Kernel P Systems: Applications and Implementations

Florentin Ipate, Raluca Lefticaru, Laurenţiu Mierlă, Luis Valencia Cabrera, Huang Han, Gexiang Zhang, Ciprian Dragomir, Mario J. Pérez Jiménez and Marian Gheorghe

Abstract This paper explores the modelling capacities of a new class of P systems, called kernel P systems (kP systems). A specific language for describing kP systems and its translation into Promela, the specification language of Spin, are described.

F. Ipate (✉) · R. Lefticaru
Faculty of Mathematics and Computer Science, University of Bucharest, Academiei 14, Bucharest, Romania
e-mail: florentin.ipate@ifsoft.ro

R. Lefticaru
e-mail: raluca.lefticaru@gmail.com

F. Ipate · R. Lefticaru · L. Mierlă · M. Gheorghe
Faculty of Mathematics and Computer Science, University of Piteşti, Târgu din Vale 1, Piteşti, Romania
e-mail: laurentiu.mierla@gmail.com

M. Gheorghe
e-mail: m.gheorghe@sheffield.ac.uk

L. V. Cabrera · M. J. P. Jiménez
Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla Avda, Reina Mercedes s/n 41012 Sevilla, Spain
e-mail: lvalencia@us.es

M. J. P. Jiménez
e-mail: marper@us.es

H. Han · G. Zhang
School of Electrical Engineering, Southwest Jiaotong University, Chengdu 610031, People's Republic of China
e-mail: 329298217@qq.com

G. Zhang
e-mail: zhgxdylan@126.com

C. Dragomir · M. Gheorghe
Department of Computer Science, University of Sheffield, Portobello Street, Regent Court, Sheffield S1 4DP, UK
e-mail: c.dragomir@sheffield.ac.uk

This Promela specification has been further used for simulation and property verification with the Spin model checker. Also, a parallel implementation on GPU parallel architectures, realized using CUDA, is presented and the results are compared with the ones obtained using Promela and Spin. A case study, namely the Subset sum problem, which has been modelled with kernel P systems and further implemented in Promela is presented.

Keywords Membrane computing · Kernel P systems · Subset sum problem · Formal verification

1 Introduction

Membrane computing, initiated in the seminal work of Păun [1], is a new branch of natural computing, whose models, *membrane systems* or *P systems*, are inspired from the structure and the functioning of the living cell and of other more complex systems, like tissues, organs or populations of cells [2]. In the last years, many P system models have been proposed and important developments have been reported [3], concerning the computational power of different variants, their capabilities to solve NP-complete problems, decidability or complexity aspects. Furthermore, these distributed parallel computing models have been successfully used to model, simulate and formally verify various systems [3, 4].

Kernel P systems (*kP systems*, for short) represent a new class of membrane systems, recently proposed in [5]. They use well-known features of P systems and also include new elements, in order to provide a coherent and comprehensive description of various classes of P systems and allow the possibility of integrating different types of P systems into the same formalism. For example, neural-like P systems and P systems with active membranes have been simulated by the newly introduced kP systems [5]. Other initial case studies, applying kP systems to solve the 3-colouring problem, previously solved with tissue P systems, have been presented in [6, 7] and analysed in terms of efficiency and complexity.

The expressive power of kP systems is further investigated in this paper and their modelling capacity is illustrated on a new case study, representing another NP-complete problem. Furthermore, compared with [6], additional modalities to implement kP systems are given and the model checker Spin [8] has been used to simulate and verify properties expressed for the Promela model. A parallel implementation of P systems using GPU and CUDA is also presented, and some initial results are reported.

