Modeling, Verification and Testing of P Systems Using Rodin and ProB

Florentin Ipate, Adrian Ţurcanu

Department of Computer Science, University of Pitesti, Romania

Summary. In this paper we present an approach to modelling, verification and testing for cell-like P-systems based on Event-B and the Rodin platform. We present a general framework for modelling P systems using Event-B, which we then use to implement two P-system models in the Rodin platform. For each of the two models, we use the associated Pro-B model checker to verify properties and we present some of the results obtained.

1 Introduction

Membrane computing, the field initiated by Gheorghe Păun [14], studies computing devices, called P systems, inspired by the functioning and structure of the living cell.

In the last years, the research on various programming approaches related to P systems ([6], [16]) and formal semantics ([4], [2], [7]), or with respect to decidability of some model checking properties [5], has created the need for methods for formally verifying and testing such systems.

Formal verification has been studied for different variants of P systems by using rewriting logic and the Maude tool [2] or, for stochastic systems [3], PRISM and associated probabilistic temporal logic [10]. More recently, NuSMV [9] and Spin [13] have been used to verify various properties of transition P systems. Various approaches to building test cases for such P systems have also been proposed [8], [11], [12].

Event-B is a formal modeling language introduced about 10 years ago by J.R. Abrial [1], used for developing mathematical models of complex systems which behave in a discrete fashion. Event-B is an evolution of the B language, one of the most used modeling language in industry since its introduction in the 90s. The efforts for developing Event-B have been supported by two European research projects: RODIN[1], which produced a first platform for Event-B called *Rodin*, and DEPLOY[2], which is currently enhancing this platform based on feedback from

[1] *http://rodin.cs.ncl.ac.uk* - Project running between 2004-2007
[2] *http://deploy-project.eu* - Project running between 2008-2012

its industrial partners (Bosch, SAP, Siemens and Space Systems Finland), which experiment with the latest development of the platform.

The core technology behind Rodin platform is theorem-proving, but also model-checking (ProB) or animation tools (Anim-B) have been integrated as plug-ins.

In this paper we propose a new approach for verifying and testing transition P systems, based on Event-B and its associated model-checker, ProB. Given the industrial support for Event-B and the strength of the Rodin platform, this approach will have an important impact on the practical use of P systems.

The paper is structured as follows. The next section presents general notions about P systems, the Event-B language, the Rodin platform and the model checker Pro-B. In Section 3 we present a general framework for modeling P systems using Event-B. This is then used to implement two P-system models in the Rodin platform: a simple example in Section 4 and a tritrophic ecosystem in Section 5. For each of the two models, we use the associated Pro-B model checker to verify properties and we present the results obtained. Finally, some conclusions and future work are given in Section 6.

2 Background

2.1 P systems

A basic cell-like P system is defined as a hierarchical arrangement of membranes identifying corresponding regions of the system. With each region there are 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 use transformation and communication rules [15]. We will call these processing rules. From now onwards we will refer to this model as simply a P system.

Definition. A *P system* is a tuple $\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n)$, 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 ([, and]), each with the label of the membrane it points to; μ also describes the position of each membrane in the hierarchy.

The rules in each region have the form $u \rightarrow (a_1, t_1)...(a_m, t_m)$, 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 all the symbols $(a_i, here)$ are used as 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 [14].

Electrical charges from the set $\{+, -, 0\}$ can be also associated with membranes, obtaining P systems with polarizations. In this case, with the same notations from the above definition, we can have many types of rules:

- evolution rules, associated with membranes and depending on the label and the charge of the membranes:
  $[u \rightarrow v]_i^p$, $p \in \{+, -, 0\}$, $u \in V, v \in V^*$;
- communication rules, sending an object into a membrane and possibly changing its polarization:
  $u[\,]_i^p \rightarrow [v]_i^{fp}$, $p, fp \in \{+, -, 0\}$, $u, v \in V$;
- communication rules, sending an object out of a membrane and possibly changing the polarization of the membrane:
  $[u]_i^p \rightarrow [\,]_i^{fp} v$, $p, fp \in \{+, -, 0\}$, $u \in V$, $v \in V \cup \{\lambda\}$

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

2.2 Event-B and Rodin

Event-B is based on set theory as its mathematical foundation. The Event-B models are abstract state machines in which transitions between states are implemented as *events*.

