# Eilenberg P Systems: a Bio-Computational Model

Marian Gheorghe[1]*, Mike Holcombe[1] and Petros Kefalas[2]

[1] Department of Computer Science
Sheffield University, Sheffield, S14DP, UK
[2] Department of Computer Science
City College, 54624 Thessaloniki, Greece
Email: {M.Gheorghe,M.Holcombe}@dcs.shef.ac.uk kefalas@city.academic.gr

**Abstract.** The paper presents the main features of P systems, X machines and of a new computational device called Eilenberg P system. The sequential and the parallel Eilenberg P systems are presented and results reflecting the computational power of these models and their effectiveness in solving NP-complete problems are briefly mentioned. The behaviour of a bee colony is modelled as a society of communicating agents acting in parallel and synchronizing their behaviour. Two computational models for defining the agents behaviour are introduced and compared and tools developed for these models are briefly illustrated.

## 1 Introduction

Various computational models that are successfully used elsewhere in computer science to model software engineering problems (Petri nets [39], $\pi$ calculus and ambient calculus [27], Statecharts [30], X machines [22]), artificial intelligence paradigms (agents of various types [16]), theoretical approaches (formal language theory [38], process algebra [41]) have been considered to modelling different biological phenomena and aspects of the living organisms. A lot of effort has been spent into pragmatic modelling leading to specific computational approaches facilitating model exchanges between various research groups through different standard software platforms (SBML [25], CellML [23]) [35].

The idea of modelling various biological aspects by means of formal language based methods and of linguistics approaches has been considered since its inception. Historically the decade that unveiled the structure of DNA also witnessed a revolutionary development in linguistics initiated by the work of Noam Chomsky [11]. Recently parallels between biological evolution and the history of language have been revealed [29] and a thorough survey has been dedicated to the use of linguistics methods in the study of DNA as language and of the genome as the book of life [40].

In the last years attempts have been made to devise computational models in the form of generative devices like P systems [36], inspired by bio-chemical mechanisms occurring in living cells, splicing systems [38], expressing transformations defined upon DNA strands (more information on these subjects may be found at [24] and [26], respectively). New computational paradigms of modelling concurrent systems' behaviour

* Research partially supported by EPSRC grant GR/R84221/01

such as Gamma [4] or Cham [9], inspired by chemical reactions developing in parallel, were introduced in the field of concurrent programming. Experiments have been made in order to show how DNA strands may be used as a massive parallel computer to solving well-known hard problems [1]. All these models, paradigms or experiments rely on some bio-chemical facts in approaching new ways of computing either at some abstract level or in a more practical manner. There are, on the other hand, computational models like random grammars, Boolean networks [31], developed to express bio-chemical reactions occurring at the cell level, or X machines, utilized to model metabolic pathways [20], or the behaviour of bee colonies [14]. The important role of generative models has been emphasised in the context of analysis of the emergence of functional adaptive systems [31].

Some variants of X machines may become suitable for molecular computing models due to their completeness property, capability of naturally simulating different computationally complete models - Turing machines [13], two-stack automata [22], P systems with replicated rewriting [2] -, suitability to define dynamic systems reacting to input stimuli, flexibility in expressing hierarchical or distributed systems, and ability in capturing hybrid specifications.

This work will present membrane computing or P systems which is a new emerging component of natural computing paradigm and X machines which is one of the formal methods used largely in software engineering, but with significant biological modelling capabilities. A simple example showing the behaviour of a honey-bee colony will be presented using P system and X machine models. Tools developed for these models demonstrate the practicality of the approach. A new model combining P systems and X machines is also presented.

## 2 P Systems and X Machines

We only briefly summarize some topics currently under development in the rapidly growing area of *membrane computing* (*P systems*) which is part of the natural computing paradigm. Already a monograph has been dedicated to this subject [37] and some fairly recent results have been reported ([10, 24]). Membrane computing has been introduced with the aim of formally defining a computing device which abstracts out from the cell and some of its functionality (initially has been called a *super-cell system* [36]). Some of the main elements of the living cells are the membranes which separate the cell from its environment by means of plasma membrane and compartmentalize the inner part of the cell through internal membranes; each compartment contains its own enzymes and other specialized molecules. The following feature has been identified for the computing device that abstracts out from the living cell structure and its functionality: the *membrane structure,* where *multisets* of chemical or biological elements react according to prescribed developmental *rules.* The cell as a complex living system has several compartments which are separated by membranes of various types. The elements floating in these compartments are either simple chemical ions or more complex DNA molecules all reacting according to some rules in a broadly nondeterministic manner and mostly simultaneously.

