Int. Journ. of Unconventional Computing, Vol. 0, pp. 1–18
Reprints available directly from the publisher
Photocopying permitted by license only

P Systems and X-Machines – A Survey

MARIAN GHEORGHE[1,*], FLORENTIN IPATE[2], SAVAS KONUR[1,*],
MIHAI IONUŢ NICULESCU[2], GEXIANG ZHANG[3]

[1]*School of Computing and Engineering, University of Bradford, UK*
[2]*Faculty of Mathematics and Computer Science, University of Bucharest, Romania*
email: florentin.ipate@unibuc.ro,ionutmihainiculescu@gmail.com
[3]*School of Automation, Chengdu University of Information Technology*
People's *PeopleâĂŹs Republic of China*
email: zhgxdylan@126.com

Received September 20, 2025. Accepted October 20, 2025.

between
The paper is a survey of the main interactions of P systems and X-machines. Firstly, hybrid models resulting from these interactions are analysed, showing their computational capabilities. Secondly, testing methods relying on various X-machine based approaches for applications using different classes of P systems are described and illustrated with some examples. Finally, multi-agent system tools using a combination of features from these two formalisms are presented together with some problems modelled and simulated with such tools.

Keywords: Membrane computing, P systems, X-Machines, modelling, verification, testing, multi-agent systems, tools

1 INTRODUCTION

Membrane computing is a research field initiated around 25 years ago by Gh. Păun [47, 48]. This is a computing paradigm that is inspired by the structure and functions of the living cells, becoming a component of the broader research field of *unconventional computing* or *natural computing*. Its key models are called *membrane systems* or *P systems*. The basic membrane system models consist of a number of regions connected in a certain way and containing multisets of elements and transformation rules acting

* Contact author email: m.gheorghe@bradford.ac.uk, s.konur@bradford.ac.uk

such as,

upon them. These are inspired by molecular biology entities – compartments, bio-chemical compounds and simple molecules interacting in various ways either through complex DNA translation or transcription processes or basic chemical reactions such as oxidation-reduction, hydrolysis, condensation, and neutralization. These are abstracted into mathematical concepts and constructs, a network or hierarchy of regions (or membranes), multisets of objects or strings and transformation and communication rules or more complex cell division or death operations, defining sound and coherent computational models. A key characteristic of these models is the high parallelism and distributivity of the interactions of objects, both at the system level and in each region.

A presentation of the most important membrane computing models is available from a recent survey paper [51]. The relationships between different variants of P systems and other computational models, such as Petri nets, process algebras, mobile ambients and X-machines have been reported in [46].

However, these surveys do not address the relationships between classes of P systems and X-machines. The first paper exploring this topic investigates how P systems with replicated rewriting can be modelled by X-machines (Eilenberg machines) [1].

In this paper, we present hybrid models based on P systems and X-machines; translating P systems into X-machine models; using P systems and X-machines in testing software systems based on P system models; tools based on the interaction of P systems and X-machines.

1.1 Chronological Presentation of Membrane Computing Evolution

Membrane computing started initially as a *theoretical computer science* research, aiming to investigate the *computational power* and *complexity of these models*. Various classes of membrane systems (P systems) have been investigated. These may be classified as *cell-like*, *tissue-like* and *neural-like* P systems. The first research monograph [45] presents some of these basic types of membrane systems investigating the computational power of different variants, each of them defined by a specific system structure and a set of interaction rules. These models have been conceived in such a way that most of them have a maximal computational power, being Turing complete, and able, in some circumstances, to solve hard problems through the generation of exponential space in polynomial (very often, linear) time.

It became obvious that the membrane computing paradigm with its many variants, can solve problems from various areas [10]: *molecular biology* – mechanosensitive communication channels, respiration in bacteria and respiration/photosynthesis interaction in cyanobacteria, cell-mediated immunity, p53 signaling pathways and photosynthesis; *computer science* – analysis of

a public key protocol, approximate algorithms for NP-complete optimization problems, computer graphics and parallel sorting algorithms; *linguistics* – conversational models and formal semantics, and parsing natural languages; membrane computing supporting tools.

At the end of *the first decade* of research in membrane computing a comprehensive *handbook* was published [46] reporting the key achievements grouped up into different categories, covering the most important theoretical results – catalytic P systems, communication P systems, P automata, P systems using strings, splicing/tissue/population P systems, conformon P systems, active membranes, membrane division and creation, P systems with objects on membranes, spiking neural P systems, metabolic P systems, probabilistic and quantum P systems, P colonies, time in membrane systems, semantics of P systems; and some of the most significant applications: metabolic dynamics, economy process modelling, evolutionary membrane algorithms, self-assembly, computer science problems. Finally, connections with other computational models, such as Petri nets, mobile ambients and X-machines have been presented.

During *the second decade*, the research focussing on theoretical aspects in membrane computing community kept a pace comparable to that of the previous period, but it is clear that it shifted consistently towards more diverse applications and also deepened the investigations started earlier. Some of the key *application areas* where membrane computing modelling has made an impact are now briefly presented. There have been initiated investigations in *systems and synthetic biology* [18] on the role of estrogen in cellular mitosis and DNA damage, molecular diffusion and compartmentalization in signal transduction pathways in the study of bacterial chemotaxis, analysis of dynamical population systems, dynamical structures with reaction kinetics in chronobiology, biochemical networks, analysis of *E.coli* respiratory chain. In these studies have been used membrane computing models, specific software simulation tools or simulation algorithms such as *Infobiotics Workbench*, τ*-DPP*, *P-Lingua* and *MeCoSim* supporting population dynamics P systems, *nondeterministic waiting time algorithm*, *MetaPLab*, *FLAME* or model checkers such as *ProB* in Rodin and *PRISM*. More *complex and real-life applications* have been investigated with a broad spectrum of membrane computing products [58]. First, several key research topics that have expanded during this period are presented and analysed – evolutionary membrane algorithms, numerical P systems and different types of fuzzy reasoning methods applied to them. These are then used to model radar emitter signal analysis, digital image processing, controller design, mobile robot path planning, constrained manufacturing parameter optimization problems and distribution network reconfiguration (with evolutionary membrane algorithms); the diagnosis of electric power systems faults (with numerical P

