Three Classes of Alphabetic Flat Splicing P Systems

Rodica Ceterchi[1][0000−0001−5354−8588],
Marian Gheorghe[2][0000−0002−2409−4959],
Lakshmanan Kuppusamy[3][0000−0003−2358−905X], and
K.G. Subramanian[4,*][0000−0001−8726−5850]

[1]University of Bucharest, Faculty of Mathematics and Computer Science,
14 Academiei St, 010014 Bucharest, Romania.
[2]Department of Computer Science, University of Bradford,
West Yorkshire, Bradford BD7 1DP, UK.
[3]School of Computer Science & Engineering, VIT, Vellore - 632 014, India.
[4] School of Mathematics, Computer Science and Engineering, Liverpool Hope University, Hope Park, Liverpool L16 9JD, UK. ([*] Honorary Visiting Professor)

Abstract. In this paper alphabetic flat splicing operations are considered in the context of cell, tissue and Eilenberg P systems. The computational power of these new models is investigated with respect to complexity measures, such as number of membranes, for all of them, and number of states for Eilenberg P systems. Applications on generating chain code picture languages with these formalisms are also presented.

Keywords: Alphabetic Flat Splicing Cell and Tissue Membrane Systems · Alphabetic Flat Splicing Eilenberg Membrane Systems · Computational Power · Chain Code Picture Languages

1 Introduction

The operation of splicing was introduced by Head [16] in developing an abstract model of the bio-chemical DNA recombination in the presence of restriction enzymes. Subsequently there has been a lot of interest and research on splicing of linear and circular strings [32, 10, 9, 7], as well as on DNA computing [30].

Motivated by a splicing operation on circular strings, the notion of flat splicing on linear strings was introduced by Berstel et al. [6].

Research on flat splicing on strings has been continued by considering a special class of splicing, called alphabetic splicing. In this case P systems with rewriting using regular or linear rules and alphabetic flat splicing rules are considered [26] and their generative power compared to that of alphabetic flat splicing systems, as reported in [6], and to Chomsky hierarchy of families of languages. The generative power of P systems with rewriting, having up to three membranes and using alphabetic flat splicing rules, has been investigated in [25] and matrix grammars with flat splicing rules introduced and studied in [11]. The first two

research investigations previously mentioned combine two bio-inspired computer science concepts, splicing operations with membrane computing.

The bio-inspired computing model with the generic name of P system was introduced in the area of membrane computing by Păun, first in a technical report [27] and then in a journal paper [28]. There has been a tremendous growth in research in this area, both in theory and applications in various areas, with several variants of the basic model of P system having been introduced and investigated. Some of the early theoretical results have been reported in a research textbook [29] and then key theoretical developments and diverse applications have been presented in a comprehensive handbook [31]. A survey paper covers the most significant research results published up until 2020 [33].

A variant of P systems where the rewriting rules are distributed across the transitions of an X machine, called Eilenberg P Systems (EPS) or PX Systems, was introduced in [8] for a smaller set of string rewriting operations and then extended in [1]. A variant of this string rewriting model running strings in parallel was introduced and investigated in [4]. In [5] EPS with symbol objects have been investigated. Other lines of research, on the interaction between P systems and X-machines, have focused on mapping various classes of P systems into X-machines [20, 24], on using EPS computation paradigm for modelling biological systems with dynamic structure [34] or on developing testing methods, similar to those investigated for EPS models, for systems using various classes of membrane computing models [17, 18].

In this paper we extend the research on P systems with rewriting using alphabetic flat splicing [26] and their relationships with Chomsky hierarchy [25], by considering not only cell-like P systems ($AFS.PS$), but also tissue P system models ($AFS.TPS$). A variant of Eilenberg P systems which uses alphabetic flat splicing rules, $AFS.EPS$, is also introduced and investigated.