The model inheriting most of these characteristics is a hierarchically organized system containing in each region a multiset of objects and/or other regions as well. The whole structure is separated from the environment by a membrane called the *skin membrane.* The evolution rules are applied in a nondeterministic maximally parallel manner: the objects evolve and the rules applied to them are nondeterministically chosen and all the objects that may evolve at a given stage should do so. The rules reflect both object transformation and object transfer from a region to another one; these characteristics might both be present in the same rule or might come separately in distinct rules. There are also rules which act upon the membrane structure as well creating new membranes or dissolving some of the existing ones. In this way a *dynamic membrane structure* supports the transformations occurring within it. The membrane structure with the set of objects and the evolution rules constitutes a P system. The membrane structure and the set the objects occurring inside of it at a given moment identify a *configuration* of a P system. By using the rules in the specified way a *transition* from one configuration to another one takes place. A sequence of transitions defines a *computation* and a successful computation is one which halts. A *halting computation* is defined to be the result produced by the system which is usually read from a specified region or consists of all the objects sent out of the system. The *result* may consist of the set of all *vectors of natural numbers* describing the multiplicities of the objects at the end of a successful computation or the *output* produced starting from an initial configuration.

**Definition 1.** *A P system is a construct [37]*

$$\Pi = (V, T, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n),$$

*where*

- $V$ *is a an alphabet; its elements are called objects;*
- $T \subseteq V$ *is the output alphabet;*
- $\mu$ *is a membrane structure consisting of* $n$ *membranes, with the membranes labeled in a one to one manner with the elements* $1$ *to* $n$;
- $w_i, 1 \leq i \leq n$ *are strings which represent multisets over* $V$ *associated with the regions* $1, \ldots, n$ *delimited by the membranes with the same labels of* $\mu$*;*
- $R_i, 1 \leq i \leq n$ *are finite sets if evolution rules over* $V$ *associated with the regions* $1, \ldots, n$ *of* $\mu$, *of the form* $a \longrightarrow x$ *where* $a$ *is a word over* $V$ *and* $x$ *is a word over* $\{a_{here}, a_{in}, a_{out} \mid a \in V\}$.

The evolution rules are rewriting rules with targets associated to its right hand side objects. When the result of a halting computation is collected in a specified region then this is identified in the above definition. An example with five regions is shown in Fig. 1. The multiset $w_2$ consists of $xuxy$ and one of the evolution rules of $R_2$ is $v \rightarrow u_{here}x_{out}$; when this rule is applied then every $v$ from this region is replaced by an $u$ which will stay in 2 and an $x$ which will be sent outside of this region, i.e. in the region labelled by 1.

Various other ingredients have been also considered in addition to the basic Elements mentioned above. Some of these have been brought from other similar generative devices: an *order relationship* among the rules from the same region, *Lindenmayer parallel rewriting systems* when objects are replaced by strings, *probabilistic systems,* P systems with *graph rules.* However most of the features associated with these systems have a biological motivation: *electrical charges, energy, membrane permeability, catalysts, carriers, promoters* and *inhibitors.* P systems using only *symport/antiport* communication rules have been introduced, as well as various kinds of P automata [37]. Also there are variants of P systems with string objects and appropriate evolution rules. These variants will be used in section 4.

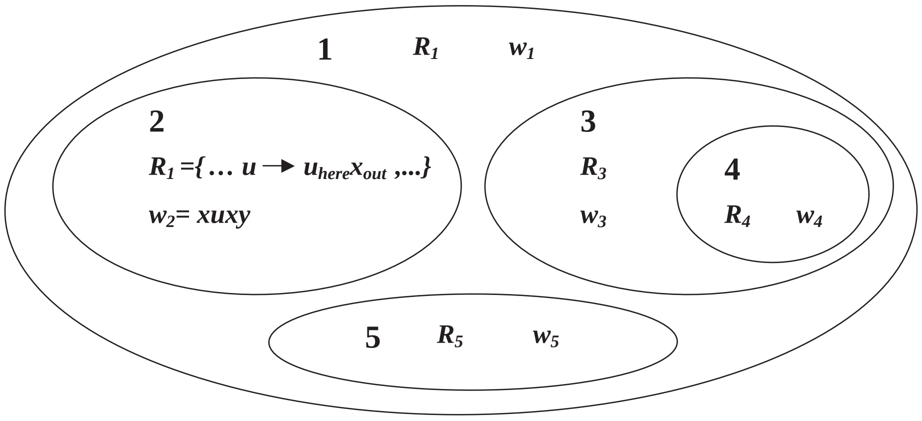

**Fig. 1.** An example of a P System with five regions.

Although the theoretical side has been intensively investigated, some applications and connections with other research areas have been considered also: *distributed algorithms* defined in the context of communicating membrane systems [12], models of *ecological systems* [42], models of *social insects* behaviour [14], *software* simulating different variants of P systems [24]. *New programming language paradigms* relying on various bio-computational models refer to P systems as well [15].