These studies have

systems enhanced with fuzzy reasoning capabilities); the design of mobile robot controllers (using enzymatic numerical P systems); the study of (real-life) ecosystems – the scavenger birds, zebra mussel, Pyrenean chamois and giant panda (with probabilistic P systems). All these investigations requested not only computational models, but also efficient and robust simulation and analysis tools. The most significant *software implementations running on various hardware platforms* have been presented in [57]. The most significant software tools presented are *P-Lingua* and its graphical interface and running environment *MeCoSim, Cyto-Sim, MetaPlab, BioSimWare, Infobiotics Workbench*, and *The Java Environment for Nature-inspired Approaches* (JENA). Implementations on parallel devices, such as *GPU, CUDA* and *FPGA* are also discussed.

In the last years, increasingly, the research interest of the membrane computing community focusses on the theory and applications of a class of membrane computing models derived from spiking neural networks, called *spiking neural P systems* (shortly, SN P systems). They have become very popular and widely used in artificial intelligence based problems. Both theoretical results and AI-based applications are discussed in [59]. The theoretical aspects of SN P systems include the study of their *computational power and complexity*. Some of the key AI-based *real-life applications of SN P systems*, such as complex optimization, classification, fault diagnosis, medical image processing, information fusion, cryptography, and robot control topics, are presented in the above mentioned work.

The area of membrane computing is now, after 25 years, a mature research field, with its own theory and a rich portfolio of application domains. The field continues to grow and develop steadily, being present within the landscape of unconventional research field [24].

computing research

1.2 X-Machines

In mid 1970s Samuel Eilenberg has presented a general approach on unifying various classes of automata into one single model, called *X-machine* (or *Eilenberg machine*) [16]. The model has been then used as a formal specification mechanism in software engineering by Mike Holcombe [25], then considered for program testing purposes. Holcombe and Ipate have then developed a coherent state-based testing method using a special class of X-machines, called *stream X-machines* [26,32]. A stream X-machine resembles a finite state machine (FSM), but with two significant differences: (a) there is a memory component attached to the machine, and (b) the transitions are not labeled with simple inputs/outputs, but with functions. These functions process input values and current memory values, producing output values and updating the memory component. The X-machine model is general enough to simulate other computational models such as FSMs, pushdown automata

and Turing machines. The stream X-machine model has been used especially for specifying dynamic systems and for testing software systems [26]. It is the backbone of the agent-based framework FLAME [11] supporting various simulations, especially in biology [49].

2 COMBINING P SYSTEM AND X-MACHINE MODELS

In this section we present a class of hybrid models obtained by combining two different classes of membrane systems with X-machines (Eilenberg machines).

We start with a basic class of P systems by, first, introducing some informal definitions of the models used in the current and next sections of the paper.

The *P system* Π of degree $m \geq 1$ [45] consists of a finite set V, called *alphabet*; an *output alphabet* $T \subseteq V$; the *membrane structure* μ given as a tree with m nodes, where each node denotes a membrane; *initial strings* w_i, $1 \leq i \leq m$, where w_i belongs to membrane i; R_i, $1 \leq i \leq m$, are finite *sets of evolution rules* of the form $A \rightarrow (u, tar)$, $A \in V$, u is a string and $tar \in \{here, in, out\}$ – when the rule is applied to a string, it leads to a new string by replacing A with u and the new string is kept in the current membrane ($tar = here$) or sent to one of the direct descendants of the current membrane ($tar = in$), arbitrarily chosen, or sent to the parent membrane ($tar = out$).

A *stream Eilenberg (X-)machine* [26] is a system consisting of two finite sets Σ and Γ, called *input* and *output alphabets*, respectively; a (possibly infinite) set of *memory symbols* M; a set of *basic partial relations* Φ on $\Sigma \times M \times M \times \Gamma$; the *next state function* $F : Q \times \Phi \longrightarrow 2^Q$; the sets of *initial states* I and *final states* T; the *initial memory* value is m_0.

An *Eilenberg P system* (*EP system*, for short) $E\Pi$ [5] is a pair consisting of a *membrane structure* with m *membranes*, where the membranes are labeled in a one to one manner with the elements $1, \ldots, m$, and an Eilenberg machine X whose memory is defined by the membranes of μ. X consists of an *alphabet* V; Γ, Q, F are as in any Eilenberg machine (Γ is called the *terminal* alphabet); the *initial* values occurring in membranes $1, \ldots, m$ are finite languages over V, denoted $M_1, \ldots, M_m$; the set of *basic partial relations* Φ is given by $\{\Phi_1, \ldots, \Phi_p\}$, where $\Phi_i = (R_{i,1}, \ldots, R_{i,m})$, $1 \leq i \leq p$, (a *tuple of sets of rules*) where $R_{i,j} \subseteq R_j$ and R_j, $1 \leq j \leq m$, is the set of evolution rules from membrane j; the set of initial states is $I = \{q_0\}$, $q_0 \in Q$; and all the states are final, i.e., $F = Q$.

The EP systems have some similarities with the grammar systems controlled by graphs [12], replacing a one-level structure, which is the current sentential form, with a hierarchical structure defined by means of the

membranes. They are also similar to the X-machines based on distributed grammar systems [19].

One can note that Σ and m_0 from an Eilenberg machine are replaced by V and $M_1, \ldots, M_m$, respectively, in an EP system.

A P system has m sets of evolution rules, each one associated with a membrane. An EP system has the evolution rules distributed across p tuples of sets of rules Φ_i, $1 \leq i \leq p$.