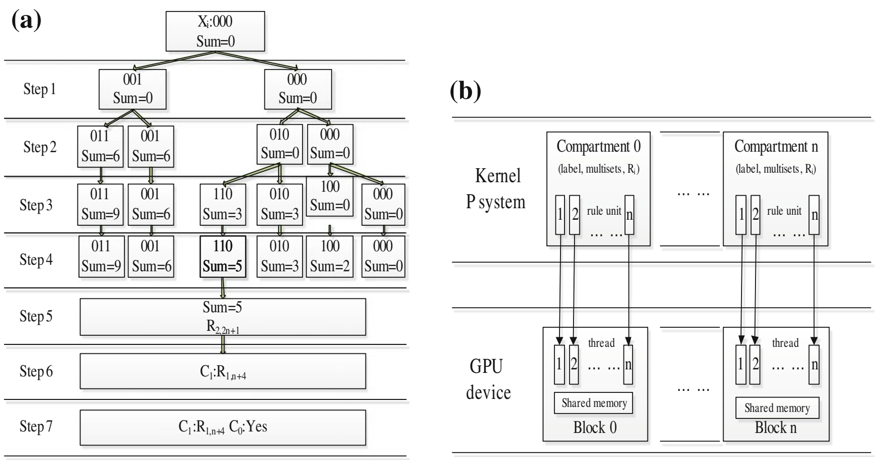

Fig. 1 **a** The procedure of the kernel P system for solving the subset sum for $n = 3$. **b** Basic design of the simulation

2 Background

In this section we formally introduce a particular type of *kernel P systems*, namely *simple kernel P systems*, or *skP systems* for short [6]; a description of the general *kP system* model is provided in [5]. The skP systems have a graph-like structure, similar to *tissue P systems*, *compartments*, *multisets of objects* in them and *rules* of two types: (a) *object processing* and (b) *system structure* rules. The kP system rules have *guards*, with activators and inhibitors, described in a more general way than in traditional P systems. The execution strategy applied in such systems will be *maximal parallelism*, traditionally used in membrane computing, but, in the context of general kP system formalism, other types of strategies can be defined.

In the rest of the paper we use the following notations. For an *alphabet* A, the set of all strings over A is denoted by A^*. When the empty string, λ,is not considered, then the set is denoted by A^+. If u and v are strings over A, then their *concatenation* is given by uv. The *length* of the string u is denoted by $|u|$. If $u \in A^*$ and $a \in A$ then $|u|_a$ denote the number of occurrences of symbol a in string u. A *multiset* m over a set A is a pair (A, f) where $f : A \rightarrow \mathbf{N}$ is a mapping. The *support* of the multiset m is defined as $supp(m) = \{x \in A | f(x) > 0\}$. If $m = (A, f)$ is a finite multiset over A and $supp(m) = \{a_1, \ldots a_k\}$, then it will be denoted as $m = \left\{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\right\}$or represented by a string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$.

Definition 1 Let $Rel = \{<, \leq, \neq, \geq, >\}$ be a set of relational operators. Let A be a nonempty finite set and denote $\bar{A} = \{\bar{a} | a \in A\}$. A *multiset over* $A \cup \bar{A}$ *with relational operators from* Rel is an expression $w = \theta_1 a_1^{n_1} \ldots \theta_k a_k^{n_k}$, where $a_1^{n_1} \ldots a_k^{n_k}$ is a string over $A \cup \bar{A}$ (in which a_i and a_j are not necessarily distinct, $1 \leq i < j \leq k$), and $\theta_j \in Rel$, for each j, $1 \leq j \leq k$.

Definition 2 A *guard* g is a finite disjunction of expressions w introduced by Definition 1.

A particular case of a guard g is the empty set. In this case the guard is omitted. The elements of the set A are considered activators and those of $\bar{A}$ inhibitors. For example, a guard $g = <a^3 = \bar{b}$ is true if the current multiset contains at most two objects a and no b, for instance a^2c.

Definition 3 Given two non empty finite sets A (alphabet), L (labels), and $l \in L$, the set of rules associated with l is denoted by R_l. A rule from R_l has one of the following two forms: (**a**) $[x]_l \rightarrow [y_1]_{l_1} \ldots [y_h]_{l_h}\{g\}$, where $x \in A, y_j \in A^*, l_j \in L$ for all $1 \leq j \leq h$, and g is a guard (*h-membrane division* rule); (**b**) $x \rightarrow y\{g\}$, where $x \in A^+$, y is a string over $A \times L$, $y = (a_1, t_1) \ldots (a_h, t_h)$, with $a_j \in A, t_j \in L$ and g is a guard (*rewriting and communication* rule).

Definition 4 Given two nonempty finite sets A, L, a *compartment*, C, is a tuple (l, w_0, R_l), where $l \in L$, w_0 is a multiset over A, and R_l is a set of rules associated with l.

A compartment can be viewed as a region labelled by an element l of L which initially contains a multiset w_0 of objects (elements of A) and such that it has associated a set R_l of rules.