Biological modelling has been approached by some 'classical' methods coming from various well-established research areas: formal languages, computational models, software engineering, artificial intelligence, artificial life. One of these approaches is based on X machines which previously have been investigated in software engineering. Similarly well known models of concurrent and distributed systems with a very successful history in software engineering modelling (Petri nets [28], Statecharts [17]) have been also applied to biological modelling ([39, 30]).

The *X machine* model was introduced by Eilenberg in mid seventies [13] as a general computational machine. These devices are equivalent in computational power to Turing machines. The study of X machines has been abandoned for more than a decade and has been reconsidered in the late eighties in a slightly modified form called *stream X machine* as a specification language for dynamic systems [18]. A stream X machine resembles a finite state machine (FSM) with outputs but with two significant differences: (a) there is a *memory* component attached to the machine, and (b) the transitions are not labelled with simple inputs/outputs but with *functions* that operate on inputs and memory values, update the memory values and yield output symbols. These differences allow the X machines to be more expressive and flexible than the FSMs. Other machine models like pushdown automata or Turing machines are too low level and hence of little use for real system specifications. The machine, depending on the *current state* of control and the *current memory* value, consumes an *input* symbol from the input stream and determines the *next state,* the *next memory* value and an *output* value which is catenated at the right side of the output stream.

**Definition 2.** *A stream X machine is a construct [22]*

$$X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0),$$

*where:*

- $\Sigma$ *is a finite set called the* input *alphabet;*
- $\Gamma$ *is a finite set called the* output*alphabet;*
- $Q$ *is a finite set called the set of* states*;*
- $M$ *is a (possibly infinite) set of* memory symbols*;*
- $\Phi$ *is a set of basic partial functions* $\phi : \Sigma \times M \to M \times \Gamma$, *called the set of* basic processing functions;
- $F$ *is the* next state function $F : Q \times \Phi \to 2^Q$;
- $I$ *and* $T$ *are the sets of* initial *and* final *states;*
- $m_0$ *is the* initial memory *value.*

A configuration $(q, m, \sigma)$ of an X machine is given by a state $q$, a memory value $m$ and an input symbol $\sigma$. A transition takes place between two configurations $(q, m, \sigma)$ and $(q', m', \sigma')$ if there exists a function $\phi$ such that $(\sigma, m) \in dom(\phi)$ (where $dom(\phi)$ denotes $\phi$'s domain), $q' \in F(q, \phi)$ and there exists $\gamma \in \Gamma$ such that $\phi(\sigma, m) = (m', \gamma)$. An initial configuration has the form $(q_0, m_0, \sigma)$, where $q_0 \in I$, $m_0$ is the initial memory value and $\sigma$ is an input symbol. A computation is a sequence of transitions starting from the initial configuration and arriving in a configuration with a state $q \in T$. A computation associates a stream of inputs and produces a stream of outputs. In the initial configuration the input symbol used is the leftmost symbol of the input stream and the output stream is empty. A computation is successful if in its last configuration the input stream is emptied. In Fig. 2 it is shown an X machine with four states and eight functions. If $s_1$ is the current state with the current memory value $m$ and the next input value $\sigma$ then the function $\phi_1$ is triggered and this will update the memory value, $m'$ and will output a value $\gamma$ that is added to the right hand side of the current output stream. The computation will then resume from the state $s_2$ which together with $m'$ and the input value next to $\sigma$ define the next configuration of this machine.

A stream X machine acts as a translator which, during a successful computation, Transforms an input stream into an output one.

The power of these devices has been investigated in the case of simple functions (push and pop) and with the memory organized as a stack [22].

The main strength of the X machine model has been explicitly proven in the area of software testing and verification, a very important component of almost all software engineering processes. A very powerful theory has been developed which describes how to test the behavioural equivalence of two stream X machines with a finite test set providing certain design for test conditions are satisfied ([22, 19]). The issue of what happens when the two machines are refined in a controlled way has also been investigated [22].