A computation in $E\Pi$ is defined as follows: it starts from the initial state q_0 and an initial configuration of the memory defined by $M_1, \ldots, M_m$ and proceeds iteratively by applying in parallel rules in all membranes, processing in each one all the strings that can be rewritten: in a given state q and for an emerging transition from q labeled Φ_i, $1 \leq i \leq p$, in each membrane j, $1 \leq j \leq m$, each string is processed by a single rule, from those which are applicable, if any, from $R_{i,j}$.

If several rules may be applied to a string, then one rule and one symbol to which it is applied are randomly chosen.

The next state, belonging to $F(q, \Phi_i)$, will be the target state of the selected transition. The result (a set of strings containing only symbols from Γ), called the *language* computed by $E\Pi$, is collected outside of the system, in the environment, at the end of a halting computation.

The family of languages generated by EP systems with at most m membranes, at most s states and using at most p partial relations (tuples of sets of rules) is denoted by $EP_{m,s,p}$. If one of the parameters is not bounded then it is replaced by $*$.

In order to study the computational power of this family matrix grammars (with appearance checking) in the binary normal form [12, 14] are used. The family of languages generated by matrix grammars (with appearance checking) is denoted by MAT (MAT_{ac}). It is known that $MAT \subset MAT_{ac} = RE$ [12, 14], where RE denotes the family of recursively enumerable languages.

The following two results are proved in [8].

Theorem 1. *(i)* $EP_{4,1,1} = MAT$*; (ii)* $EP_{1,1,*} = RE$.

This theorem shows that EP systems with only one membrane and one state, but with an unbounded number of tuples of sets of rules compute all RE languages. What happens when all three parameters are bounded? The next result shows that by increasing either the number of membranes or the number of states, EP systems with a bounded number of tuples of sets of rules can compute all RE languages.

Theorem 2. *(i)* $EP_{1,3,8} = RE$*; (ii)* $EP_{2,1,7} = RE$.

Now, we consider a second class of P systems, called *alphabetic flat splicing systems* [9], where the rules are different from those previously presented. Also, a new type of membrane structure is introduced.

A *flat splicing rule* r has the form $(\alpha|\gamma|\beta)$, where α, β and γ are strings over an alphabet V. For a string $x = u\alpha\beta v$, the rule r, applied to it, yields the string $z = u\alpha\gamma\beta v$ – this is written as $x \vdash_r z$. When α, β are symbols from V or the empty string λ then the rule is called *alphabetic flat splicing rule*. A P system using such rules is called *alphabetic flat splicing P system*. The rules have precise target indicators, i.e., each of them has the form $r : (i, (\alpha|\gamma|\beta), t)$, where i and t are the labels of the host membrane and target membrane, respectively. For such systems the result is obtained in a designated membrane i_0 rather than in the environment. The languages computed (generated) by alphabetic flat splicing P systems are considered, being defined similar to the case of P systems presented above.

Three types of alphabetic flat splicing P systems are defined and investigated in [9]: *Modified Alphabetic Flat Splicing Tissue P Systems* (*MAFSTs*, for short), *Modified Alphabetic Flat Splicing P Systems* (*MAFSPs*, for short) and *Modified Alphabetic Flat Splicing Eilenberg P Systems* (*MAFSEs*, for short). MAFSTs use a graph as a membrane structure (hence, the presence of "tissue" in their name), whereas MAFSPs use a tree, as in the case of previously presented P systems. MAFSEs represent a hybrid model where an MAFSP and an X-machine are merged as in the case of EP systems. The family of languages generated by MAFSXs with at most $m \geq 1$ membranes is denoted by $MAFSX(m)$, $X \in \{T, P\}$. The family of languages generated by MAFSEs with at most m membranes, at most s states and at most p tuples of sets of rules are denoted by $MAFSE(m, s, p)$. As usual, when one of the parameters is unbounded, it is denoted by $*$.

Some relationships between these classes of membrane systems using alphabetic flat splicing rules have been established [9]:

Theorem 3. *(i)* $MAFST(m) \subset MAFST(m+1)$, $m \geq 1$;
(ii) $MAFSP(m) \subset MAFST(m)$, $m \geq 3$;
(iii) $MAFST(m) \subset MAFSE(m, 2, m+2)$, $m \geq 1$.

For a subclass of MAFSEs, called *one flow*, the following result has been proved in [9]:

Theorem 4. *For any MAFSX, X*$\in$ *{T,P}, with at most* $m \geq 1$ *membranes,* Π, *there is an MAFSE with at most 2 membranes, at most m states and at most* m^2 *tuples of sets of rules ,* Π^e, *such that* $L(\Pi) = L(\Pi^e)$.

These models are also used to generate the *double stairs* and *diamond chain code picture languages* [9]. Pure Eilenberg P systems have been used to generate 2D languages [3].

3 TRANSFORMING P SYSTEMS INTO STREAM X-MACHINES

In this section we present three cases of transforming various P systems into stream X-machines or different types of *Communicating stream X-Machines* (*CsXMs*, for short). The aim of the mapping operations is to provide to these classes of P systems a set of tools already developed for the X-machine models – efficient simulators, such as FLAME [11], testing methods [36] and a broad palette of software engineering approaches [25, 26].

In [1] the process of transforming *P systems with replicated rewriting* into *stream X-machines* and a class of *CsXMs using communication matrices* [7] is presented. A P system with replicated rewriting uses strings and the rules have the format $X \rightarrow (v_1, tar_1)|| \ldots ||(v_n, tar_n)$, $n \geq 1$. When such a rule is applied to a string $x_1 X x_2$ one gets n strings $x_1 v_i x_2$, $1 \leq i \leq n$, which are sent to the membranes indicated by the target indicators tar_i. These rules introduce additional parallelism to the existing parallel behaviour of these models and finding a suitable X-machine model represents a challenge. In this respect, in addition to mapping these P systems into standard stream X-machines, a specific class of CsXMs is selected. This model of CsXM allows each of the X-machine components to communicate with any of the others by using the corresponding matrix cells. The complexity aspects regarding the cost of the parallel computation and communication have been established, together with estimates for implementing standard operations in distributed environments, like routing, broadcast, and convergecast.