Definition 5 Let r be a rule from R_l, a set of rules associated with a compartment $C = (l, w_0, R_l)$, such that the guard of r is g. The guard g is considered *true* at an instant when the current multiset of compartment C is z if and only if the following happens: (**a**) if $g = \theta_1 a_1^{n_1} \ldots \theta_k a_k^{n_k}$,then for every j, $1 \leq j \leq k$, either (i) if $a_j \in A$ then $|z|_{a_j} \theta_j n_j$ holds, or (ii) if $a_j \in \bar{A}$, then $|z|_{a_j} \theta_j n_j$ does not hold; (**b**) if g is a finite nonempty disjunction of multisets over $A \cup \bar{A}$ with relational operators from *Rel*, $g = w_1 | \ldots | w_p$, then there exists j, $1 \leq j \leq p$, such that w_j is true, according to (*a*); (**c**) if g is the empty set then it is always evaluated true.

Definition 6 A *simple kernel P* (*skP*) system of degree $n \geq 1$ is a tuple, $sk\Pi = (A, L, IO, C_1, \ldots, C_n, \mu, i_0)$ where: A and L are nonempty finite sets; IO is a finite alphabet $IO \subseteq A$; $C_1, \ldots, C_n$ are compartments; $\mu = (V, E)$ is an undirected graph, where $V \subseteq L$ are vertices and E the edges; $i_0 \in L$.

A skP system, $sk\Pi = (A, L, IO, C_1, \ldots, C_n, \mu, i_0)$, can be viewed as a set of n compartments, $C_1, \ldots, C_n$, interconnected by edges from E, of an undirected graph μ. The elements of the set A are called *objects* and the elements of L are called *labels*. *IO* is the alphabet of the *environment objects*. Each compartment is identified by a label of L, has initially a multiset over A, and a finite set of rules. The compartment receiving the result of a computation is denoted by i_0; in the sequel this will always be the *environment*.

A h-membrane division rule $[x]_l \rightarrow [y_1]_{l_1} \ldots [y_h]_{l_h}\{g\}$ associated with a compartment $C = (l, w_0, R_l)$ is applicable at a given instant to the current multiset z, if the guard g is evaluated true with respect to z and the object x is in the multiset z. When applying such a rule to x, the compartment labelled l will be replaced by

h compartments labelled $l_1,\ldots,l_h$ and x is replaced by multiset y_j in compartment l_j; the content of l, but x, after using all the applicable rewriting and communication rules is copied in each of these compartments; all the links of l are inherited by each of the newly created compartments. A rewriting and communication rule $x \rightarrow (a_1, t_1)\ldots(a_h, t_h)\{g\}$ associated with a set of rules, R_l, of a compartment $C = (l, w_0, R_l)$, is applicable at a given moment to the current multiset z, if the guard g is evaluated true, x is contained in z and the target $t_j \in L$, $1 \leq j \leq h$, must be either the label of the current compartment, l, or of an existing neighbour of it $\left((l, t_j) \in E\right)$. When applying the rule, objects a_j are sent to the compartment labelled by t_j, for each j, $1 \leq j \leq h$. If a target, t_j, refers to a label that appears more than once then one of the involved compartments will be non-deterministically chosen. When t_j indicates the label of the environment, then a_j is sent to the environment.

3 Modelling with Kernel P Systems

In order to illustrate the modelling and expressive power of skP systems, as well as their efficiency, the *Subset Sum* problem [9, 10] has been considered. This problem can be formulated as follows: given a finite set V, a function *weight* on V with positive integer values (an additive function), a number k representing the target sum, decide if there exists a subset W of V, such that $weight(W) = k$.

Theorem 1 *The Subset Sum problem for a set with* n *elements can be solved by a skP system and an answer to whether a solution exists or not is obtained in at most* $\mathrm{n} + 3$ *steps using maximum* $2^{\mathrm{n}} + 1$ *compartments.*

Proof A sketch of the proof is provided below. For a given set $V = \{v_1, \ldots, v_n\}$, with $weight(v_i) = k_i$, where k_i is a positive integer, $1 \leq i \leq n$, we build the following skP system, which depends on n, (for checking whether there is a subset, W, with $weight(W) = k$), $sk\Pi(n) = (A, L, IO, \mu, C_1, C_2, 0)$ where