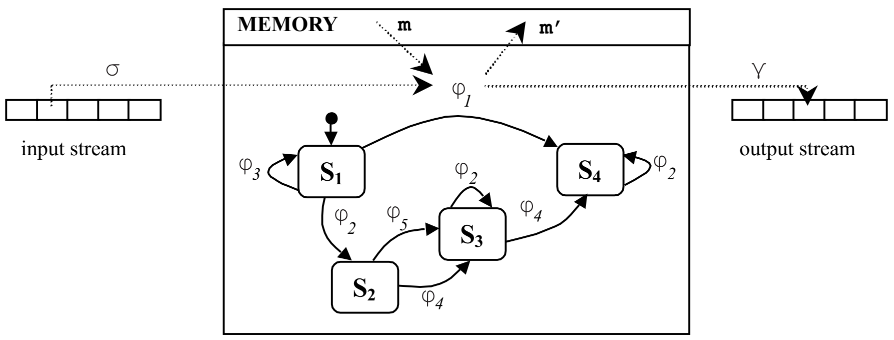

**Fig. 2.** An abstract example of an X-machine; $\phi_i$: functions operating on inputs and memory, $S_i$: states.

A new model of concurrency has been introduced which is based around the asynchronous communication of a collection of X machines mediated through a matrix representing the channel capabilities.

A *communicating X machine* model consists of several X machines, which are able to exchange messages. These are normally viewed as inputs to some functions of an X machine model, which in turn may affect the memory structure. A communicating X machine model can be generally defined as a construct:

$$CX = (X_1, \ldots, X_n, CR),$$

where

- $X_i$ is the $i$-th X machine component that participates in the system, and
- $CR$ is a *communication* relationship between the $n$ X machine components.

There are several approaches in order to formally define a communicating X machine. Some of them deviate from the original definition of the X machine which has the effect of not being able to reuse existing models [6, 5]. Also, in these approaches $CR$ is defined in different ways, with the effect of achieving either synchronous or asynchronous communication.

In the current work, $CR$ is defined as a relationship $CR \subseteq X \times X$ where $X = \{X_i | 1 \leq i \leq n\}$, which determines the communication channels that exist between the

X machines of the system. A tuple $(X_i, X_j) \in CR$ denotes that X machine $X_i$ can write a message through a communication channel to a corresponding input stream of X machine $X_j$. A formal definition of communicating X machine model is provide in [32], but is omitted in this approach as the example presented below will show only a fragment which deals with an X machine component

## 3 An Example: Collective Foraging

In this section both P systems and X machines will be used in order to specify the collective foraging behaviour of a colony of honey-bees. The behaviour is restricted to a number of rules referring to: travelling from the nest to the source, searching for the source, collecting nectar from the source, travelling back to the nest, transmitting the information about the source, the reaction of a bee in the nest to the dancing of a nest mate [43].

The P system model has the following elements organized on three layers:

- the environment ($w_1$) containing the nectar source;
- the nest ($w_2$);
- the bees ($w_3$ to $w_n$).

$w_1$ will contain information about the amount of nectar carried by a bee, its current position, and information about the source. $w_2$ will have for each bee in the hive, the amount of nectar carried, its current position, a memory value identifying the nectar source position as well as an identification of each bee. $w_i$, $3 \leq i \leq n$ will give the position of a bee, the amount of nectar carried by a bee and a position of the source as a memory value; when some nectar will be transferred to another bee, the position and amount to be transferred will be also specified. The bees can be either in the environment or in the nest.

The membrane structure is: $\mu = [_1[_2[_{i_1}]_{i_1} \ldots [_{i_p}]_{i_p}]_{i_2}[_{i_{p+1}}]_{i_{p+1}} \ldots [_{i_{n-2}}]_{i_{n-2}}]_1$.

There are seven different types of rules [14] but only two are presented below. The rules are distribute across the sets $R_i$.

- $(nectar, p, m, i) \longrightarrow (nectar, p, m, i)_2(0, p, m, i)_2^k$ according to this rule the nectar load of the foraging bee $i$ and $k$ copies indicating the source position are passed from the environment to the nest; the $k$ copies above simulate the information that is passed through the waggle dance to $k$ bees (the amount of nectar is not considered);
- $(nectar, p, m, i) \longrightarrow (nectar, p, m)_i$ - this rule shows that the bee $i$ (re)starts foraging after nectar has been given to the honey-bees in the hive.

The next model will approach the foraging problem from a different perspective. In this case instead of modelling the whole problem, we will approach different aspects and model them as separate X machines. At the end we will have a set of such machines. In [32] it is proposed a way of aggregating these components by providing a protocol for communication among them. We will illustrate here only the X machine which models the dancing behaviour.