The *P systems with symbol objects* using *evolution and communicating rules* and *dissolution rules*, with a *partial order* amongst them have been considered to be mapped into a class of *CsXMs with ports and channels* [37], where the communication between X-machine components is done through a channel, which is an unbounded buffer instead of a single cell of the communication matrix as in [7]. The complexity of the translation process has been assessed and compared with [1]. It has been shown that this is better than the complexity of the translation into a standard stream X-machine, but comparable with that of mapping into a CsXM with communication matrices. These CsXMs with ports and channels are more flexible when operations such as dissolution and division are considered, and its instances can be formally verified by a model checker with a set of specific temporal logic operators referring to the X-machine model [17].

A more complex type of membrane computing models, called *kernel P systems* have been mapped into a class of CsXMs where the communication amongst the stream X-machine components is a combination of the previously presented models and each component is restricted to a stream X-machine where the state diagram has no loops [44]. This CsXM model is the backbone of FLAME framework, an agent-based simulation tool running on

high-performance platforms [11]. This mapping allows to efficiently simulate large scale kernel P system models [40].

The opposite problem of *transforming a stream X-machine or an CsXM into a P system* has been also investigated. A restricted class of CsXMs has been considered: the memory set associated to every component is finite and every function processes a symbol from a multiset at any given moment in time, rather than an input from a stream. A *tissue P system* [45] has been the obvious choice for this investigation. The essence of this transformation consists in mapping every stream X machine component of the CsXM into a tissue P system membrane. The objects of the tissue P system are obtained from the states, memory values, input and output symbols and each computation of a function that links a state, a memory value, and an input symbol with another state, a new memory value and an output symbol (the number of such computations is finite) is associated with an evolution or communication rule. In [39], the formal mapping is provided and a case study is presented using specific simulation and visualisation tools.

4 P SYSTEMS AND EILENBERG MACHINES WORKING TOGETHER AS DISTINCT MODELS: THE CASE OF TESTING

All software applications, as any engineering products, irrespective of their nature and purpose, are thoroughly tested before being released, installed and used. Testing appearing everywhere, is part of any technology, and does not have a substitute. In many hardware or software systems testing is conducted together with formal verification, especially when a certain formal model is utilised. In software industry testing is a necessary mechanism to increase the confidence in the product correctness and to make sure that it works properly.

Testing represents a significant line of research investigated in connection with P systems in the context of a broad spectrum of applications based on these models. As already mentioned, there is a substantial work reported on various topics related to: modelling and analysis of biological systems [10, 18], real-life applications [58], spiking neural P system problems [59] and many tools running on different hardware platforms [57].

The testing approaches investigated with respect to P system based applications are *black box testing* methods which require that for a given specification defined as a P system, an implementation of it exists and this will be tested utilising a test set derived from its specification.

The first testing approach on implementations based on membrane systems have been developed for the class of cell-like P systems. In this case, a *grammar-like testing* method, including *rule coverage* and *context-dependent*

rule coverage criteria, has been introduced. Intuitively, the first coverage principle means that we have to identify for every rule of the P system model a computation including the rule. In the second case, the rules are defined in certain contexts; a special situation is when the context is given by the right hand side of another rule, which means that the computation should reveal sequences of rules occurring in successive computation steps.

Formally, the rule coverage is defined as follows: for a rule $r : a \to v$, a multiset u_r covers r if there is a computation $xay \Longrightarrow xvy \Longrightarrow^* u_r$, where x, y, u_r are multisets over the alphabet of the P system. A test set for an application based on a P system model satisfying the rule coverage principle is a set of multisets u_r covering r for any rule r of the P system.

We illustrate the rule coverage approach with a simple example.

Let Π_1 be a P system with one compartment, with the alphabet V_1 and the set of rules $R_1 = \{r_1 : s \to ab, r_2 : a \to c, r_3 : b \to bc, r_4 : b \to c\}$. Each multiset w, is denoted by a vector of non-negative integer numbers $(|w|_s, |w|_a, |w|_b, |w|_c)$, where $|w|_x$, $x \in V_1$, denotes the number of symbol objects x in w. One can easily observe that $T_1 = \{(0, 1, 1, 0), (0, 0, 1, 2), (0, 0, 0, 2)\}$ is a test set for Π_1 satisfying the rule coverage criterion.

State based approaches represent powerful techniques for testing using finite state machine specifications. Given a finite state machine specification and a “black boxâĂŹâĂŹ implementation for which we can only observe its behaviour, we want to test whether the implementation under test conforms to the specification; this is called *conformance testing* or *fault detection* and a finite set of sequences that solves this problem is called a *test suite*. For an implementation derived from a cell-like P system model with non-cooperative evolution and communication rules a Finite State Machine (FSM) based testing approach has been introduced in [29], where a finite set of test sequences satisfying the conformance testing principle is obtained. Of course, coverage criteria for the test sets can be also considered.

``black box testing''

The testing approach provided in [29] uses a set of traces of execution of the P system model, up to a certain limit. These are used in order to obtain a Deterministic Finite Cover Automaton. A DFCA of a finite language U is a deterministic finite automaton that accepts all sequences in U and possibly other sequences longer than any sequence in U and containing a sequence from U as a prefix. This DFCA is an approximation of the given P system.

For the previously introduced P system Π_1 execution traces up to length 4 are considered in [29]. In each step these labels defining a multiset m is denoted by the symbol (m) belonging to the alphabet of the DFCA. The sequences of symbols associated to the multisets of labels of rules used in each step define the finite language U which is used to obtain the DFCA. The transition function of the DFCA obtained from U is $\delta_1(q_0, (r_1)) = q_1$,

$\delta_1(q_1,(r_2r_3)) = q_2,\ \delta_1(q_1,(r_2r_4)) = q_3,\ \delta_1(q_2,(r_3)) = q_2,\ \delta_1(q_2,(r_4)) = q_3$; the alphabet, given by the symbols associated to the multisets of rule labels, is $\{(r_1),(r_2r_3),(r_2r_4),(r_3),(r_4)\}$. The test set is built based on [27].

