P system verification. It proposes a P system codification for PROMELA, the modeling language accepted by SPIN, and provides the theoretical background for transforming the temporal logic properties expressed for the P system into properties of the executable implementation. The paper also performs a comparison between P system verification using NUSMV and SPIN, respectively.

Acknowledgments

This work was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 643/2008. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

- [1] O. Andrei, G. Ciobanu and D. Lucanu, Executable specifications of P systems, in *Proc 5th Workshop on Membrane Computing*, eds. G. Mauri et al., Lecture Notes in Computer Science, Vol. 3365 (Springer, 2005) 126–145.

- [2] M. Ben-Ari, *Principles of the SPIN Model Checker* (Springer-Verlag, London, 2008).
- [3] A. Cimatti, E. M. Clarke, F. Giunchiglia and M. Roveri, NUSMV: A new symbolic model checker, *Int. J. Software Tool Tech. Tran.*, **2**(4) (2000) 410–425.
- [4] G. Ciobanu, Gh. Păun and M. J. Pérez-Jiménez (eds.), *Applications of Membrane Computing*, Natural Computing Series (Springer-Verlag, Berlin Heidelberg, 2006).
- [5] E. M. Clarke, O. Grumberg and D. A. Peled, *Model checking* (MIT Press, Cambridge, MA, USA, 1999).
- [6] Z. Dang, O. H. Ibarra, C. Li and G. Xie, On the decidability of model-checking for P systems, *J. Automata, Lang. Combinatorics*, **11**(3) (2006) 279–298.
- [7] M. Gheorghe, F. Ipate, R. Lefticaru and C. Dragomir, An integrated approach to P systems formal verification, in *Proc. 11th Int. Conf. on Membrane Computing*, eds. M. Gheorghe, T. Hinze and Gh. Păun (ProBusiness Verlag, Berlin, 2010) 225–238.
- [8] F. Ipate, M. Gheorghe and R. Lefticaru, Test generation from P systems using model checking, *J. Logic Algebr. Program.* **79**(6) (2010) 350–362.
- [9] Gh. Păun, Computing with membranes, *J. Comput. Syst. Sci.*, **61**(1) (2000) 108–143.
- [10] Gh. Păun, *Membrane Computing: An Introduction* (Springer-Verlag, Berlin, 2002).
- [11] Gh. Păun, G. Rozenberg and A. Salomaa (eds.), *The Oxford Handbook of Membrane Computing* (Oxford University Press, Oxford, UK, 2010)
- [12] A. Pnueli, The temporal logic of programs, in *Proc. of Annual Symposium on Foundations of Computer Science* (IEEE, 1977) 46–57.
- [13] F. J. Romero-Campero, M. Gheorghe, L. Bianco, D. Pescini, M. J. Pérez-Jiménez and R. Ceterchi, Towards probabilistic model checking on P systems using PRISM, in *Proc. 7th Int. Workshop Membrane Computing*, Lecture Notes in Computer Science, Vol. 4361 (Springer, 2006) 477–495.