The X machine $X$ will have the following elements:

- $\Sigma = \{fbee, space, nest, source\}$;
- $\Gamma = \{\text{"}dancing\text{"}, \text{"}flying\ out\text{"}, \text{"}flying\ in\text{"}, \text{"}source\ found\text{"}, \text{"}keep\ flying\text{"}\}$;
- $Q = \{in\ the\ nest, out\ of\ nest\}$;
- $M = \{(bee\_pos, source\_pos) \mid bee\_pos, source\_pos \in Fin\}$, where $Fin$ is a finite set of integer values $\{0, \ldots, Position\_max\}$;
- $\Phi$ contains
  - $dancing(fbee, (bee\_pos, source\_pos)) = ((bee\_pos, source\_pos), \text{"}dancing\text{"})$;
  - $fly\_out(space, (bee\_pos, source\_pos)) =$ $((bee\_pos', source\_pos), \text{"}flying\ out\text{"})$;
  - $fly\_in(nest, (bee\_pos, source\_pos)) = ((bee\_pos', source\_pos), \text{"}flying\ in\text{"})$;
  - $find\_source(source, (bee\_pos, source\_pos)) =$ $((source\_pos, source\_pos), \text{"}source\ found\text{"})$;
  - $keep\_fly\_out(space, (bee\_pos, source\_pos)) =$ $((bee\_pos', source\_pos), \text{"}keep\ flying\text{"})$.
- $F(in\ the\ nest, dancing) = in\ the\ nest$; $F(in\ the\ nest, fly\_out) = out\ of\ nest$; $F(out\ of\ nest, fly\_in) = in\ the\ nest$; $F(out\ of\ nest, find\_source) = out\ of\ nest$; $F(out\ of\ nest, keep\_fly\_out) = out\ of\ nest$;
- $I = \{in\ the\ nest\}; F = Q$.

In both systems either their rules (in the first case) or the X machine components (in the second case) run in parallel approaching the honey-bees behaviour as cooperative agents.

We will show how the formal models previously defined might be transformed into an executable code such as to be able to simulate the behaviour of the defined system.

A P system simulator has been implemented; it allows to input a P system specification and then to simulate its and highly parallel behaviour [3].

The MzScheme code for the P system model defined in the previous section, is presented below (only the rules for the environment are provided):

```
(define A (vector (n1 p1 m1)...(nk pk mk) (n1 p1 m1 3)... (n1 p1 m1 m)
                   ...
                  (nk pk mk 3)... (nk pk mk m)))
                                ;the alphabet codifying the system objects
(define MS ((1 2) (1 3) (1 4)... (1 p)
            (2 p+1)... (2 m)); the membrane structure
(define objects
  (vector ((n1 p1 m1 3)... (nh ph mh j)) )
                                ; initial configuration
(define rules
  (vector (((n p m i)->((n p m i) 2)((n p m i) 2)... ((n p m i) 2))
          ((n p m i)->((n p m) i)) ; rules above defined
   )
```

The X machine model is supported by a mark-up language called XMDL [32]. An XMDL fragment code of the X machine specification of the model previously defined is given below:

```
#input = {fbee, space, nest, source}.
#output = {"dancing", "flying out", "flying in", "source found",
           "keep flying"}.
#memory = {(b,s)| 0<=b<=Position, 0<=s<=Position}.
#state = {in_the_nest, out_of_nest}.
#fun dancing(?inp,(?b_p,?s_p)) =
      if ?inp==fbee then ((?b_p,?s_p),"dancing").
#fun fly_out(?space,(?b_p,?s_p)) =
      if next(?b_p,?s_p,?b_p',?s_p') then ((?b_p',?s_p'),"flying out").
```

From the above sample code we may notice the obvious similarity between their formats and the formalisms used by these models. Both tools are able to animate the specified systems and XMDL is also providing a sort of formal analysis of the system by generating test sets and supporting model checking analysis.

## 4 Eilenberg P Systems

There have been two kinds of attempts to investigate different links between P systems and X machines. On the one hand various ways of simulating P systems with replicated rewriting as stream X machines and communicating stream X machines with a communication matrix [2] and P systems with symbol objects as communicating stream X machines with ports and channels [33] have been investigated. On the other hand there have been defined some computational models called Eilenberg P systems (or Eilenberg P systems) that combine features of both P systems and X machines [7].

In general most of the existing P systems use a sort of regulated rewriting mechanism that can be a priority relationship, permitting/forbidding context, electrical charges, membrane permeability etc. [37]. In Eilenberg P systems this additional mechanism is provided as a transition diagram associated with an X machine giving birth to two classes of Eilenberg P systems, a sequential version called *Eilenberg P systems* and a parallel one, called *Parallel Eilenberg P systems.* In both variants, each transition has a specific set of evolution rules acting upon the string objects contained in different regions of the membrane system. The system will start in a given state and with an initial set of string objects. Given a state and a current set of string objects, in the case of Eilenberg P systems, the machine will evolve by applying rules associated with one of the transitions going out from the current state. The system will resume from the destination state of the current transition. In the parallel variant, instead of one state and a single set of string objects we may have a number of states, called *active states,* that are able to trigger outgoing transitions and such that each state hosts a different set of string objects; all the transitions emerging from every active state may be triggered once the rules associated with them may be applied; then the system will resume from the next states, which then become active states. Eilenberg P systems are models of cells evolving under various conditions when certain factors may inhibit some evolution rules or some catalysts may activate other rules. Parallel Eilenberg P systems introduce a parallel behaviour of the system in respect of the transitions emerging from active states, model cellular division and parallel development of the new born cells as well as cell collision when multiple transitions join a target state. The objects belonging to these systems are in this case string objects and the evolution rules are associated with such objects as well.

