Towards Automated Verification of P Systems Using Spin

Raluca Lefticaru, Cristina Tudose, and Florentin Ipate

University of Pitesti, Department of Computer Science
Str. Targu din Vale 1, 110040, Pitesti, Romania
name.surname@upit.ro

Summary. This paper presents an approach to P systems verification using the Spin model checker. A tool which implements the proposed approach has been developed and can automatically transform P system specifications from P-Lingua into Promela, the language accepted by the well known model checker Spin. The properties expected for the P system are specified using some *patterns*, representing high level descriptions of frequently asked questions, formulated in *natural language*. These properties are automatically translated into LTL specifications for the Promela model and the Spin model checker is run against them. In case a counterexample is received, the Spin trace is decoded and expressed as a P system computation. The tool has been tested on a number of examples and the results obtained are presented in the paper.

1 Introduction

Membrane computing is a branch of natural computing, inspired from the structure and functioning of the living cell. Its models, called *P systems*, aim to simulate the evolution of a living cell, as well as the interaction or cooperation of cells in tissues, organs, or other types of populations of cells [16, 17]. P systems were introduced in 1998, in a seminal research report of Gheorghe Păun, further published as a journal paper [15].

The new field of membrane computing has known a fast development and many applications have been reported [2], especially in biology and bio-medicine, but also in unexpected directions, such as economics, approximate optimization and computer graphics [17]. Also, a large number of software tools for simulating P systems have been developed, many of them with the purpose of dealing with real world problems, such as those arisen from biology. An overview of the state of the art in P system software can be found in [17], chapter 17. The P-Lingua framework [9], one of the most promising software projects in membrane computing, proposes a new programming language, aiming to become a standard for the representation and simulation of P systems.

Designing a P system to solve a certain real world problem is a difficult task and many simulations are needed to check whether the proposed model behaves as expected. After designing a P system that aims to solve a given problem, a validation is needed, to ensure that the proposed model corresponds to what it is expected. One way to achieve this validation is to formally prove that the P system computations realize the given task. However, the formal proof is somehow hindered by the parallel and non-deterministic nature of the P systems. Consequently, automated tools, such as model checkers, would be very useful to prove or disprove 'on-the-fly' that the P system meets the expected specifications, expressed as temporal logic formulas.

Model checking is an automated technique for verifying if a model meets a given specification [4]. It has been applied for verifying models of hardware and software designs, such as sequential circuits designs, communication protocols, concurrent systems etc. A *model checker* is a tool that receives as input a property expressed as a temporal logic formula and a model of the system, given as an operational specification, and verifies, through the entire state space, whether the property holds or not. If a property violation is discovered then a counterexample is returned, that details why the model does not satisfy the property specified. Two widely used temporal specification languages in model checking are Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [3].

Spin is probably the most well-known LTL model checker [8]. It was written by Gerard Holzmann in the '80, developed over three decades at Bell Laboratories and it received in 2001 the prestigious ACM System Software Award. The transition systems accepted by SPIN (Simple Promela Interpreter) are described in the modelling language Promela (Process Meta Language) and the LTL formulas are checked using the algorithm advocated by Gerth et al. [7]. Spin can also operate as a simulator, following one possible execution path through the system and presenting the resulting execution trace to the user.

In this paper we present an approach to automatic translation of P systems into executable specifications in Promela, the language accepted by the Spin model checker, and its further verification using Spin. The paper intends to realize a bridge between P-Lingua, a very promising framework for defining and simulating P systems, and Spin, one of the most successful model checkers. The tool presented in the paper assists in designing and verifying P systems by automatically transforming the P-Lingua specifications into Promela. The properties expected for the P system are specified in a 'natural language', using an user-friendly interface, then they are automatically translated into LTL specifications for the Promela model; furthermore, the Spin model checker is run against them. In case a counterexample is received, the Spin trace is decoded and expressed in terms of a P system derivation.