- $A = \{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_n\} \cup \{v_1, \ldots, v_n\} \cup \{v, S, T, F, X, Y, yes, no\}$ where A_i, $1 \leq i \leq n$ are used in the division process of C_2 compartments, until the X element is used to control the end of the process; B_i, $1 \leq i \leq n$ stand for the n elements in set; v_i, $1 \leq i \leq n$ are used to represent the weight of each element, and v the total weight for each compartment; Y is used to trigger the evaluation of weight v against k, generating T (true) or F (false); *yes*, *no* are the two possible answers depending on the objects T, F;
- $L = \{0, 1, 2\}$; 0 is the label of the environment, 1 and 2 of the compartments;
- IO consists of *yes*, *no*; after $n + 3$ steps, one of them will be sent out;
- $C_1 = \left(1, w_{1,0}, R_1\right), C_2 = \left(2, w_{2,0}, R_2\right)$, where $w_{1,0} = S, w_{2,0} = A_1 code(n)$, with $code(n) = v_1^{k_1}\ldots v_n^{k_n}$ being the code of the weights of the elements of V;
- μ is given by the graph with edge $(1, 2)$;

- R_1 contains: $r_{1,1} : S \rightarrow (yes, 0)\{\geq T\}, r_{1,2} : S \rightarrow (no, 0)\{\geq F \geq \overline{T}\}$; $r_{1,\ 1}$ or $r_{1,\ 2}$ sends into the environment the answer *yes* or *no*, respectively;
- R_2 contains: *membrane division rules* $r_{2,i} : [A_i]_2 \rightarrow [B_i A_{i+1}]_2 [A_{i+1}]_2$, $1 \leq i < n, r_{2,n} : [A_n]_2 \rightarrow [B_n X]_2 [X]_2$; these rules generate in n steps all the subsets of V (2^n subsets); each of them being a potential subset W; *rewriting rules* $r_{2,n+i} : v_i \rightarrow v\{= B_i = X\}, 1 \leq i \leq n, r_{2,2n+1} : X \rightarrow Y$ and *rewriting and communication rules* $r_{2,2n+2} : Y \rightarrow (F, 1)\{\neq v^k\}$, $r_{2,2n+3} : Y \rightarrow (T, 1)\{= v^k\}$.

The computation leads to an answer, *yes* or *no*, in $n + 3$ steps.

4 Experimental Evaluation

This section is devoted to skP system simulation and verification of certain properties using the Spin model checker [8]. The skP system is converted into Promela, the specification language for Spin. This is obtained automatically from the skP system specification. A simple specification language allows to express skP system elements and declarations in a simple, concise and intuitive way. The complete implementation of the Subset Sum problem as modelled with kP systems is provided at [11]. Spin [8] is a very popular simulator and model checker tool, purposely built for the formal verification of distributed software systems.

We will refer to an instance of the Subset Sum problem where the number of elements in our set is 10 (the example is available at [11]). Given this parameter, we define a skP system, according to the algorithm illustrated in the previous section, that requires two active compartments, an alphabet of 38 symbols and a total of 25 rules. A successful simulation requires 13 computational steps, the outcome of which is a "*yes*" symbol that is sent to the environment, indicating that there is at least one subset with the sum of its elements equal to the specified value. In the context of membrane computing, Spin has been successfully used to formally verify various types of P systems [12, 13]. The properties we investigate are expressed as LTL formulae and their validity is assessed with respect to our model, an instance of the Subset Sum problem having 10 set elements. The key insight of our approach is the distinction between a P system step and a generic step of a Promela model, which takes into account each atomic instruction. However, a P system's state is reached only after the maximally parallel application of the rules in every compartment. Hence, an auxiliary variable is introduced, indicative of whether the system has reached a state or it is in the process of executing rules. This strategic flag is then mentioned in the LTL properties, instructing Spin to only consider states with relevant configurations (i.e. P system states) in its evaluation. A common property pattern in our study was that of necessary precedence: if a certain condition *invariantPrecond* holds for a state, then another condition *invariantPostcond* must hold in the next state. This is expressed in LTL as: *ltl p5 {[] (!(invariantPrecond)* || *X(!isPsystemStep U(invariantPostcond && isPsystemStep))* || *!isPsystemStep)}*, where *invariantPrecond* and *invariantPostcond* are the two subject conditions and *isPsystemStep* is a Boolean variable which acts as a filter on states, as we are only interested if the property holds for a P system state. A concrete formula that exemplifies this pattern is *ltl p4 {[] (!(ms[1].x[@S] == 1 && ms[1].x[@F] == 1)* || *X(!isPsystemStep U (ms[2].x[@no] == 1 && isPsystemStep))* || *!isPsystemStep)}*, which asserts that if in a certain state the membrane of type "1" contains both an *S* and an *F* object, then, in the next state, the environment will have received a "*no*" object. Another interesting property successfully validated states that if a *Y* symbol has been generated into the compartments of type 2, then, at the next P system step, the multiplicity of the object *T* in the compartment of type 1 will be equal to the number of solutions of the problem. The property is captured by the following formula: *ltl p6 {[] (!(invariantPrecond)* || *X(!isPsystemStep U(solutionCount == TCount && isPsystemStep))* || *!isPsystemStep)}.*