**Definition 3.** *An Eilenberg P system is a construct*

$$\Pi X = (\mu, X),$$

*where $\mu$ is a membrane structure consisting of $n$ membranes, with the membranes and the regions labelled in a one to one manner with the elements $1, \ldots, n$ and an $X$ machine whose memory is defined by the regions $1, \ldots, n$ of $\mu$. The $X$ machine is a system*

$$X = (V, \Gamma, Q, M_1, \ldots, M_n, \Phi, F, I),$$

*where*

- $V$ *is the* alphabet *of the system;*
- $\Gamma, Q, F$ *are as in Definition 2;* $\Gamma \subseteq V$, *is called now* terminal alphabet*;*
- $M_1^0, \ldots, M_n^0$ *are finite languages over $V$ and represent the initial values occurring in the regions $1, \ldots, n$ of the system; this is called the initial memory configuration of the system;*
- $\Phi = \{\Phi_1, \ldots, \Phi_p\}$, $\Phi_i = (R_{i,1}, \ldots R_{i,n})$, $1 \leq i \leq p$ *and $R_{i,j}$ is the set of evolution rules (possibly empty) associated with region $j$, of the form $A \to (u, tar)$, with* $A \in V$, $u \in V^*$, $tar \in \{here, out, in\}$;
- $I = \{q_0\}$, $q_0 \in Q$ *is the initial state; all the states are final states (equivalent to $Q = T$).*

The rules occurring in $R_{i,j}$ have only one target indication in their right hand side due to object strings manipulations. A *computation* in $\Pi X$ is defined as follows: it starts from the initial state $q_0$ and an initial configuration of the memory defined by $M_1^0, \ldots M_n^0$ and proceeds iteratively by applying in parallel rules in all regions, processing in each one all the strings that can be rewritten; in a given state $q$, each string of the current memory component $M_i$, $1 \leq i \leq n$ is processed by a single rule following the target indication of that rule (for instance, when rewriting $xAv$ by a rule $A \to (u, tar)$, the string $xuv$ obtained will be send to the region indicated by $tar$, with the usual meaning in P systems [37]); if several rules may be applied to a string, then one rule and one symbol to which it is applied are randomly chosen; the rules are from a component $\Phi_i$ which is associated with one of the transitions emerging from the current state $q$ and the resulting strings constitute the new configuration of the memory, $M_1', \ldots, M_n'$; 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 $\Gamma$) is collected outside of the system at the end of a halting computation.

Parallel Eilenberg P systems have the same underlying construct $(\mu, X)$, with the only difference that instead of one single membrane structure, it deals with a set of instances having the same organization ($\mu$), but being distributed across the system. More precisely, these instances are associated with states called *active states;* these instances can divide up giving birth to more instances or collide into single elements depending on the current configuration of the active states and the general topology of the underlying machine. Initially only $q_0$ is an active state and the initial memory components associated with $q_0$ are $M_1^0, \ldots, M_n^0$. All active states are processed in parallel in one step: all emerging transitions from these states are processed in parallel (and every single transition processes in parallel each string object in each region, if evolution rules match them).

*Cell division:* if $q_j$ is one of the active states, $M_{j,1}, \ldots, M_{j,n}$ is its associated memory components, and $\Phi_{j,1}, \ldots, \Phi_{j,t}$ are $\Phi'$s components associated with the emerging transitions from $q_j$, then the rules occurring in $\Phi_{j,i}$, $1 \leq i \leq t$, are applied to the string objects from $M_{j,1}, \ldots, M_{j,n}$, the control passes onto $q_{j,1}, \ldots, q_{j,t}$, which are the target states of the transitions earlier nominated, with $M_{j,1,1}, \ldots, M_{j,n,1}, \ldots, M_{j,1,t}, \ldots, M_{j,n,t}$, their associated memory components, obtained from $M_{j,1}, \ldots, M_{j,n}$, by applying rules of $\Phi_{j,1}, \ldots, \Phi_{j,t}$; the target states become active states,

$q_j$ is desactivated and $M_{j,1}, \ldots, M_{j,n}$ vanish. Only $\Phi_{j,i}$ components that have rules matching the string objects of $M_{j,1}, \ldots, M_{j,n}$, are triggered and consequently only their target states become active and associated with memory instances $M_{j,1,i}, \ldots, M_{j,n,i}$. If none of $\Phi_{j,i}$ is triggered then in the next step $q_j$ is desactivated and $M_{j,1}, \ldots, M_{j,n}$ vanish too. If some of $\Phi_{j,i}$ indicate the same component of $\Phi$ then the corresponding memory configurations $M_{j,1,i}, \ldots, M_{j,n,i}$ are the same as well; this means that always identical transitions emerging from a state yield the same result.