The key contributions and main results of the paper are: (i) introduction of a new alphabetic flat splicing rewriting mechanism for Eilenberg P systems, $AFS.EPS$, and extension of the existing alphabetic flat splicing P system model, $AFS.PS$, to tissue P systems, $AFS.TPS$; (ii) investigation of the computational power of these models with respect to descriptional complexity measures such as number of membranes and states - the latter for Eilenberg P systems; (iii) relationships amongst classes of languages computed by these computational models with 2 and 3 membranes; (iv) the proof that the families of languages computed by specific classes of $AFS.PS$ and $AFS.TPS$, with m, $m \geq 2$, membranes, called one flow and rule distinguishable, are included in the family of languages computed by $AFS.EPS$ with 2 membranes; and (v) applications computing chain code picture languages with the models introduced here.

The structure of the paper is the following: after this Introduction, in Section 2 are presented key concepts and notations utilised in the paper; Section 3 introduces the main definitions of alphabetic flat splicing tissue (cell) P systems and Eilenberg P systems; examples illustrating the working of the newly introduced alphabetic flat splicing systems are presented in Section 4; the main results are discussed in Section 5 and applications showing how to compute chain

code picture languages are described in Section 6; finally, conclusions and further investigations are mentioned in Section 7.

2 Preliminaries

We assume that the readers are familiar with formal languages and P systems. We refer to [28, 31] for the latter and below we recall some basic definitions and notations for the former.

Let V be a finite set of characters (or symbols), called *alphabet* and let V^* denotes the free monoid generated by the set of all strings over the alphabet V under the concatenation operation. A string (or word) $x = a_1 a_2 \dots a_n \in V^*$, with $a_i \in V, 1 \leq i \leq n$, is a finite sequence of symbols and the number of characters of x, i.e, n, denotes the length of x. Let λ denotes the empty string, with length 0. Let $V^+ = V^* - \{\lambda\}$. A string v is a substring of $x \in V^*$ if there are strings $u, w \in V^*$ such that $x = uvw$. The Chomsky family of languages (and their classes of grammars) are : REG (*regular*), LIN (*linear*), CF (*context-free*), CS (*context-sensitive*) and RE (*recursively enumerable*). The family of finite languages is denoted by FIN. The *Chomsky hierarchy* of family of languages is given by $FIN \subset REG \subset LIN \subset CF \subset CS \subset RE$.

Splicing systems use some different operations from the rules of the grammars mentioned above, as illustrated by the current literature of this area [16, 32, 10, 9, 7]. However, here the notion we adopted was considered in [25]. First, we consider some basic operations of flat splicing on words from [6]. A flat splicing rule r is of the form $(\alpha|\gamma - \delta|\beta)$, where α, β, γ and δ are words over V and are called *handles* of r. For two strings $x = u\alpha\beta v$ and $y = \gamma w \delta$, with $u, v, w \in V^*$, an application of the flat splicing rule $r = (\alpha|\gamma - \delta|\beta)$ to the pair (x, y) yields the string $z = u\alpha\gamma w\delta\beta v$ and we write $(x, y) \vdash_r z$. This means, the application of the rule r inserts the second word y between α and β in the first word x yielding the resultant string z. When all the handles of a rule r are symbols in V or the empty string λ, then the flat splicing rule r is called *alphabetic*.

A *flat splicing system* (shortly, FSS) [6] is given as $\mathcal{S} = (V, I, R)$, where V is an alphabet, I is an initial set of strings over V and R is a finite set of flat splicing rules. The language L generated by $\mathcal{S}$ is the smallest language containing I and such that for any two strings $x, y \in L$ and any rule $r \in R$, if the rule r is applicable to the pair (x, y) producing z, that is, if $(x, y) \vdash_r z$, then z is also in L. When all the rules of FSS $\mathcal{S}$ are alphabetic, then $\mathcal{S}$ is called an *alphabetic flat splicing system* (shortly, $AFSS$). When I of a FSS ($AFSS$) $\mathcal{S}$ is a finite set, a regular language or a context-free language, then FSS ($AFSS$) $\mathcal{S}$ is called finite, regular or context-free, respectively. The families of languages generated by FSS ($AFSS$), with initial sets of strings being finite, regular or context-free, are denoted by $\mathcal{L}(FSS, X)$ ($\mathcal{L}(AFSS, X)$), for X being FIN, REG or CF, respectively.