More complex types of P systems and with many applications have been considered for building model based testing. Specific testing methods for *Spiking Neural P systems* (*SN P systems*, for short) [59] and *kernel P systems* (*kP systems*, for short) [21] have been developed based on learning some classes of stream X-machines.

One way of constructing the required DFCA approximation of a given system based on observations of its behavior is through learning from queries [28]. This approach has been extended to stream X-machines and used to generate test sets to implementations based on SN P systems [30] and then to those using kP system models [34].

An example from [34] illustrates the process described above. Let a kP system $k\Pi_{add}$ with four compartments and the following sets of rules
$R_i = \{r_{i,1} : a \to (a,t_3)\ \{=a\}; r_{i,2} : aa \to (a,t_3)(a,t_3)\ \{\geq a^2\}\},\ i = 1,2;$
$R_3 = \{r_{3,1} : a \to (t,t_4)\ \ \{=a^{n+m+2}\}\},\ \ R_4 = \emptyset$. The execution strategy is choice, i.e., at most one rule per compartment is executed in each step. The initial multisets in the four compartments are $w_{1,0} = a^{m+1}$, $w_{2,0} = a^{n+1}$, $w_{3,0} = w_{4,0} = \lambda$.

The kP system $k\Pi_{add}$ adds the non-negative values $m, n \geq 0$, represented as a^{m+1}, a^{n+1}, respectively, where a^1 is a representation for 0, and produces the sum of them a^{m+n+2} in compartment C_3, which then sends it to C_4.

Consider the upper limit used in the construction of the DFCA to be 3. The labels denoting the rules that appear in each compartment in the finite computation have the form $\phi = (\psi_1,\psi_2,\psi_3)$, where each ψ_i, $1 \leq i \leq 3$, is the label of a rule selected from compartment C_i or no rule, denoted e_i. The number of such labels is at most $3 \times 3 \times 2$, but in fact only half of them appear in these computations. The learning algorithm produces a stream X machine approximating the kP systems $k\Pi_{add}$ such that they coincide on sequences of length at most 3. The symbols that appear on the transitions of this model are exactly the labels extracted from the traces of execution of $k\Pi_{add}$. Based on this stream X-machine model, a test set is generated, which, in addition to that produced in [29] containing sequences of partial relations, includes the configurations defined by the multisets to which the relations are applied and those computed by them.

The following test sequence corresponds to the computation starting with the initial multisets $w_{1,0} = a^2$, $w_{2,0} = a^3$, $w_{3,0} = w_{4,0} = \lambda$:
$[a^2,a^3,\lambda](r_{1,2},r_{2,2},e_3)[\lambda,a,a^4](e_1,r_{2,1},e_3)[\lambda,\lambda,a^5](e_1,e_3,r_{3,1})[\lambda,\lambda,a^4]$.

Such a test sequence will reveal errors that appear when any of the rules involved has an error. For example, if $r_{2,1}$ is wrongly implemented as

$r'_{2,1} : a \to (a^2, t_3)\ \{= a\}$ (a typical instance of mutation testing [42]) then the test result after the second step will be the configuration $[\lambda, \lambda, a^6]$ which is different from the value from the test sequence.

Other testing approaches based on P system models are those for identifiable P systems [23], using model checkers [31] and search-based testing [13] – the latest testing approach is not of type "model based". The class of kP system models benefited, through kPWORKBENCH [41], from complementary model checking and testing validation procedures [20, 33].

5 TOOLS BASED ON THE COMBINED USE OF P SYSTEMS AND X-MACHINES

A direct consequence of combining in different ways P systems and X-machines, as discussed in Sections 2 and 4, is (i) the construction of a series of hybrid models, domain specific languages and tools for multi-agent systems and (ii) the development of simulation, verification and testing tools on different platforms.

The research mentioned in (i) is based on hybrid models. Their key elements, introduced in [38], are the memory and the control structure of the X-machine model [26] and the dynamic structure connecting the compartments of the population P systems, where links between compartments are created or destroyed during the evolution of the model [4]. The rigorous model has been defined in [53] and based on preliminary domain specific languages for stream X-machines, namely XMDL [35], and population P systems, PPSDL [52], several formal notations and multi-agent simulation systems, generically called $OPERAS$, have been created [38, 53, 54].

The basic principle behind $OPERAS$ framework is that each agent can be defined in terms of two separate characteristics, one related to its behaviour (modelling its knowledge, actions, and control over its internal states) and the other one responsible for the reconfiguration of the system structure (adding and removing agents and links between them). Two preliminary versions have been created, $OPERAS_{XX}$, using both characteristics of the agents from communicating X-machines, and $OPERAS_{CC}$, entirely based on population P systems. A further version, $OPERAS_{XC}$, and a slight variation of it, $OPERAS_{XP}$, use a combination of population P systems and X-machines [54]. A number of multi-agent systems have been modelled and analysed using these frameworks.

A challenging modelling and simulation project is the autonomous intelligent swarms of satellites proposed for NASA missions that have complex behaviours and interactions. The emergent properties of swarms are very

complex and powerful, but equally difficult to design and assure that proper behaviours will emerge. A thorough investigation of formal method techniques for verification and validation of NASA swarm-based missions has been provided [50]. Among the formal methods considered, X-machines have showed good properties, but also a drawback referring to the predictive qualities for emergent behaviour of multiple agents. In order to overcome this issue, a multi-agent formal model based on $OPERAS_{XC}$ has been considered in [55]. Individual agents, with their behaviour, have been modelled as communicating X-machines, whereas the relationships among them have been formalised with primitives inspired by the way the compartments of population P system models create and destroy links. This $OPERAS_{XC}$ approach provides a formal framework for the design and simulation of multi-agent systems. The formal verification of the hybrid model is achieved by using an extension of the CTL language [17].