Model checking based verification of P systems is a topic which has attracted a significant amount of research in the last years; the main tools used so far are Maude [1], Prism [18], NuSMV [13], Spin [12] and ProB [10]. This paper makes further advances in this area. Firstly, each model checker uses a particular language

for describing the models accepted. The activity of specifying a P system in a certain language, such as Promela for Spin, or SMV for NuSMV, can be tedious and error-prone. Many model checking tools cannot directly implement the transitions of a P system working in maximally parallel mode and consequently, the models obtained are complex, because they are simulating the parallelism using many sequential operations. With this respect, the tool presented here automatically transforms a P system definition file into an executable specification for the Spin model checker. The input file is the P-Lingua specification of a P system, which is an easy way of expressing the P systems and can also be used for simulation with the P-Lingua framework.

Secondly, the executable specifications written for different model checkers are not functionally equivalent with the P systems, for example they can contain extra states and variables corresponding to intermediate steps, which have no correspondence in the P system configurations [12]. For this reason, the P system properties that need to be verified, should be reformulated as properties of the executable implementation. The tool described in this paper takes the properties expressed in a natural language and transforms them into LTL formulas for the Promela model. It hides all the specialized information of the Spin model checker and provides the answer (true or false); in case a counterexample is found, this is decoded and expressed as a P system computation.

The paper is structured as follows. We start by describing the theoretical foundations of this approach in Section 2; the proposed framework is presented in Section 3. Some examples are explained in Section 4, the related work is presented in Section 5 and finally the conclusions are drawn in Section 6.

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. 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 cell-like P systems, with transformation and communication rules [16].

Definition 1. *A* P system *is a tuple* $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$*, 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 $u \rightarrow (a_1, t_1) \ldots (a_m, t_m)$, 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 [15].

A *configuration* of the P system Π, is a tuple $c = (u_1, \ldots, u_n)$, 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; $c = (u_1, \ldots, u_n)$ 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. $a \rightarrow (c, in)(d, out)$. Electrical charges, from the set $\{+; -; 0\}$, can be associated with membranes, as described in [16].

2.2 Linear Temporal Logic

The Linear Temporal Logic (LTL) was introduced by Amir Pnueli in 1977 [14] for the verification of computer programs. Compared to the branching time logic CTL (Computation Tree Logic) [4], 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 [4] 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.
- $f\,\mathbf{R}\,g$ ("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.

2.3 P System Specification in Promela

In this section we will present the theoretical background for verifying P systems using the Spin model checker, as proposed in [12].

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 $(s, s') \in H$; $L : S \longrightarrow 2^{AP}$ is an interpretation function, that labels each state with the set of atomic propositions true in that state.*

In [11] it is explained how an associated Kripke structure can be built for a given P system. For this, the object multiplicities in the P systems membranes have to be restricted to a finite domain and so additional state variables and predicates are defined, following the guidelines from [6]. However, the Spin model checker cannot directly implement this kind of model and a Promela implementation is not functionally equivalent to the P system.

In the following we will present the transformation of a simple P system into a Promela model. For more details, [12] can be consulted and some examples can be downloaded from `http://fmi.upit.ro/evomt/psys/psys_spin.html`. Consider the one-membrane P system $\Pi = (V, \mu, w, R)$, with the alphabet $V = \{a_1, \ldots, a_k\}$ and the set of rules $R = \{r_1, \ldots r_m\}$ (each rule r_i has the form $u_i \rightarrow v_i, u_i, v_i \in V^*$). 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 variable *state* showing the current state of the model, $state \in \{running, halt, crash\}$;
- one boolean variable $bStateInS$ expressing if the current configuration in the Promela model represents a state in the P system; $bStateInS$ is *false* when

intermediary steps are executed and *true* when the computation is over (a step in the P system derivation is completed); a corresponding atomic proposition `pInS` will evaluate whether $bStateInS$ holds;

- 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 $r_i, 1 \leq i \leq m$;
- a set of propositions, which will be used in LTL formulas; they are introduced by `#define` and named suggestively. 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.