*Cell collision:* if

$\Phi_1, \ldots, \Phi_t$ enter the same state $r$ and some or all of them emerge from active states, then the result associated with $r$ is the union of membrane instances produced by those $\Phi_i'$s emerging from active states and matching string objects from their membrane instances.

A computation of an Eilenberg P (Parallel Eilenberg P) system halts when none of the rules associated with the transitions emerging from the current states (active states) may be applied.

Like most of the variants of P systems, both Eilenberg P and Parallel Eilenberg P systems are computationally complete. More precisely it has been shown that Eilenberg P systems with either one membrane, three states and eight sets of rules or two membranes, one state and seven sets of rules are computationally complete [7]. On the other hand any Parallel Eilenberg P system may be simulated by an Eilenberg P system. When symbol objects are replacing string objects then the corresponding Eilenberg P systems with at least one membrane, two states and four sets of rules compute no more than Parikh images of ET0L languages [8].Parallel Eilenberg P system are very effective in solving some hard problems. For example SAT problem may be solved in time linear with the number of clauses and number of variables by Parallel Eilenberg P systems using string objects [7] or symbol objects [8].

These results show that Eilenberg P systems represent a very promising area of investigation providing a wide front of research both on its theoretical side and in using them as models of various biological phenomena.

## 5 Conclusions

The work reported in this paper refers to two computational models that are proposed to modelling the behaviour of social insects, in particular honey-bee colonies. The models are parallel distributed paradigms with specific communication mechanisms and are supported by software tools able to animate the systems specified in these frameworks.

These approaches have some similarities: they are computational devices with components running in a fully parallel manner and acting as a society of agents. For both models the design is flexible and reusable as the whole system may be built from individual components (multisets and rules in the membrane approach and component X machines in the second case) that are assembled and enriched with the communication aspects. There are also some important differences between the two approaches. In the case of the membrane systems the outputs are defined only at the end of the computation and in some specified components; the inputs are defined only in some variants, the memory is implicitly defined as being the set of symbols associated to each component. For X machines the inputs and the outputs are explicitly associated to every basic function and a memory element is part of any computation.

A new computational model, called Eilenberg P system, has been also introduced. The sequential and the Parallel Eilenberg P systems are presented and results reflecting the computational power of these models and their effectiveness in solving NP-complete problems are briefly mentioned.

**Acknowledgements.** The authors wish to thank the anonymous referees for their helpful comments on a first version of this paper.

## References

1. L.M. Adleman, Molecular Computation of Solutions to Combinatorial Problems, *Science,* 226:1021-1024 (1994)
2. J. Aguado, T. Bălănescu, M. Gheorghe, M. Holcombe, F. Ipate, P Systems with Replicated Rewriting and Stream X Machines, *Fundamenta Informaticae,* 49:17-33 (2002)
3. D. Balbotín Noval, M. J. Pérez Jiménez, F. Sancho Caparrini, A MzSceme Implementation of Transition P Systems. In *Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Springer LNCS Vol.2597, (2003), 58-73
4. J-P. Banatre, D. Le Metayer, The Gamma Model and its Discipline of Programming, *Science of Computer Programming*, 15:55-77 (1990)
5. J. Barnard, COMX: a Design Methodology using Communicating X-machines, *Journal of Information and Software Technology,* 40:271-280 (1998)
6. T. Bălănescu, T. Cowling, H. Georgescu, M. Gheorghe, M. Holcombe, C. Vertan, Communicating Stream X Machines Are no more than X Machines, *Journal of Universal Computer Science* 9:497-502 (1999)
7. T. Bălănescu, M. Gheorghe, M. Holcombe, F. Ipate, Eilenberg P Systems, in *Membrane Computing. International Workshop, WMC-CdeA 2002,* Springer LNCS Vol.2597, (2003), 43-57
8. F. Bernardini, M. Gheorghe, M. Holcombe, Eilenberg P Systems with Symbol Objects (submitted 2003)
9. G. Berry, G. Boudol, The Chemical Abstract Machine, *Theoretical Computer Science,* 96:217-248 (1992)
10. M. Cavaliere, C. Martin-Vide, Gh. Păun, Brainstorming Week on Membrane Computing, Rovira i Virgili University, Tarragona, (2003)
11. N. Chomsky, *Syntactic Structures,* Mouton, The Hague, (1957)