It is known that $\mathcal{L}(AFSS, CF)$ is included in the family of context-free languages [6]. In other words, alphabetic flat splicing rules and context-free initial sets will produce only context-free languages.

3 Basic Definitions

The following definition is very similar to that in [25].

Definition 1. *An Alphabetic Flat Splicing Tissue P System (AFS.TPS) with m membranes is a tuple*

$$\Pi = (V, \mu, F_1, \ldots F_m, R_1, \ldots, R_m, i_0)$$

where the membranes are in a bijective relation to $1, \ldots, m$; *V is a finite set of symbols, denoting the system's* alphabet*; μ is the* membrane structure, *a graph with m nodes, each of them corresponding to a membrane and edges describing connections between membranes, i.e., $\mu = (A, E)$, with $A = \{1, \ldots, m\}$ and $E \subseteq \{\{x, y\} | x, y \in A\}$; $F_1, \ldots F_m$ are the* initial finite subsets *of V^* of the m membranes; $R_1, \ldots, R_m$ are* finite sets of splicing rules *associated with the m membranes; and i_0 is the label of the* output membrane.

Each splicing rule of a membrane i, $1 \leq i \leq m$, $r : (i, (\alpha|\gamma - \delta|\beta), t)$, has a target indicator, t, the label of a membrane connected with i, as indicated by μ. The two membrane labels that appear in r, are called labels of the *host membrane*, i, and *target membrane*, t. When such a rule, r, is applied to a pair of strings, (x, y), $x = u\alpha\beta v, y = \gamma w\delta$ from membrane i and producing the result $z = u\alpha\gamma w\delta\beta v$, i.e, $(x, y) \vdash_r z$, then z is sent to membrane t, as per the target indicator of r, if $\{i, t\} \in E$. The string z is kept in the same membrane i, if $t = i$. In this case the target indicator may be omitted while describing the splicing rule. The rules are applied in a *maximal parallel* mode, which means that in every membrane, all the strings that can evolve must do so. If more than a rule is applicable to a string then one of them is non-deterministically chosen. The parallel behaviour of these systems is similar to that of many string rewriting P systems, i.e., both at the system level and in each region [28, 29, 31].

Remark 1. In [25] the membrane structure is a tree and the rules are defined in a slightly different form. Instead of membrane labels, the target indicators of the rules are from the set $\{in, out, here\}$.

Some preliminary concepts are introduced in order to define a computation. A *configuration* is a tuple $(\omega_1, \ldots, \omega_m)$, where ω_i, $1 \leq i \leq m$, is a finite set of strings $\{w_{i,j} | w_{i,j} \in V^*, 1 \leq j \leq n_i\}$ belonging to membrane i. For two configurations $c = (\omega_1, \ldots, \omega_m)$ and $c' = (\omega'_1, \ldots, \omega'_m)$, if c' is obtained from c by evolving c in a maximal parallel manner, then this is denoted by $c \models c'$ and is called a *transition*. More precisely, for the configuration $c = (\omega_1, \ldots, \omega_m)$, in each membrane i, $1 \leq i \leq m$, where $\omega_i = \{w_{i,j} | w_{i,j} \in V^*, 1 \leq j \leq n_i\}$, k_i of the strings from ω_i may evolve, $0 \leq k_i \leq n_i$. Let us denote them $w_{i,j_1}, \ldots, w_{i,j_{k_i}}$. For each w_{i,j_h} there is a string x_{i,j_h} in membrane i and a rule r_{i,j_h} such that $(w_{i,j_h}, x_{i,j_h}) \vdash_{r_{i,j_h}} z_{i,j_h}$, $h \in \{j_1, \ldots, j_{k_i}\}$. Each z_{i,j_h} is sent to a certain region according to the target indicator of r_{i,j_h}. In this way are obtained the sets of strings ω'_i, $1 \leq i \leq m$, that appear in c'.