In order to describe in Promela one computation step of a P system, a set of operations, additional variables and intermediary states are needed. For example, consider $\Pi_1 = (V_1, \mu_1, w_1, R_1)$, a simple one-membrane P system having $V_1 = \{s, a, b, c\}$, $\mu_1 = [_1]_1$, $w_1 = s$, $R_1 = \{r_1 : s \rightarrow ab; r_2 : a \rightarrow c; r_3 : b \rightarrow bc; r_4 : b \rightarrow c\}$. The following code excerpt corresponds 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
```

The `do-od` loop realizes the non-deterministic application of the rules. It is followed next by an `if` statement deciding the next state from the set $\{halt, crash, running\}$. The code is self-explanatory, for more details [12] can be consulted, so we will focus next on the properties to be checked.

Property	LTL specification
$\mathbf{G}\,p$	`[] (p || !pInS)`
$\mathbf{F}\,p$	`<> (p && pInS)`
$p\,\mathbf{U}\,q$	`(p || !pInS) U (q && pInS)`
$\mathbf{X}\,p$	`X (!pInS U (p && pInS))`
$p\,\mathbf{R}\,q$	`(p && pInS) V (q || !pInS)`
$\mathbf{G}\,(p \rightarrow q)$	`[] (!p || q || !pInS)`
$\mathbf{G}\,(p \rightarrow \mathbf{F}\,q)$	`[] ((p -> <>(q && pInS)) || !pInS)`

Table 1. Reformulating the P system properties for the Promela implementation

2.4 LTL Properties Transformation

The P system semantics is implemented for Spin as a sequence of transitions (or operations) and, consequently, additional intermediary states are introduced into the model. Furthermore, we consider that in the Promela executable model every possible path will contain infinitely often states corresponding to the P system configurations (i.e. the intermediary states do not form infinite loops). From these assumptions, it follows that every path in the P system has at least one corresponding path in the Promela model and vice versa. Furthermore, restrictions on the multiplicity of objects and rules applied are imposed.

The next step needed for model checking P systems with Spin is reformulating the properties to be verified in equivalent formulas for the associated Promela model. For example, a property like 'always $b > 0$' (the number of occurrences of b objects is always greater than 0) should 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 steps).

In Table 1 we summarize the transformations of all LTL formulas for the Promela specification, as they are formally proved in [12].

3 Tool Description

The P system model-checking approach presented has been implemented in a software tool, which can perform 'on-the-fly' verification of properties expressed in a natural language. The tool, as well as the specifications of the P systems used in our tests, can be downloaded from the following web page: `http://fmi.upit.ro/evomt/psys/psys_spin.html`. An advantage of this tool is that the users do not need to be experts in formal verification or to write complex, specialized LTL formulas, in order to apply the model checking verification technique. They only need to specify the P-system, using P-Lingua, to choose the type of property to be checked and the conversion to Promela is performed automatically. If the property is false, the counterexample returned by Spin is parsed

by the tool and represented as a P system computation. A high-level overview of the process is presented in Fig. 1.

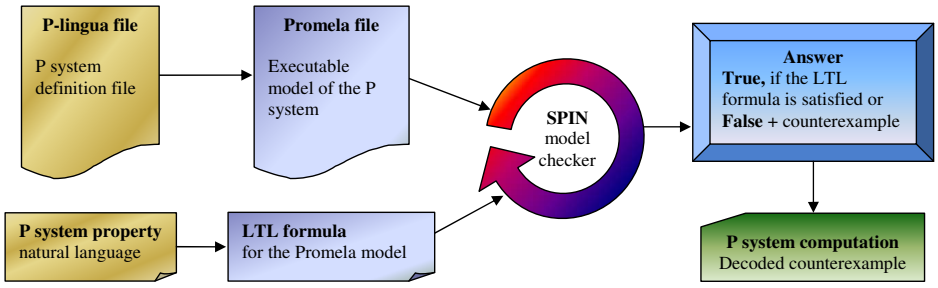

Fig. 1. Tool overview

In order to use the tool, the next steps are required:

- The user specifies the P-system in P-Lingua. He/she can also check, using the P-Lingua parsers, if the P system definition file is syntactically correct.