An Event-B model is made of several components. Each component can be either a machine or a context. Contexts contain the static structure of the system: sets, constants and axioms. Axioms define the main properties of sets and constants. On the other hand, machines contain the dynamic structure of the system: variables, invariants, and events. Invariants state the properties of variables and events defines the dynamic of the transition system.

An event is a state transition with the following simplified structure:

```
Event eventName
refines <list of refined events (if any) >
when
    grd1 :
        ⋮
    grdn :
then
```

```
    act1 :
        ⋮
    actn :
end
```

Guards (*grd1*, . . . , *grdn*) are necessary conditions for an event to be enabled. They are theorems derivable from invariants, axioms and previously declared guards. An event may have no guards; in this case it is permanently enabled.

Actions (*act1*, . . . , *actn*) describe how the occurrence of an event will modify some of the variables of the machine. All actions of an event are performed at the same time. An action might be either deterministic (using the normal assignment operator :=) or non-deterministic. Non-deterministic actions use the :∈ operator; they have the form $x :\in \{$ *set of possible values* $\}$, in which case an arbitrarily chosen value from the set of possible values is assigned to the variable x.

A very important Event-B concept is refinement, which allows a model to be developed gradually. When the model is finished, a Rodin Platform tool, called Proof Obligation Generator, decides what is to be proved in order to ensure the correctness of the model (e.g. invariant preservation, consistency between original and refined models). Therefore, the proving mechanism provides the guarantee of a formally correct model before the model checker is actually used. This is a big strength of the Rodin platform

2.3 ProB - more than just another model checker

ProB is an animation and model checking tool which accepts B-models, but is also integrated within the Rodin platform. Unlike, most model checking tools, ProB works on higher-level formalisms and so it enables a more convenient modeling.

Properties of an Event-B model can be verified using either the ProB version within the Rodin platform or the standalone version, which offers a greater range of facilities, such as computation of operation coverage or the possibility to find states satisfying a predicate or enabling an operation. When the standalone version is used, the model can be automatically translated in the B language and imported into ProB.

ProB supports automated consistency checking, which can be used to detect various errors in B specifications. The animation facilities allow: to visualize, at any moment, the state space, to execute a given number of operations, to see the shortest trace to current state. Properties that are intended to be verified can be formulated using the LTL or the CTL formalism.

3 The Event-B model of a P system

In this section we present the main ideas about how to build the Event-B model of a P system.

For each object x that appears on the left side of a rule, we introduce a variable xc, representing the number of objects of that type that can be consumed. When the rule is applied, this variable is decreased accordingly. For each object x that appears on the right side of a rule, we introduce a variable xp representing the number of produced objects of that type. When the rule is applied, the variable is incremented. Furthermore, in multi-membrane systems, variables are indexed by membrane numbers.

For each rule we introduce an event. The event is enabled if the rule can be applied and its application modify the state of the system accordingly. A special event, called **actualization**, that is enabled after each step of maximal parallelism, is also needed in order to update the variables before the next computation step.

The initial model can then be refined by adding details about the *state* of the computation. At the beginning of every step of maximal parallelism, the system is considered to be in state *Running*. Another state, *Other*, is considered as an intermediate state between two such steps. A halting configuration is marked by a transition from state *Running* to another state, $Halt$. Finally, in order to keep the number of configurations under control, we can assume that each component of a configuration cannot exceed an established upper bound (denoted MAX) and also that each rule can only be applied for at most a given number of times (denoted SUP). When either of these conditions is violated, we consider that the system performs a transition from *Running* to a fourth state $Crash$. All these states are implemented using a variable, called *state*, with four possible values: *Running*, *Other*, $Halt$ and $Crash$. Obviously, all the events in the original model have to be refined - these now become transitions between states *Running*, and *Other*. Furthermore, new events have to be introduced for transitions from *Running* to $Halt$, *Running* to $Crash$, $Halt$ to $Halt$ and $Crash$ to $Crash$. Obviously, the *state* variable and the extra transitions could have been introduced directly in the original model. However, the use of refinement allows a gradual, more manageable and natural, construction of the model.

4 A simple example

We consider as first example a P system with one membrane and four rules: $\Pi_1 = \{V = \{s, a, b, c\}, []_1, w_1 = s, R = \{r_1 : s \to ab,\ r_2 : a \to c,\ r_3 : b \to bc,\ r_4 : b \to c\}\}$.