$OPERAS_{XP}$ has been used to model, as a multi-agent system, the behaviour of a colony of Pharaoh ants searching for food when the foraging process is based on the pheromone trail produced by forager ants [56]. This framework includes a NetLogo component allowing to visualize the behaviour of the colony and run different scenarios where the number of ants, the amount of food and the format of the nest can be varied with respect to various requirements and constraints.

kPWORKBENCH, a tool allowing the simulation, verification and testing of kP systems, includes a translator of kP system models into a specific type of X-machines compatible with FLAME [44], as mentioned in Section 2. This illustrates the use of different simulation platforms, as mentioned at (ii) above, and provides ways to select the most appropriate simulation environment for a given problem. In [2], a pulse generator synthetic biology case study is used to assess the performance of the native simulator for kP systems, provided by kPWORKBENCH, and FLAME sequential simulator, where the X-machine model is obtained in accordance with the translation method described in [44]. The experiments show that kPWORKBENCH simulation tool runs faster than FLAME, but the former saves every step all the generated data on an external memory. Another experiment assessing the performance of kPWORKBENCH native simulator against FLAME running in sequential and parallel mode (with 2,3 and 4 processors), by using the subset sum problem as a benchmark, has been reported in [43]. In this case FLAME simulator, which now is no longer saving the data after every computation step, shows a much better performance compared to the kPWORKBENCH simulator. Also, the experiments show that sequential FLAME runs faster than parallel FLAME - this is a consequence of the high level of communication among compartments, as imposed by the solution provided.

2, 3

6 FUTURE RESEARCH DEVELOPMENTS

After almost 25 years of research on the interactions between membrane systems and Eilenberg (X-)machines (the first paper on this subject, [1], has been published in 2002, but presented at the Workshop on Membrane Computing in 2001), one can conclude that both the hybridisation process of creating new models by mixing features of the two initial models, as well as the combined use of them, are very effective approaches for defining computational models with interesting characteristics, for providing solutions to various problems and producing tools supporting their analysis.

The above mentioned research directions show potential for further developments. The area of combining Eilenberg machines and P systems into new hybrid models is still active and attractive – [9] and [22] have been recently accepted for publication. Splicing P systems and 2D array P systems are likely candidates to produce new hybrid models with Eilenberg machines, as illustrated by recent developments [3] and [9], respectively. Existing model-checking techniques may be extended to hybrid models enabling their systematic formal verification. Automata-learning and reinforcement-learning methods can be employed to automatically infer and refine hybrid models based on observed behaviour. The inferred models will then serve as a foundation for systematic testing and verification. Implementation of parallel algorithms running on high-performance computers or FPGA/GPU platforms for hybrid models can be developed to enhance scalability and computational efficiency. The second line of research has very promising perspectives for further developments through important applications, such as those based on spiking neural P systems [59], numerical P systems [15, 60] and other P system models used in real-life applications [58], where new testing and verification methods, as illustrated for some basic models in [6], have to be defined. Generative AI and AI agents for automatic model translation can be considered for new developments related to these topics.

published

ACKNOWLEDGMENTS

The paper is dedicated to Academician Gheorghe Păun on the occasion of his 75^{th} birthday, as a token of appreciation for his landmarking contributions to several branches of theoretical computer science and for introducing the seminal concept of Membrane Computing that has generated a very fertile research area.

Florentin Ipate and Marian Gheorghe would like to also dedicate this paper to Professor Mike Holcombe, who revealed to them the X-machine model and suggested the research potential of combining this topic with Membrane Computing.

The authors thank the anonymous reviewers for their comments that allowed to improve the final version of the paper. Marian Gheorghe's and Savas Konur's research has been supported by the Royal Society grant IESR3213176, 2022-2025. Gexiang ZhangâĂŹs research has been supported by the Sichuan Science and Technology Program (2025YFHZ0103, 2025HJRC0022).

Zhang's

REFERENCES

[1] Joaquín Aguado, Tudor Bălănescu, Marian Gheorghe, Mike Holcombe, and Florentin Ipate. (2002). P systems with replicated rewriting and stream X-machines. *Fundamenta Informaticae*, 49(1):17–33.

[2] Mehmet E. Bakir, Savas Konur, Marian Gheorghe, Ionuţ Niculescu, and Florentin Ipate. (2014). High performance simulations of kernel P systems. In *2014 IEEE International Conference on High Performance Computing and Communications, August 2014, Paris, France*, pages 409–412. IEEE.

[3] Somnath Bera, Atulya K. Nagar, Kumbakonam Govindarajan Subramanian, and Gexiang Zhang. (2024). Pure 2D Eilenberg P systems. *Journal of Membrane Computing*, 6(4):258–265.

[4] Francesco Bernardini and Marian Gheorghe. (2004). Population P systems. *Journal of Universal Computer Science*, 10(5):509–539.

[5] Francesco Bernardini, Marian Gheorghe, and Mike Holcombe. (2003). PX systems = P systems + X-machines. *Natural Computing*, 2(3):201–213.

[6] Radu Traian Bode, Florentin Ipate, and Mihai Ionuţ Niculescu. (2025). A model learning based testing strategy for numerical P systems. In *International Conference on Membrane Computing, Chengdu, China, September 19–21, 2025*. Chengdu University of Information Technology.

[7] Tudor Bălănescu, Tony Cowling, Horia Georgescu, Marian Gheorghe, Mike Holcombe, and Cristina Vertan. (1999). Communicating stream X-machine systems are no more than X-machines. *Journal of Universal Computer Science*, 5(9):494–507.

[8] Tudor Bălănescu, Marian Gheorghe, Mike Holcombe, and Florentin Ipate. (2003). Eilenberg P systems. In Gheorghe Păun, Grezorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, 3^{rd} *International Workshop on Membrane Computing, WMC-CdeA 2002, Curtea de Argeş*, volume 2597 of *Lecture Notes in Computer Science*, pages 43–57. Springer Berlin, Heidelberg.

[9] Rodica Ceterchi, Marian Gheorghe, Lakshmanan Kuppusamy, and Kumbakonam Govindarajan Subramanian. (2025). Three classes of modified alphabetic flat splicing P systems. *Journal of Membrane Computing*, 7(3):296–311.