- The P-system is automatically transformed into a Promela specification.
- The user specifies some basic configuration properties over the system variables.
- The user selects a type of property to be verified, given in natural language, and the basic propositions involved in that property.
- The property is then automatically translated into an LTL formula suitable for the generated Promela model.
- The verification of the P system is performed automatically and if the answer returned by Spin is *false*, the tool will provide a counterexample expressed as a P system computation (with the configurations and set of rules applied at each step).

The main drawback of the application of previous model checking approaches to formal verification of P systems is the difficulty for non-expert users (P systems users) to formulate the appropriate properties in temporal logic. In our case, this could be further amplified by the fact that the LTL formulas would have to be transformed as described earlier. To alleviate this problem, we define some *patterns*, representing high level descriptions of frequently asked questions, formulated in *natural language*. This patterns, which simplify the specification of properties to be verified, are employed in our tool.

Let AP be a set of atomic propositions. An example of such proposition could be: *the number of b objects from a membrane m is greater than* 0. Let $p \in AP$. A basic configuration property ϕ over AP is defined as:

$$\phi ::= p \mid \phi \vee \phi \mid \phi \wedge \phi.$$

Each pattern is given as a natural language phrase that contains one or more basic configuration properties. The patterns considered so far are (in what follows

ϕ and ψ are basic configuration properties, the statement must hold on every computation):

- **Invariance (G ϕ):** a configuration in which ϕ is true must persist indefinitely.
- **Occurrence (F ϕ):** a configuration where ϕ is true will eventually occur.
- **Next occurrence (X ϕ):** a configuration where ϕ is true will occur after initial configuration.
- **Sequence (ϕ U ψ):** a configuration where ϕ is true is reachable and is necessarily preceded all the time by a configuration in which ψ is true.
- **Dual of sequence (ϕ R ψ):** on every computation, along the computation up to and including the first configuration where ϕ is true, in all the configurations ψ holds. However, a configuration where ϕ is true is not required to hold eventually.
- **Consequence (G ($\phi \rightarrow$ F ψ)):** if a configuration where ϕ is true occurs, then a configuration where ψ is true will eventually occur.
- **Instantly consequence (G ($\phi \rightarrow \psi$)):** if a configuration where ϕ is true occurs, then ψ holds also in that configuration.

These patterns can be used to formulate frequently asked questions, without worrying about their transformation into a temporal logic formula suited for our model. The transformation is performed automatically, employing the formulas from Table 1.

4 Case Studies

In this section, we present some examples of P system properties we have verified using the framework introduces previously. In order to simplify the presentation, we consider only one-membrane P systems. The first example is Π_1, the P system defined in Section 2. This is a simple P system, that has been used in several papers. In order to facilitate the comparison with other representations, such as SMV [11] or Event-B [10], Π_1 is presented among the examples on which we illustrate our approach. The other P systems have different properties, that can be verified (e.g. the number of c objects is always the square of b objects), or present polarizations.