A *computation* of an $AFS.TPS$ is a sequence of configurations, $c_0, c_1, \ldots, c_n$ where $c_0 = (F_1, \ldots, F_m)$ and for any consecutive configurations, c_i, c_{i+1}, $0 \leq i \leq n-1$, $c_i \models c_{i+1}$. A *halting computation* is a computation arriving in a configuration, c_n, where no rule is applicable to it.

The language *defined* (*generated* or *computed*) by an $AFS.PS$, Π, as in Definition 1, denoted by $L(\Pi)$, is composed of the initial strings from the output membrane, F_{i_0}, and the strings obtained through halting computations in the output membrane, i.e., any $w_{i_0,j}$, $1 \leq j \leq n_{i_0}$, such that there is a halting computation $c_0 = (F_1, \ldots, F_m) \models \ldots \models c_n = (\omega_1, \ldots, \omega_m)$, where $\omega_{i_0} = \{w_{i_0,1}, \ldots, w_{i_o,n_{i_0}}\}$.

If we indicate by $AFS.TPS(m)$ the $AFS.TPS$ with at most m, $m \geq 1$, membranes, then the corresponding family of generated languages is denoted by $\mathcal{L}(AFS.TPS(m))$.

Remark 2. We also consider the case when the membrane structure is a *tree* and such a model is called *Alphabetic Flat Splicing P System* ($AFS.PS$). Similar to the above mentioned model, we indicate by $AFS.PS(m)$ the $AFS.PS$ with at most m, $m \geq 1$, membranes, and the corresponding family of generated languages is $\mathcal{L}(AFS.PS(m))$.

Now, we turn our attention to Eilenberg P systems (EPS) and refer to their definition from [8]. We introduce a variant of Eilenberg P system which has alphabetic flat splicing rules in the membranes of the P system. The working of the resulting P system is similar to an EPS except for a few differences, namely, the output is collected in an inner membrane and some or all the states of the system can be final states, while in an EPS the result is collected outside the system and all the states of the system are final states. We call the new variant *Alphabetic Flat Splicing Eilenberg P System* ($AFS.EPS$). Also, all the strings that reside in an output membrane, when the computation halts, constitute the language generated by the $AFS.EPS$.

Definition 2. *An Alphabetic Flat Splicing Eilenberg P System (AFS.EPS) is a tuple*

$$\Pi^e = (V, \mu, Q, F_1, \ldots F_m, \mathcal{C}, \delta, q_0, F, i_0)$$

where V is a finite set of symbols, denoting the system's alphabet*; μ is the* membrane structure *with m membranes, as in Definition 1; Q is a finite set of* states*; $F_1, \ldots F_m$ are finite subsets of V^*, denoting the sets of* initial strings *of the m membranes; $\mathcal{C}$ is a finite set of* components*, defined over $R_1, \ldots, R_m$, the* finite sets of splicing rules *associated with the m membranes, $\mathcal{C} = \{C_1, \ldots, C_p\}$. Each component has the form $C_i = (R_{i,1}, \ldots R_{i,m})$ and $R_{i,j} \subseteq R_j$, $1 \leq j \leq m$, for $1 \leq i \leq p$. δ is a* transition function*, $\delta : Q \times \mathcal{C} \longrightarrow Q$; $q_0 \in Q$ is the* initial state*; $F \subseteq Q$ is the set of* final states*; and i_0 the output membrane.*

A *computation* of an $AFS.EPS$ is given by a sequence of configurations and states. It starts from the initial configuration $c_0 = (F_1, \ldots, F_m)$ and initial state q_0. The system evolves in a maximal parallel manner in the following way:

given configurations $c = (\omega_1, \ldots, \omega_m)$, $c' = (\omega'_1, \ldots, \omega'_m)$ and states $q, q' \in Q$, the $AFS.EPS$ evolves in a maximal parallel manner from the configuration c and state q into configuration c' and state q' if there is a component $C_i \in \mathcal{C}$, $1 \leq i \leq p$, such that $\delta(q, C_i) = q'$ and in at least one membrane j, $1 \leq j \leq m$, there are rules in $R_{i,j}$ applied in parallel to the strings from ω_j. All the strings that can evolve in configuration c and state q must evolve.