12. G. Ciobanu, Distributed Algorithms over Communicating Membrane Systems, *BioSystems,* 70:123-134 (2003)
13. S. Eilenberg, *Automata, Languages and Machines,* Academic Press, (1974)
14. M. Gheorghe, M. Holcombe, P. Kefalas, Computational Models of Collective Foraging, *BioSystems,* 61:133-141 (2001)
15. J.-L. Giavitto, O. Michel, Modelling the Topological Organization of Cellular Processes, *BioSystems,* 70:149-164 (2003)
16. R. Gregory, R. Paton, J. Saunders, Q. H. Wu, A Model of Bacterial Adaptability based on Multiple Scales of Interaction: COSMIC, in *Computing in Cells and Tissues,* (R.C. Paton et al eds.), Springer, Series on Natural Computing, 2003 (to appear).
17. D. Harel, Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming,* 8:231-274 (1987)
18. M. Holcombe, X-machines as a Basis for Dynamic System Specification, *Software Engineering Journal,* 3(2):69-76 (1988)
19. M. Holcombe, X-machines in Computing, Biology and Art, In: *Proceedings International Workshop Grammar Systems,* R. Freund, A. Kelemenova (eds.), Silesian University at Opava, Bad Ischl, (2000) 343-346.
20. M. Holcombe, Computational Models of Cells and Tissues: Machines, Agents and Fungal Infection, *Briefings in Bioinformatics,* 2:271-278 (2001)
21. M. Holcombe, K. Bogdanov, M. Gheorghe, Functional Test Generation for Extreme Programming, *Proceedings 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering,* Italy, (2001), 109-113.
22. M. Holcombe, F. Ipate, *Correct Systems. Building a Business Process Solution,* Springer, Series on Applied Computing, (1998)
23. http://www.cellml.org
24. http://psystems.disco.unimib.it
25. http://www.sbml.org
26. http://www.wi.leidenuniv.nl/ pier/dna.html
27. http://www.wisdom.weizmann.ac.il/ aviv/
28. K. Jensen, *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use,* Vol.1-3, Springer, Berlin, (1992, 1994, 1997)
29. S. Jones, *The Language of the Genes,* Flamingo, London, (Revised edition), (2000)
30. N. Kam, I. R. Cohen, D. Harel, The Immune System as a Reactive System: Modelling T Cell Activation with Statecharts, Technical report MCS01-09, The Weizmann Institute of Science, Israel, (2001)
31. S. A. Kauffman, it The Origins of Order. Self-organization and Selection in Evolution, Oxford University Press, (1993)
32. P. Kefalas, Formal Modelling of Reactive Agents as an Aggregation of Simple Behaviours, In: I.P. Vlahavas, C.D. Spyropoulos (eds.), Springer LNAI Vol.2308 (2002), 461-472
33. P. Kefalas, G. Eleftherakis, M. Holcombe, M. Gheorghe, Simulation and Verification of P Systems through Communicating X Machines, *BioSystems,* 70:135-148 (2003)
34. P. Kefalas, M. Holcombe, G. Eleftherakis, M. Gheorghe, A Formal Method for the Development of Agent Based Systems, In *Intelligent Agent Software Engineering,* V. Plekhanova (ed.), Idea Group Publishing, (2003), 68-98
35. H. Kitano, Computational Systems Biology, *Nature,* 420:206-210 (2002)
36. Gh. Păun, Computing with Membranes, *Journal of Computer System Sciences,* 61:108-143 (2000). Also, Turku Center for Computer Science TUCS Report No.208, (1998), http://www.tucs.fi
37. Gh. Păun, *Membrane Computing. an Introduction,* Springer, Berlin, (2002)
38. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing - New Computational Paradigms,* Springer, Berlin, (1998)

39. M. Peleg, I. Yeh, R.B. Altman, Modeling Biological Processes Using Workflow and Petri Net Models, *Bioinformatics,* 18:825-837 (2002)
40. D. B. Searls, The Language of Genes, *Nature* 420:211-217 (2002)
41. C. Tofts, Describing Social Insect Behaviour using Process Algebra, *Transaction of Society for Computer Simulation* (1993), 227-283
42. Y. Suzuki, J. Takabayashi, H. Tanaka, Investigations of an Ecological System Using an Abstract Rewriting System of Multisets, In: *Recent Topics in Mathematical and Computational Linguistics*, Gh. Păun (ed.), The Publishing House of the Romanian Academy, Bucharest, (2000), 300-309
43. H. de Vries, J. C. Biesmeijer, Modelling Collective Foraging by means of Individual Behaviour Rules in Honey-bees, *Behavioral Ecology and Sociobiology,* 44:109-124 (1998)