[10] Gabriel Ciobanu, Gheorghe Păun, and Mario J. Pérez-Jiménez, editors. (2006). *Applications of Membrane Computing*. Springer Berlin, Heidelberg.

[11] Simon Coakley, Marian Gheorghe, Mike Holcombe, Swan Chin, David Worth, and Chris Greenough. (2012). Exploitation of high performance computing in the FLAME agent-based simulation framework. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, HPCC-ICESS 2012*, pages 538–545.

[12] Erzsébet Csuhaj-Varjú, Jürgen Dassow, Jozef Kelemen, and Gheorghe Păun. (1994). *Grammar Systems – A Grammatical Approach to Distribution and Cooperation*. Routledge.

[13] Ana Ţurlea, Marian Gheorghe, Florentin Ipate, and Savas Konur. (2019). Search-based testing in membrane computing. *Journal of Membrane Computing*, 1(4):241–250.

[14] Jürgen Dassow and Gheorghe Păun. (1989). *Regulated Rewriting in Formal Language Theory*. Springer Berlin, Heidelberg.

[15] Jinping Dong, Gexiang Zhang, Yangheng Hu, Yijin Wu, and Haina Rong. (2024). An optimization numerical spiking neural membrane system with adaptive multi-mutation operators for brain tumor segmentation. *International Journal of Neural Systems*, 34(8):2450036.

[16] Samuel Eilenberg. (1974). *Automata, Languages and Machines*. Academic Press.

[17] George Eleftherakis, Petros Kefalas, and Anna Sotiriadou. (2001). Extending temporal logic to facilitate formal verification of X-machine models. *Annals of the University Bucureşti. Mathematics-Informatics*, 50(1):79–95.

[18] Pierluigi Frisco, Mario J. Pérez-Jiménez, and Marian Gheorghe, editors. (2014). *Applications of Membrane Computing in Systems and Synthetic Biology*. Springer Cham.

[19] Marian Gheorghe. (2000). Generalised stream X-machines and cooperating distributed grammar systems. *Formal Aspects of Computing*, 12(6):459–472.

[20] Marian Gheorghe, Rodica Ceterchi, Florentin Ipate, Savas Konur, and Raluca Lefticaru. (2018). Kernel P systems: From modelling to verification and testing. *Theoretical Computer Science*, 724:45–60.

[21] Marian Gheorghe, Florentin Ipate, Ciprian Dragomir, Laurenţiu Mierlă, Luis Valencia-Cabrera, Manuel García-Quismondo, and Mario Pérez-Jiménez. (2013). Kernel P systems – Version I. In *Eleventh Brainstorming Week on Membrane Computing (11BWMC), Seville*, pages 97–124. Fénix Editora.

[22] Marian Gheorghe, Florentin Ipate, Kumar Kannan, Savas Konur, Lakshmanan Kuppusamy, Rodica Lefticaru, Anand Mahendran, and Mihai Ionuţ Niculescu. ~~(to appear).~~ (2025) Spiking neural P systems and kernel P systems. *Journal of Membrane Computing*. 7(4):437—459

[23] Marian Gheorghe, Florentin Ipate, and Savas Konur. (2016). Testing based on identifiable P systems using cover automata and X-machines. *Information Sciences*, 372:565–578.

[24] Marian Gheorghe, Savas Konur, and Florentin Ipate. (2017). Kernel P systems and stochastic P systems for modelling and formal verification of genetic logic gates. In Andrew Adamatzky, editor, *Advances in Unconventional Computing, Volume 1: Theory*, pages 661–675. Springer Cham.

[25] Mike Holcombe. (1988). X-machines as a basis for dynamic system specification. *Software Engineering Journal*, 3(2):69–76.

[26] Mike Holcombe and Florentin Ipate. (1998). *Correct Systems. Building a Business Process Solution*. Springer Berlin, Heidelberg.

[27] Florentin Ipate. (2010). Bounded sequence testing from deterministic finite state machines. *Theoretical Computer Science*, 411(16–18):1770–1784.

[28] Florentin Ipate. (2012). Learning finite cover automata from queries. *Journal of Computer and System Sciences*, 78(1):221–244.

[29] Florentin Ipate and Marian Gheorghe. (2009). Finite state based testing of P systems. *Natural Computing*, 8(4):833–846.

[30] Florentin Ipate and Marian Gheorghe. (2022). A model learning based testing approach for spiking neural P systems. *Theoretical Computer Science*, 924:1–16.

[31] Florentin Ipate, Marian Gheorghe, and Raluca Lefticaru. (2010). Test generation from P systems using model checking. *Journal of Logic and Algebraic Programming*, 79(6):350–362.

[32] Florentin Ipate and Mike Holcombe. (1997). An integration testing method that is proved to find all faults. *International Journal of Computer Mathematics*, 63(3–4):159–178.

[33] Florentin Ipate, Raluca Lefticaru, and Cristina Tudose. (2011). Formal verification of P systems using Spin. *International Journal of Foundations of Computer Science*, 22(1):133–142.

[34] Florentin Ipate, Ionuţ Mihai Niculescu, Raluca Lefticaru, Savas Konur, and Marian Gheorghe. (2023). A model learning based testing approach for kernel P systems. *Theoretical Computer Science*, 966:113975.

[35] Paraskevi Kapeti and Petros Kefalas. (2000). A design language and tool for X-machines specification. In Dimitrios I. Fotiadis and Stavros D Nikolopoulos, editors, *Advances in Informatics*, pages 134–145.

[36] Petros Kefalas, George Eleftherakis, Mike Holcombe, and Marian Gheorghe. (2003). Simulation and verification of P systems through communicating X-machines. *BioSystems*, 70(2):135–148.

[37] Petros Kefalas, George Eleftherakis, and Evangelos Keris. (2003). Communicating X-machines: A practical approach to formal and modular specification of large systems. *Journal of Information and Software Technology*, 45(5):269–280.