A computation *halts* when the system reaches a final state and none of the rules of the components associated with the state through the transition function is applicable.

The language *defined* (*generated* or *computed*) by an $AFS.EPS$, Π^e, as in Definition 2, denoted by $L(\Pi^e)$, is composed of the initial strings from the output membrane, F_{i_0}, and the strings obtained through halting computations in the output membrane, i.e., $w_{i_0,j}$, $1 \leq j \leq n_{i_0}$, such that there is a halting computation $c_0 = (F_1, \ldots, F_m) \models \ldots \models c_n = (\omega_1, \ldots, \omega_m)$ and a sequence of states $q_0, \ldots, q_n$, and components $C_{i_0}, \ldots, C_{i_n}$, with $\delta(q_h, C_{i_h}) = q_{h+1}$, $0 \leq h \leq n-1$, q_n is a final state and $\omega_{i_0} = \{w_{i_0,1}, \ldots, w_{i_o,n_{i_0}}\}$.

If we indicate the three parameters, namely, the number of membranes, at most m, the number of states, at most s, and the number of components, at most p, then we denote the system by $AFS.EPS(m, s, p)$ and the corresponding family of generated languages by $\mathcal{L}(AFS.EPS(m, s, p))$.

4 Examples

Example 1. Consider the $AFS.PS(3)$

$$\Pi_1 = (\{a, x, y, z, t\}, [_1[_2]_2[_3]_3]_1, \{xayaza, a\}, \{a\}, \{a, t\}, R_1, R_2, R_3, 3).$$

The sets of rules are: $R_1 = \{r_1 : (1, (x|a - \lambda|a), 2), r_2 : (1, (x|a - \lambda|a), 3)\}$; $R_2 = \{r_3 : (2, (y|a - \lambda|a), 1)\}$; $R_3 = \{r_4 : (3, (z|a - \lambda|a), 1), r_5 : (3, (z|t - \lambda|a))\}$. Any halting computation starts in membrane 1, from $xayaza$, and stops in membrane 3, which is the output membrane. In the first step of the computation one can use, in membrane 1, either rule r_1 or r_2. They produce the same result, $xaayaza$, but r_1 sends it to membrane 2 whereas r_2 to membrane 3. In the next step, if membrane 2 has been chosen, then rule r_3 is applied and the result, $xaayaaza$, is resent back to membrane 1. Otherwise, if membrane 3 has been chosen then either r_4 or r_5 is utilised. In the first case the string $xaayazaa$ is obtained and resent to membrane 1, whereas in the second one, the string $xaayazta$ is obtained and kept in membrane 3, where the computation stops, as no more rule is applicable. One can observe that the string from membrane 1 where a symbol a is added to it between x and a is then sent either to membrane 2 or 3, where a symbol a is inserted between y and a or between z and a, respectively, and resent back to membrane 1. The computation stops in membrane 3 when the rule r_5 inserts a symbol t between z and a and no more rule is applicable in membrane 3 which contains the string. The strings obtained through halting computations are of the form $xa^{p+q}ya^pzta^q$, $p, q \geq 1$. Hence, the language computed by Π_1,

including the initial strings in membrane 3, is $L(\Pi_1) = \{xa^{p+q}ya^pzta^q | p, q \geq 1\} \cup \{a, t\}$.