We build the corresponding Event-B model in two steps: first, we are interested only of its evolution, then we refine it by introducing the variable **state** presented before.

The first model is just a machine with six variables, all natural numbers (sc, ac, ap, bc, bp, cp) and six events: the initialization event, four events (each of them corresponding to a rule) and the actualization event.

For example, the event corresponding to the first rule and the actualization event are as follows:

Event *rule1*

```
when
    grd1 : $sc > 0$
then
    act1 : $sc := sc - 1$
    act2 : $ap := ap + 1$
    act3 : $bp := bp + 1$
end
```

and respectively,

Event *actualization*

```
when
    grd1 : $sc + ac + bc = 0$
    grd2 : $ap + bp + cp > 0$
then
    act1 : $ac := ap$
    act2 : $bc := bp$
    act3 : $ap := 0$
    act4 : $bp := 0$
end
```

For this model, 20 proof obligations are generated and automatically checked. Using the associated model checker ProB, we verify our formally correct model and we discover that, the sequence of events *rule1, rule2, rule4* leads to the permanent enabling of the actualization event. In order to fix this problem, we refine our model by introducing the variable **state**. Obviously, all events in the original model need to be refined. For example, the refinement of the event *rule1* is as follows:

Event *rule1*
refines *rule1*

```
when
    grd1 : $n1 < SUP$
    grd2 : $sc > 0$
    grd3 : $ac + ap < MAX$
    grd4 : $bc + bp < MAX$
then
    act1 : $n1 := n1 + 1$
    act2 : $sc := sc - 1$
    act3 : $ap := ap + 1$
    act4 : $bp := bp + 1$
    act5 : $state := Other$
end
```

We also introduce new events, corresponding to the transitions between the states of the computation, such as:

Event *RunToCrash*

```
when
    grd1 : $state = Running$
    grd2 : $(s > 0 \wedge n1 = SUP) \vee (ac > 0 \wedge n2 = SUP) \vee (bc > 0 \wedge (n3 =$
        $SUP \vee n4 = SUP)) \vee (ac = MAX) \vee (bc = MAX) \vee (cp = MAX)$
then
    act1 : $state := Crash$
end
```

and

Event *RunToHalt*

```
when
    grd1 : $state = Running$
    grd2 : $sc + ac + bc = 0$
then
    act1 : $state := Halt$
end
```

In this case, there are 60 proof obligations generated, all automatically proven. Using the model checking facilities available in the Rodin platform, we verify the consistency of our model and we find no deadlocks or invariant violation.

Once a formally proven model of the P system is in place, we can use the formalism to verify its properties or to generate tests for certain coverage criteria [12]. The underlying idea of testing using model checkers is to formulate the coverage criterion as a temporal logic formula, negate it and interpret counterexamples (returned by ProB) as test cases. For example, a counterexample for the (negated) LTL formula $G\{not(n4 > 0)\ or\ state = Other\}$ is a test case which covers rule4.

Some examples of properties, their truth values and counterexample returned (for false properties) are given in Table 1.

Note that, in this table, the symbol "/ =" means "$\neq$" and the values for the two constants MAX and SUP were both considered to be 10.

5 Modeling an Ecosystem

We consider now a more complex example: a P system Π_2 with two membranes and electrical charges, which models a tritrophic ecosystem. Its alphabet is $V = \{C, H, P, b, cycle1, cycle2, cycle3, cycle4, g, s\}$, where C stands for carnivores, H for herbivores, P for plants, b for bones, g for garbage and s for volatile substances that attracts carnivores. The initial multiset is P^{100} in the first membrane and H^{300}, C^{10} and $cycle1$ in the second one.

The ecosystem evolves in four cycles:

- Cycle1: Reproduction of plants
  - rule14: $P^2[\ \ ]_2 \rightarrow [P^3]$ some plants reproduce
  - rule15: $P^2[\ \ ]_2 \rightarrow [P^2]$ other plants do not reproduce

LTL Property	Truth Value
$G\{ac \in \{0,1\}\ or\ state = Other\}$	True
$F\{cp > 2\ \&\ state/ = Other\}$	False rule1 rule2 rule4 RunToHalt
$\{cp = 0\ \ or\ \ state = Other\}$ U $\{cp = 2\ \&\ state/ = Other\}$	True
$G\{(state = Running \Rightarrow cp <= MAX)$ or $(state = Other)\}$	True
$G\{not(n4 > 0)\ or\ state = Other\}$	False rule1 rule2 (rule3)^8 rule4
$F\{state = Halt\}$	False rule1 rule2 (rule3)^9 RunToCrash
$G\{((n2 > 0\ \&\ n3 > 0) => cp >= bc)$ $or\ (state = Other)\}$	True