4.1 Simple P Systems

Consider the following one-membrane P system: $\Pi_1 = (V_1, \mu_1, w_1, R_1)$, having $V_1 = \{s, a, b, c\}$, $\mu_1 = [_1]_1$, $w_1 = s$, $R_1 = \{r_1 : s \rightarrow ab; r_2 : a \rightarrow c; r_3 : b \rightarrow bc; r_4 : b \rightarrow c\}$. Its corresponding derivation tree is given in Fig. 4.1.

In order to realize the verification of the desired properties, the basic configuration propositions, which are part of the formulas must be specified first, for example:

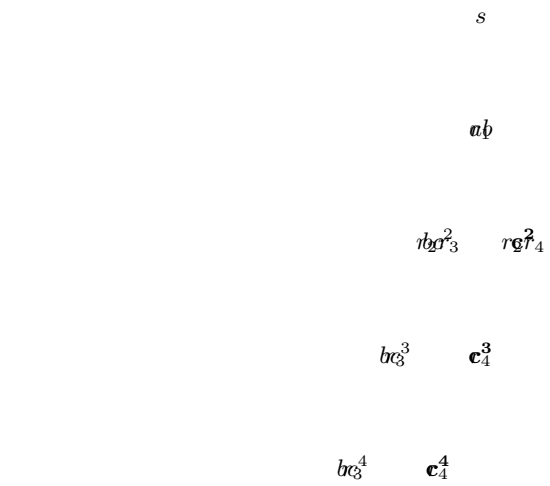

Fig. 2. Derivation tree for Π_1

Prop. ID	Property	Promela proposition
pab1	a==1 && b==1	#define pab1 (a==1 && b==1)
pab0	a==0 || b==0	#define pab0 (a==0 || b==0)
pc0	c==0	#define pc0 (c==0)
pa0c2	a==0 && c==2	#define pa0c2 (a==0 && c==2)
ps0	s==0	#define ps0 (s==0)
pa0	a==0	#define pa0 (a==0)
pa1	a==1	#define pa1 (a==1)
pc2	c==2	#define pc2 (c==2)
p1	s>0	#define p1 (s>0)
p2	c>1	#define p2 (c>1)

The first two columns are introduced by the user and they represent the basic configuration property and its name (or ID). The third column is based on the previous two and it is inserted automatically in the Promela specification. Having the basic proposition defined, more complex properties can be translated from a natural language into LTL formulas and verified using Spin. A set of properties checked for Π_1 is presented in Table 2.

A counterexample is received for every false property, e.g. $s \Longrightarrow ab$ corresponds to the second verified property from Table 2, which states that in the next configuration $a = 0 \wedge b = 0$. Similarly, the third property, expressing the invariance of c objects, $c = 0$, is falsified by $s \Longrightarrow ab \Longrightarrow bc^2$.

Consider the following one-membrane P system: $\Pi_2 = (V_2, \mu_2, w_2, R_2)$, having $V_2 = \{s, a, b, c, x\}$, $\mu_2 = [\,]_1$, $w_2 = s$, $R_2 = \{r_1 : s \to abcx; r_2 : a \to ab; r_3 : b \to bc^2; r_4 : x \to xc\}$. The computation of this P system is $s \Longrightarrow abcx \Longrightarrow ab^2c^4x \Longrightarrow ab^3c^9x \Longrightarrow ab^4c^{16}x \Longrightarrow \ldots$ and it does not halt. It can be easily observed that in every configuration the number of occurrences of c objects is the square of the

Properties	**LTL specifications for Spin**	**Truth**
Next occurrence: `pab1`	`X (!pInS U (pab1 && pInS))`	true
Next occurrence: `pab0`	`X (!pInS U (pab0 && pInS))`	false
Invariance: `pc0`	`[] (pc0 || !pInS)`	false
Occurrence: `pa0c2`	`<> (pa0c2 && pInS)`	true
Dual of sequence: `ps0, pc0`	`(ps0 && pInS) V (pc0 || !pInS)`	true
Sequence: `pc0, pa0`	`((pc0) || !pInS) U ((pa0) && pInS)`	true
Instantly consequence: `pb0, pa0`	`[](!pb0 || pa0 || !pInS)`	true
Consequence: `p1, p2`	`[]((p1 -> <>(p2 && pInS)) || !pInS)`	true

Table 2. Set of properties verified for Π_1.

number of b objects. Some properties, which take into account the value of the variable in the previous configuration, *var_old*, and the current computation *step*, are:

Prop. ID	Property	Promela proposition
`pbc`	`c==b*b`	`#define pbc (c==b*b)`
`p2`	`c-2*b_old-c_old-1==0`	`#define p2 (c-2*b_old-c_old-1==0)`
`pStep1`	`step>=1`	`#define pStep1 (step>=1)`
`pb`	`b==b_old+1`	`#define pb (b==b_old+1)`
`pc16`	`c==16`	`#define pc16 (c==16)`
`px0`	`x==0`	`#define px0 (x==0)`