Example 2. Let us consider the $AFS.TPS(3)$

$$\Pi_2 = (\{a, x, y, z, t\}, \mu, \{xayaza, a\}, \{a\}, \{a, t\}, R_1, R_2, R_3, 3).$$

The membrane structure, a graph, is given by $\mu = (\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}, \{3, 1\})$. The sets of rules are: $R_1 = \{r_1 : (1, (x|a-\lambda|a), 2)\}$; $R_2 = \{r_2 : (2, (y|a-\lambda|a), 3)\}$; $R_3 = \{r_3 : (3, (z|a - \lambda|a), 1), r_4 : (3, (z|t - \lambda|a))\}$.

This example is very similar to Example 1. The membranes are connected by a graph structure, μ, and the rules are adapted accordingly. As in Example 1, any computation starts in membrane 1 from the initial string $xayaza$ and in the first step the rule r_1 is applied leading to $xaayaza$ which is sent to membrane 2. In the second step the rule r_2 is applied and the string $xaayaaza$ is generated and sent to membrane 3. In the next step either rule r_3 or r_4 is applied. In the first case the string obtained, $xaayaazaa$ is sent to membrane 1. In the other case, the rule r_4 is applied and the string $xaayaazta$ obtained remains in membrane 3, which is output membrane. The language generated by Π_2 is $L(\Pi_2) = \{xa^{n+1}ya^{n+1}zta^n | n \geq 1\} \cup \{a, t\}$.

Example 3. In this example, $AFS.EPS$ with just two membranes are built such that they compute the languages obtained in previous examples, $L(\Pi_1)$ and $L(\Pi_2)$.

For $L(\Pi_1)$ one considers the $AFS.EPS(2, 3, 5)$

$$\Pi_1^e = (\{a, x, y, z, t\}, \mu, \{xayaza, a\}, \{a, t\}, \{q_1, q_2, q_3\}, \mathcal{C}, \delta, q_1, \{q_3\}, 2).$$

This $AFS.EPS$ has two membranes. In membrane 1 is computed what is computed in the $AFS.PS$ in membranes 1 and 2, and membrane 2 of the $AFS.EPS$ corresponds to membrane 3 of the $AFS.PS$. The membrane structure is $\mu = (\{1, 2\}, \{\{1, 2\}\})$. The components are from the set $\mathcal{C} = \{C_i | 1 \leq i \leq 5\}$. The components are defined based on the rules of Π_1, slightly modified. The sets of rules of Π_1^e are $R_1' = \{r_1' : (1, (x|a-\lambda|a)), r_2' : (1, (x|a-\lambda|a), 2), r_3' : (1, (y|a-\lambda|a))\}$, in membrane 1, and $R_2' = \{r_4' : (2, (z|a-\lambda|a), 1), r_5' : (2, (z|t-\lambda|a))\}$, in membrane 2. The components are defined by $C_1 = (\{r_1'\}, \emptyset)$, $C_2 = (\{r_3'\}, \emptyset)$, $C_3 = (\{r_2'\}, \emptyset)$, $C_4 = (\emptyset, \{r_4'\})$, $C_5 = (\emptyset, \{r_5'\})$ and the transition function by $\delta(q_1, C_1) = q_2$, $\delta(q_2, C_2) = q_1$, $\delta(q_1, C_3) = q_3$, $\delta(q_3, C_4) = q_1$, $\delta(q_3, C_5) = q_3$. In membrane 1 are inserted symbols a between x and a either using component C_1 or C_3. In the first case an a is inserted between y and a, in membrane 1, through component C_2. In the second case the string is sent to membrane 2, where a symbol a is inserted between z and a, using component C_4, and the string resent back to membrane 1, or a symbol t is inserted between z and a, using component C_5, and the computation stops. Clearly, $L(\Pi_1^e) = L(\Pi_1)$.

The $AFS.EPS$ presented below generates the language $L(\Pi_2)$ computed by the $AFS.TPS$ in Example 2.