5 Implementation of kP Systems Using GPUs

In order to solve the Subset Sum problem (SSP) using GPUs, we make the following notations. We start with a membrane structure $\mu = [[]_1[]_2]_0$, compartment labels $L = \{0, 1, 2\}$ and symbols $A_1, \ldots, A_{n+4}, B_1, \ldots, B_n, v_1, \ldots, v_n, T, Y, K, V$, *yes*, *no*, where A_i stands for the *i*th step; v_i represents the *i* element of the set; B_i means whether v_i is in the subset or not; object *K* is the desired subset sum; *V* is a generic element counting for the weight of the subset; *Y* is an object which would be changed into a flag and be sent to the compartment C_1 to show the objects in this compartment are a solution and *T* is a flag just mentioned; *yes* and *no* are the two possible answers sent from C_1 to the environment (C_0) at the end of the computation. The compartments are $C_0 = \emptyset$, $C_1 = (1, O_1, R_1)$, $C_2 = (2, O_2, R_2)$, where $O_1 = \{A_1\}, O_2 = \{A_1, v_1^{k_1} \ldots v_n^{k_n}, Y\}$, where v_i is the value of subset *ith* element and k_i is its weight. All the rules are executed in a maximally parallel way and are described as follows. It is worth pointing out that there are some differences between the rules here and the rules in Theorem 1. The rules considered here are used to stop the execution in some compartments when the current value inside the compartment is bigger than the desired subset sum, thus the execution time and memory required can be saved to a certain degree. R_1 contains $r_{1,i} : A_i \to A_{i+1}, 0 < i < n + 4; r_{1,n+4} : T \to (yes, 0)\{A_{n+4}\}; r_{1,n+5} : A_{n+4} \to (no, 0)\{\overline{T}\}$; R_2 contains $r_{2,i} : [A_i]_2 \to [A_{i+1}, B_i]_2[A_{i+1}]_2\{<V^k\}, 0 < i < n + 1; r_{2,n+i} : v_i \to V\{= B_i\}, 0 < i < n + 1; r_{2,2n+1} : Y \to (T, 1)\{= V^K\}$.

Example Figure 1a shows the procedure for solving the Subset Sum problem for 3 elements with weights 2, 3, 6 and $K = 5$. The compartments on the left side in Steps 2–4 could only execute $r_{2,\ n+1}$ for the guards of the rest rules, which requires the Sum is smaller than or equal to *K*, in compartment C_2.

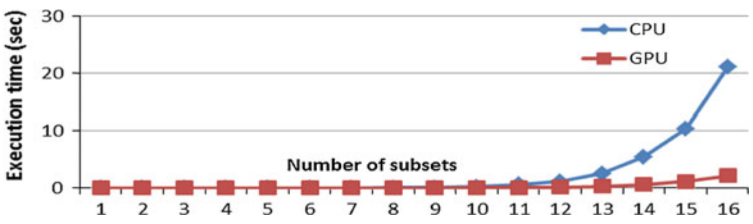

Fig. 2 Comparison of the elapsed time

Figure 1b shows the basic design of the simulation. We define the GPU blocks as the compartments and the GPU threads as units which execute the rules. Each thread and block runs in parallel. All the parameters in each compartment are sent to the GPU global memory. To decrease the time for reading data, we store constants and rules, which are never changed, into the constant memory or texture memory. For the same reason, the compartment parameters are sent to the shared memory in blocks, when the compartment chooses and applies the rules. Because of the restriction of GPU on the number of blocks and threads, the number of the compartments and rules should be less than 65536 and 512, respectively.