`px1`	`x==1`	`#define px1 (x==1)`

Properties	**LTL specifications for Spin**	**Truth**
Invariance: `pbc`	`[] (pbc || !pInS)`	true
Occurrence: `pc16`	`<> (pc16 && pInS)`	true
Instantly consequence: `pStep1, pb`	`[] (!pStep1 || pb || !pInS)`	true
Instantly consequence: `pStep1, p2`	`[] (!pStep1 || p2 || !pInS)`	true
Sequence: `px0, px1`	`(px0 || !pInS) U (px1 && pInS)`	true

Table 3. Sample of properties verified for Π_2.

4.2 P Systems with Polarizations

Consider the following P system with charges: $\Pi_3 = (V_3, \mu_3, w_3, R_3)$, having $V_3 = \{a, b, c, d\}$, $\mu_3 = [\,]_1$, $w_3 = a^3$, $R_3 = \{r_1 : [a]_1^- \rightarrow [a, d]_1^0; r_2 : [a]_1^0 \rightarrow [ab]_1^+; r_3 : [a]_1^+ \rightarrow [ac]_1^-\}$. The computation of this P system is $[a^3]_1^0 \Longrightarrow [a^3b^3]_1^+ \Longrightarrow [a^3b^3c^3]_1^- \Longrightarrow [a^3b^3c^3d^3]_1^0 \Longrightarrow [a^3b^6c^3d^3]_1^+ \Longrightarrow [a^3b^6c^6d^3]_1^- \Longrightarrow \ldots$ and it does not halt. It can be easily observed that the number of each object is always a multiple

of 3; also, if the charge is 0, then the number of occurrences of b, c, d is equal. Examples of properties which can be formulated are:

Prop. ID	Property	Promela proposition
pa3	a%3==0	#define pa3 (a%3==0)
pagt0	a>0	#define pagt0 (a>0)
pba	b%a==0	#define pba (b%a==0)
pch0	ch==0	#define pch0 (ch==0)
pch1	ch==1	#define pch1 (ch==1)
pbcd	(b==c && c==d)	#define pbcd ((b==c && c==d))

Properties	**LTL specifications for Spin**	**Truth**
Invariance: pa3	[] ((pa3) || !pInS)	true
Instantly consequence pagt0, pba	[] (!pagt0 || pba || !pInS)	true
Instantly consequence ch0, pbcd	[] (!pch0 || pbcd || !pInS)	true
Instantly consequence ch1, pbcd	[] (!pch1 || pbcd || !pInS)	false

Table 4. Sample of properties verified for Π_2

5 Related Work

A first approach to P system model checking is presented in [1]. The authors use executable specifications written in Maude, a software system supporting rewriting and equational logic, to verify LTL properties of P systems.

The decidability of model-checking properties for P systems has been analysed in [5, 6] and the experiments realized show that Spin is preferable over Omega 'to serve as the back-end solver in a future P system model-checker'.

The probabilistic model checker Prism is employed in [18] to answer specific questions about stochastic P systems.

An approach to P system test generation, based on model checking, is presented in [11, 13] and uses the NuSMV symbolic model checker. This approach is compared with P system model checking using Spin in [12] and the experimental results obtained show that Spin achieves better performance with P system models. A very recent work on P system verification [10] uses the ProB model checker to verify P systems represented in Event-B, a modelling language considered to be an evolution of the B language.

6 Conclusions and Future Work

In this paper, we present a method to automatically verify P systems using the Spin model checker. The theoretical foundations of this approach have been presented

in [12] and its advantages have been shown, in comparison to previous work, that use another main stream model checker, NuSMV [13].

The tool presented in this paper is intended to help designing and verifying P systems by automatically transforming the P-Lingua specifications into Promela, the language accepted by the Spin model checker. The P system properties are specified in a natural language after which they are translated automatically into LTL specifications for the Promela model and then the Spin model checker is run against them. In case a counterexample is received, the Spin trace is decoded and expressed as a P system computation.