[38] Petros Kefalas, Ioanna Stamatopoulou, and Marian Gheorghe. (2005). A formal modelling framework for developing multi-agent systems with dynamic structure and behaviour. In *International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2005*, volume 3690 of *Lecture Notes in Artificial Intelligence*, pages 122–131. Springer Berlin, Heidelberg.

[39] Petros Kefalas, Ioanna Stamatopoulou, Ilias Sakellariou, and George Eleftherakis. (2009). Transforming communicating X-machines into P systems. *Natural Computing*, 8(4):817–832.

[40] Savas Konur, Mariam Kiran, Marian Gheorghe, Mark Burkitt, and Florentin Ipate. (2015). Agent-based high-performance simulation of biological systems on the GPU. In *2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS) and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)*, pages 84–89.

[41] Savas Konur, Laurenţiu Mierlă, Florentin Ipate, and Marian Gheorghe. (2020). kPWorkbench: A software suit for membrane systems. *SoftwareX*, 11:100407.

[42] Raluca Lefticaru, Marian Gheorghe, and Florentin Ipate. (2011). An empirical evaluation of P system testing techniques. *Natural Computing*, 10(1):151–165.

[43] Raluca Lefticaru, Luis F. Macías-Ramos, Ionuţ Mihai Niculescu, and Laurenţiu Mierlă. (2017). Agent-based simulation of kernel P systems with division rules using FLAME. In Alberto Leporati, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *17th International Conference on Membrane Computing, CMC 2016, Milan, Italy, July 25-29, 2016*, volume 10105 of *Lecture Notes in Computer Science*, pages 286–306. Springer Cham.

[44] Ionuţ Mihai Niculescu, Marian Gheorghe, Florentin Ipate, and Alin Ştefănescu. (2014). From kernel P systems to X-machines and FLAME. *Journal of Automata Languages and Combinatorics*, 19(2–4):239–250.

[45] Gheorghe Păun. (2002). *Membrane Computing – An Introduction*. Springer Berlin, Heidelberg.

[46] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. (2010). *The Oxford Handbook of Membrane Computing*. Oxford University Press.

[47] Gheorghe Păun. (1998). Computing with membranes. Technical report, Turku Centre for Computer Science.

[48] Gheorghe Păun. (2000). Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143.

[49] David Rhodes, Mike Holcombe, and Eva Qwarnstrom. (2016). Reducing complexity in an agent based reaction model – Benefits and limitations of simplifications in relation to run time and system level output. *BioSystems*, 147:21–27.

[50] Christopher Rouff, Amy Vanderbilt, Mike Hinchey, Walt Truszkowsk, and James Rash. (2005). Properties of a formal method for prediction of emergent behaviors in swarm-based systems. In Panayiotis Bozanis and Elias N. Houstis, editors, *Second International Conference on Software Engineering and Formal Methods (SEFMâĂŹ04), September 28 - 30, 2004*, pages 24–33. IEEE Computer Society. SEFM'04

[51] Bosheng Song, Kenli Li, David Orellana-Martín, Mario J. Pérez-Jiménez, and Ignacio Pérez Hurtado. (2021). A survey of nature-inspired computing: Membrane computing. *ACM Computing Surveys*, 54(1):629–649.

[52] Ioanna Stamatopoulou, Petros Kefalas, George Eleftherakis, and Marian Gheorghe. (2005). A modelling language and tool for P systems. In Panayiotis Bozanis and Elias N. Houstis, editors, *Panhellenic Conference in Informatics, PCI'05, Volos, Greece, November, 11-13, 2005*, pages 375–383. Springer Berlin, Heidelberg.

[53] Ioanna Stamatopoulou, Petros Kefalas, and Marian Gheorghe. (2007). Modelling the dynamic structure of biological state-based systems. *BioSystems*, 87(2–3):142–149.

[54] Ioanna Stamatopoulou, Petros Kefalas, and Marian Gheorghe. (2007). OPERAS: a formal framework for multi-agent systems and its application to swarm-based systems. In Alexander Artikis, Gregory M.P. O'Hare, Kostas Stathis, and George Vouros, editors, *International Workshop on Engineering Societies in the Agents World, ESAW'07*, volume 4995 of *Lecture Notes in Computer Science*, pages 208–223. Springer Berlin, Heidelberg.

[55] Ioanna Stamatopoulou, Petros Kefalas, and Marian Gheorghe. (2007). OPERAS for space: Formal modelling of autonomous spacecrafts. In *Panhellenic Conference in Informatics, PCI'07, Patras, Greece, May, 18–20, 2007*, pages 69–78.

[56] Ioanna Stamatopoulou, Ilias Sakellariou, Petros Kefalas, and George Eleftherakis. (2008). OPERAS for social insects: Formal modelling and prototype simulation. *Romanian Journal of Information Science and Technology*, 11(3):267–280.

[57] Gexiang Zhang, Mario Pérez-Jiménez, Agustín Riscos-NúÃśez, Sergey Verlan, Savas Konur, Thomas Hinze, and Marian Gheorghe. (2021). *Membrane Computing Models: Implementations*. Springer Cham. N\'{u}\~{n}ez

[58] Gexiang Zhang, Mario J. Pérez-Jiménez, and Marian Gheorghe. (2017). *Real-life Applications with Membrane Computing*. Springer Cham.

[59] Gexiang Zhang, Sergey Verlan, Tinfang Wu, Francis George. C. Cabarle, Jie Xue, David Orellana-Martín, Jianping Dong, Luis Valencia-Cabrera, and Mario J. Pérez-Jiménez. (2024). *Spiking Neural P Systems: Theory, Applications and Implementations*. Springer Cham.

[60] Luping Zhang, Fei Xu, Dongyang Xiao, Jianping Dong, Gexiang Zhang, and Ferrante Neri. (2022). Enzymatic numerical spiking neural membrane systems and their application in designing membrane controllers. *International Journal of Neural Systems*, 32(11):2250055.