This simulation is performed in two stages: selection and execution. The former stage is to search for the rules to be executed in each compartment. The latter stage is used for performing the rules selected at the previous stage. According to the kernel P system definition, the selection stage includes the processes of checking the guard and judging whether one rule is able to be performed. At the execution stage, the compartment can receive new objects, rewriting objects, transition objects. The membrane structure may be changed due to division.

The platform environment is as follows: CPU i5-3240M 2.50 GHz, Memory 8 GB DDR3, NVIDIAGeForce GT650M 1 GB DDR3, Operating system: Windows 7 Professional. The experimental results are shown in Fig. 2 (The number of the use of $r_{2,\ n+i}$ will be very large with the increase of the value of k_i, so the value of each weight, in our simulation, is less than 200).

It can be seen from the results shown in Fig. 2 that as the number of subsets increases, the difference of the elapsed time between GPU and CPU becomes larger, and that the elapsed time on GPU is about 10 times faster than that on CPU.

6 Conclusions

This paper illustrates the modelling capabilities of the new class of kP systems on solving a well-known NP-complete problem, namely the Subset Sum. For this model, a kP system solution is provided and a Promela executable specification has been written and used with the Spin model checker for simulation and property verification. A parallel implementation on GPUs is presented and the results obtained show the efficiency of this implementation wrt CPU one.

Acknowledgments The work of FI, MG, LM and RL was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0688. LVC and MPJ acknowledge the support of the project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain, co-financed 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. GZ acknowledges the support by the National Natural Science Foundation of China (61170016), the Program for New Century Excellent Talents in University (NCET-11-0715) and SWJTU supported project (SWJTU12CX008).

References

1. Păun G (2000) Computing with membranes. J Comput Syst Sci 61:108–143
2. Păun G (2002) Membrane computing: an introduction. Springer, London
3. Păun G, Rozenberg G, Salomaa A (eds) (2010) The Oxford handbook of membrane computing. Oxford University Press, Oxford
4. Ciobanu G, Pérez-Jiménez MJ, Păun G (eds) (2006) Applications of membrane computing., Natural computing seriesSpringer, London
5. Gheorghe M, Ipate F, Dragomir C (2012) A kernel P system. In: Proceedings of 10th brainstorming week on membrane computing. Fénix Editora, Seville, Spain, pp 153–170
6. Gheorghe M, Ipate F, Lefticaru R, Pérez-Jiménez MJ, Ţurcanu A, Valencia Cabrera L, García-Quimondo M, Mierlă L (2012) 3-col problem modelling using simple kernel P systems. Int J Comput Math. doi:10.1080/00207160.2012.743712
7. Ipate F, Dragomir, C, Lefticaru R, Mierlă L, Pérez-Jiménez MJ (2012) Using a kernel P system to solve the 3-col problem. In: Proceedings of the 13th international conference on membrane computing. Computer and Automation Research Institute, Hungarian Academy of Sciences, pp 243–258
8. Ben-Ari M (2008) Principles of the spin model checker. Springer, London
9. Leporati A, Mauri G, Zandron C, Păun G, Pérez-Jiménez MJ (2009) Uniform solutions to SAT and subset Sum by spiking neural P systems. Nat Comput 8:681–702
10. Pérez-Jiménez MJ, Riscos-Núñez A (2005) Solving the subset-Sum problem by P systems with active membranes. New Gener Comput 23(4):339–356
11. Code repository http://www.p-lingua.org/mecosim/doc/case_studies/SubsetSumBIC-TA/
12. Ipate F, Lefticaru R, Pérez-Hurtado I, Pérez-Jiménez MJ, Tudose C (2012) Formal verification of P systems with active membranes through model checking. In: Gheorghe M, Păun G, Rozenberg G, Salomaa A, Verlan S (eds) Proceeding of the international conference on membrane computing 2011. Lecture Notes in Computer Science, vol 7184. Springer pp 215–225
13. Ipate F, Lefticaru R, Tudose C (2011) Formal verification of P systems using Spin. Int J Found Comput Sci 22:133–142