Table 1. Properties checked for Π_1

  - rule7: $[cycle1 \rightarrow cycle2]_2$ transition to cycle2
- Cycle2: Herbivores alimentation
  - rule8: $[HP]_2 \rightarrow +[H^2s]g$ some herbivores feed themselves and produce other herbivores, a volatile substance that attract carnivore and some garbage
  - rule9: $[HP]_2 \rightarrow +[HP]g$ other herbivores do not feed
  - rule10: $[cycle2]_2 \rightarrow +[cycle3]g$ transition to cycle3
- Cycle3: Carnivores alimentation
  - rule11: $+[CHs]_2 \rightarrow -[C^2]g$ some carnivores feed themselves and produce other carnivores and some garbage
  - rule12: $+[CHs]_2 \rightarrow -[CHs]g$ other carnivores do not feed
  - rule13: $+[cycle3]_2 \rightarrow [cycle4]g$ transition to cycle4
- Cycle4: Mortality and reinitialization
  - rule1: $-[P]_2 \rightarrow [\ \ ]P$ all the plants survive to the next cycle
  - rule2: $-[H]_2 \rightarrow [b]g$ some herbivores die
  - rule3: $-[H]_2 \rightarrow [H]g$ others survive to the next cycle
  - rule4: $-[C]_2 \rightarrow [b]g$ some carnivores die
  - rule5: $-[C]_2 \rightarrow [C]g$ others survive to the next cycle
  - rule6: $-[cycle4]_2 \rightarrow [cycle1]g$ reinitialization
  - rule16: $[g \rightarrow \lambda]_1$ elimination of garbage

For clarity of presentation, in this case we build the model in one step, introducing from the beginning the variable **state**, but without the "crash" situation. When a P system uses electrical charges, at each step of maximal parallelism, only rules that have the same initial and final polarization can be applied. In order to implement this requirement, we use two variables **polarization2** and **fp2** for the initial, and, respectively, for the final polarization of the second membrane. Besides the usual polarization values (*plus*, *minus* and *zero*), we introduce an intermediate value, *all*, that we use as an initial value for **fp2**. In the **actualization** event, **fp2** is reset to the value *all* and *polarization*2 receives the value of **fp2**.

As an example, the event corresponding to rule 1 is presented below.

Event *rule1*

```
when
    grd1 : p2c ≥ 1
    grd2 : polarization2 = minus
    grd3 : fp2 = all ∨ fp2 = zero
then
    act1 : p2c := p2c − 1
    act2 : p1p := p1p + 1
    act3 : n1 := n1 + 1
    act4 : fp2 := zero
    act5 : state := Other
end
```

The model checker ProB can then be used to verify ecosystem properties. Table 2 summarizes some of these properties, along with the result produced by the model checker. For all the properties, we considered 5000 as the maximum number of new states.

LTL property	Truth Value
$G\{p2c > 10\ or\ state = Other\}$	False
$F\{c2c = 0\ \&\ state/ = Other\}$	True
$F\{p2c = 0\ \&\ state/ = Other\}$	True
$F\{p2c = 100\ \&\ state/ = Other\}$	False
$F\{(cycle12c = 1\ \&\ state/ = Other) => g1c = 0\}$	True