Future work consists in extending the tool to accept other classes of P systems, with division and dissolving rules. More experiments will be performed to determine the performance of the Spin model checker for more complex systems, such as those solving SAT problems.

Acknowledgment

This work was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 643/2008.

References

1. Oana Andrei, Gabriel Ciobanu, and Dorel Lucanu. Executable specifications of P systems. In Giancarlo Mauri, Gheorghe Păun, Mario Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 3365 of *Lecture Notes in Computer Science*, pages 126–145. Springer Berlin / Heidelberg, 2005.
2. Gabriel Ciobanu, Mario J. Pérez-Jiménez, and Gheorghe Păun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.
3. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.
4. Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.
5. Zhe Dang, Oscar Ibarra, Cheng Li, and Gaoyan Xie. On model-checking of P systems. In Cristian Calude, Michael Dinneen, Gheorghe Păun, Mario Pérez-Jímenez, and Grzegorz Rozenberg, editors, *Unconventional Computation*, volume 3699 of *Lecture Notes in Computer Science*, pages 82–93. Springer Berlin / Heidelberg, 2005.
6. Zhe Dang, Oscar H. Ibarra, Cheng Li, and Gaoyan Xie. On the decidability of model-checking for P systems. *Journal of Automata, Languages and Combinatorics*, 11(3):279–298, 2006.
7. Rob Gerth, Doron Peled, Moshe Y. Vardi, R. Gerth, Den Dolech Eindhoven, D. Peled, M. Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *In Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.
8. Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.
9. http://www.p lingua.org. The P-lingua website. last visited, April 2011.

10. Florentin Ipate and Adrian Ţurcanu. Modelling, verification and testing of P systems using Rodin and ProB. In *Ninth Brainstorming Week on Membrane Computing (BWMC 2011)*, page this volume, 2011.
11. Florentin Ipate, Marian Gheorghe, and Raluca Lefticaru. Test generation from P systems using model checking. *Journal of Logic and Algebraic Programming*, 79(6):350–362, 2010.
12. Florentin Ipate, Raluca Lefticaru, and Cristina Tudose. Formal verification of P systems using Spin. *International Journal of Foundations of Computer Science*, 22(1):133–142, 2011.
13. Raluca Lefticaru, Florentin Ipate, and Marian Gheorghe. Model checking based test generation from P systems using P-lingua. *Romanian Journal of Information Science and Technology*, 13(2):153–168, 2010. Special issue on membrane computing, devoted to Eighth Brainstorming Week on Membrane Computing (selected and revised papers).
14. Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.
15. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
16. Gheorghe Păun. *Membrane Computing: An Introduction.* Springer-Verlag, 2002.
17. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
18. Francisco José Romero-Campero, Marian Gheorghe, Luca Bianco, Dario Pescini, Mario J. Pérez-Jiménez, and Rodica Ceterchi. Towards probabilistic model checking on P systems using PRISM. In Hendrik Jan Hoogeboom, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing - 7th International Workshop, WMC 2006, Revised, Selected, and Invited Papers*, volume 4361 of *Lecture Notes in Computer Science*, pages 477–495. Springer, 2006.