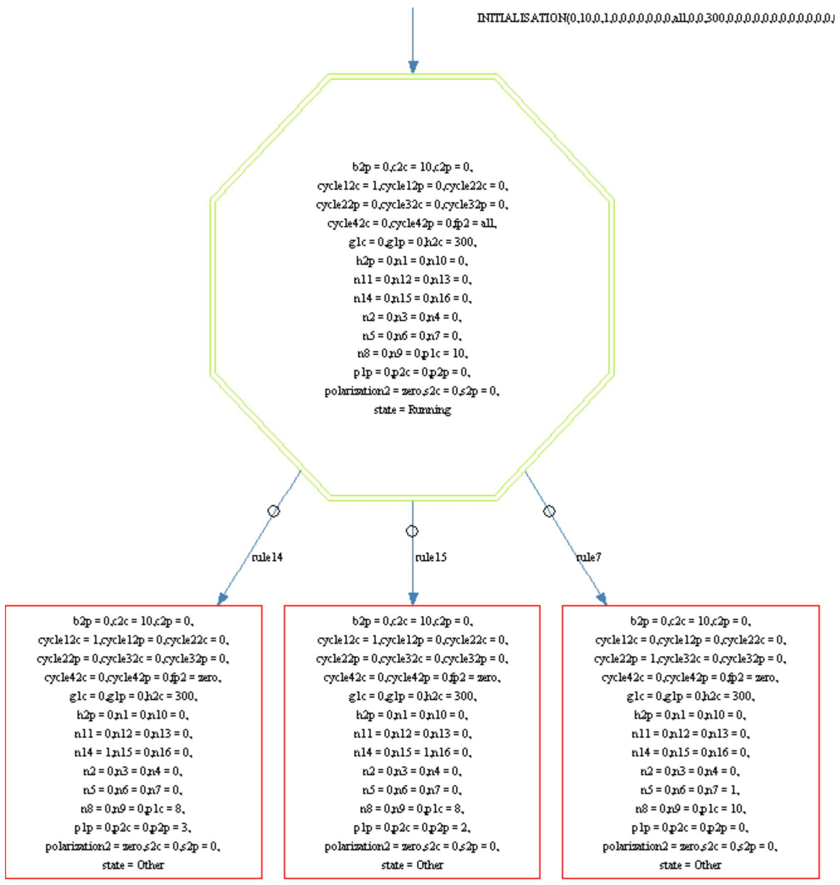
Table 2. Properties checked for the ecosystem Π_2

If we manually execute the initialization event and we choose to see the state space then the result obtained is as shown in Figure 1. Therefore, we have three enabled events - rule14, rule15 and rule7 - and we can see what it happens if we choose each of them.

Another option allows us to verify if the model contains deadlocks or invariant violations. After 20136 new states explored in 93 seconds the coverage analysis shows that the events **RunToHalt** and **HaltToHalt** are not covered.

6 Conclusions and future work

Event-B, Rodin and ProB are not just another modeling language, platform and model checker. Based on rigorous mathematical foundation and allowing high-level modeling, they are strongly supported by the industry.


INITIALISATION(0,10,0,1,0,0,0,0,0,0,0,0,all,0,0,300,0,0,0,0,0,0,0,0,0,0,0,0,0,
rule14
rule15
rule7

```
b2p = 0,c2c = 10,c2p = 0,
cycle12c = 1,cycle12p = 0,cycle22c = 0,
cycle22p = 0,cycle32c = 0,cycle32p = 0,
cycle42c = 0,cycle42p = 0,fp2 = all,
g1c = 0,g1p = 0,h2c = 300,
h2p = 0,n1 = 0,n10 = 0,
n11 = 0,n12 = 0,n13 = 0,
n14 = 0,n15 = 0,n16 = 0,
n2 = 0,n3 = 0,n4 = 0,
n5 = 0,n6 = 0,n7 = 0,
n8 = 0,n9 = 0,p1c = 10,
p1p = 0,p2c = 0,p2p = 0,
polarization2 = zero,s2c = 0,s2p = 0,
state = Running
```

```
b2p = 0,c2c = 10,c2p = 0,
cycle12c = 1,cycle12p = 0,cycle22c = 0,
cycle22p = 0,cycle32c = 0,cycle32p = 0,
cycle42c = 0,cycle42p = 0,fp2 = zero,
g1c = 0,g1p = 0,h2c = 300,
h2p = 0,n1 = 0,n10 = 0,
n11 = 0,n12 = 0,n13 = 0,
n14 = 1,n15 = 0,n16 = 0,
n2 = 0,n3 = 0,n4 = 0,
n5 = 0,n6 = 0,n7 = 0,
n8 = 0,n9 = 0,p1c = 8,
p1p = 0,p2c = 0,p2p = 3,
polarization2 = zero,s2c = 0,s2p = 0,
state = Other
```

```
b2p = 0,c2c = 10,c2p = 0,
cycle12c = 1,cycle12p = 0,cycle22c = 0,
cycle22p = 0,cycle32c = 0,cycle32p = 0,
cycle42c = 0,cycle42p = 0,fp2 = zero,
g1c = 0,g1p = 0,h2c = 300,
h2p = 0,n1 = 0,n10 = 0,
n11 = 0,n12 = 0,n13 = 0,
n14 = 0,n15 = 1,n16 = 0,
n2 = 0,n3 = 0,n4 = 0,
n5 = 0,n6 = 0,n7 = 0,
n8 = 0,n9 = 0,p1c = 8,
p1p = 0,p2c = 0,p2p = 2,
polarization2 = zero,s2c = 0,s2p = 0,
state = Other
```

```
b2p = 0,c2c = 10,c2p = 0,
cycle12c = 0,cycle12p = 0,cycle22c = 0,
cycle22p = 1,cycle32c = 0,cycle32p = 0,
cycle42c = 0,cycle42p = 0,fp2 = zero,
g1c = 0,g1p = 0,h2c = 300,
h2p = 0,n1 = 0,n10 = 0,
n11 = 0,n12 = 0,n13 = 0,
n14 = 0,n15 = 0,n16 = 0,
n2 = 0,n3 = 0,n4 = 0,
n5 = 0,n6 = 0,n7 = 1,
n8 = 0,n9 = 0,p1c = 10,
p1p = 0,p2c = 0,p2p = 0,
polarization2 = zero,s2c = 0,s2p = 0,
state = Other
```

Fig. 1. The state space shown by ProB

In this paper we have presented a general framework for modeling P systems using Event-B and applied the proposed approach on two examples.

Our future work will concentrate on modeling other types of P systems, refinement, simplification, decomposition of models, as well as applying search based techniques for test generation.

Acknowledgement

Adrian Ţurcanu was supported by project POSDRU -"Dezvoltarea scolilor doctorale prin acordarea de burse tinerilor doctoranzi cu frecventa"- 88/1.5/S/52826 and Florentin Ipate was partially supported by project Deploy, EC-grant no. 214158, and Romanian Research Grant CNCSIS-UEFISCDI no. 7/05.08.2010 at the University of Pitesti.

References

1. Abrial, J.R., Modeling in Event-B. System and software engineering, Cambridge University Press, (2010).
2. Andrei, O., Ciobanu, G., Lucanu, D.: A rewriting logic framework for operational semantics of membrane systems. Theor. Comput. Sci. 373(3), 163–181 (2007)
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.) Membrane Computing - 8th International Workshop, WMC 2007, Revised Selected and Invited Papers. LNCS, vol. 4860, pp. 138–159. Springer (2007)
4. Ciobanu, G.: Semantics of P systems. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, chap. 16, pp. 413–436. Oxford University Press (2010)
5. 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)
6. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Software for P systems. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, chap. 17, pp. 437–454. Oxford University Press (2010)
7. Kleijn, J., Koutny, M.: Petri nets and membrane computing. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, chap. 15, pp. 389–412. Oxford University Press (2010)
8. Gheorghe, M., Ipate, F.: On testing P systems. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 9th International Workshop, WMC 2008, Revised Selected and Invited Papers. LNCS, vol. 5391, pp. 204–216. Springer (2009)
9. Gheorghe, M., Ipate, F., Lefticaru, R., Dragomir, C. , An integrated approach to P systems formal verification, in *Proc. 11th Int. Conf. on Membrane Computing*, eds. M. Gheorghe, T. Hinze and Gh. Păun, 225–238, ProBusiness Verlag, Berlin (2010)
10. Hinton, A., Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer (2006)
11. Ipate, F., Gheorghe, M.: Testing non-deterministic stream X-machine models and P systems. Electronic Notes in Theoretical Computer Science 227, 113–126 (2009)
12. Ipate, F., Gheorghe, M., Lefticaru, R., Test generation from P systems using model checking, *J. Logic Algebr. Program.* 79(6), 350–362 (2010)
13. Ipate, F., Lefticaru, R., Tudose, C., Formal Verification of P Systems Using SPIN, Int. J. Found. Comput. Sci. 22(1): 133-142 (2011)
14. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
15. Păun, G.: Membrane Computing: An Introduction. Springer-Verlag (2002)
16. Serbanuta, T., Stefanescu, G., Rosu, G.: Defining and executing P systems with structured data in K. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 9th International Workshop, WMC 2008, Revised Selected and Invited Papers. LNCS, vol. 5391, pp. 374–393. Springer (2009)