The following $AFS.EPS(2,3,4)$ is considered

$$\Pi_2^e = (\{a,x,y,z,t\}, \mu, \{xayaza, a\}, \{a,t\}, \{q_1,q_2,q_3\}, \mathcal{C}, \delta, q_1, \{q_3\}, 2).$$

The membrane structure is as in Π_1^e. The set $\mathcal{C}$ has four components, based on the following sets of rules corresponding to the two membranes, obtained from those of Π_2, with adequate changes referring to the labels of the host and target membranes. The sets of rules are $R_1' = \{r_1' : (1, (x|a - \lambda|a)), r_2' : (1, (y|a - \lambda|a), 2)\}$, in membrane 1 and $R_2' = \{r_3' : (2, (z|a - \lambda|a), 1), r_4' : (2, (z|t - \lambda|a))\}$, in membrane 2. The components are $C_1 = (\{r_1'\}, \emptyset)$, $C_2 = (\{r_2'\}, \emptyset)$, $C_3 = (\emptyset, \{r_3'\})$, $C_4 = (\emptyset, \{r_4'\})$ and the transition function by $\delta(q_1, C_1) = q_2$, $\delta(q_2, C_2) = q_3$, $\delta(q_3, C_3) = q_1$, $\delta(q_3, C_4) = q_3$. Clearly, $L(\Pi_2^e) = L(\Pi_2)$.

5 Main Results

The result below is a direct consequence of Definitions 1 and 2.

Theorem 1. *The following relationships hold:*
(i) $\mathcal{L}(AFS.ZPS(m)) \subseteq \mathcal{L}(AFS.ZPS(m+1)), Z \in \{T, \lambda\}, m \geq 1$.
(ii) $\mathcal{L}(AFS.EPS(m,s,p)) \subseteq \mathcal{L}(AFS.EPS(m',s',p')), m' \geq m \geq 1,\ s' \geq s \geq 1,$ $p' \geq p \geq 1$.

The following result shows the relationship between $AFS.PS$, $AFS.TPS$ and $AFS.EPS$ with the same number of membranes.

Theorem 2. $\mathcal{L}(AFS.PS(m)) \subseteq \mathcal{L}(AFS.TPS(m)) \subseteq \mathcal{L}(AFS.EPS(m,1,1)) \subseteq \mathcal{L}(AFS.EPS(m,s,p)),\ m,s,p \geq 1$.

Proof. The first inclusion $\mathcal{L}(AFS.PS(m)) \subseteq \mathcal{L}(AFS.TPS(m))$, follows immediately from the fact that the tree structure is a particular case of a graph structure.

Let Π be from $AFS.TPS(m)$, $m \geq 1$,

$$\Pi = (V, \mu, F_1, \ldots, F_m, R_1, \ldots, R_m, i_0).$$

Then the following $AFS.EPS$ is constructed

$$\Pi^e = (V, \mu, \{q\}, F_1, \ldots, F_m, \mathcal{C}, \delta, q, \{q\}, i_0),$$

where $\mathcal{C} = \{C_1\}$, $C_1 = (R_1, \ldots, R_m)$ and $\delta(q, C_1) = q$.

Obviously, Π^e belongs to the family $AFS.EPS(m,1,1)$ and $L(\Pi) = L(\Pi^e)$.

The final inclusion follows from Theorem 1 (ii).

In [25] it has been proved $\mathcal{L}(AFS.PS(1)) \subset \mathcal{L}(AFS.PS(2))$ and $\mathcal{L}(AFS.PS(3)) - CF \neq \emptyset$, using for rules targets in, out and $here$. These results remain true when membrane labels are used instead. In [6] it has proved that $\mathcal{L}(AFSS, CF)$ is included in the family of context-free languages, whereas the later result above shows that $\mathcal{L}(AFS.PS(3))$ contains non context-free languages.

Now, we establish relationships between the families $\mathcal{L}(AFS.PS(2))$, $\mathcal{L}(AFS.PS(3))$ and $\mathcal{L}(AFS.TPS(3))$. We start with a preliminary result.

Lemma 1. $\mathcal{L}(AFS.TPS(3)) \setminus \mathcal{L}(AFS.PS(3)) \neq \emptyset$

Proof. Let us consider the language $L = \{xa^{n+1}ya^nzta^n | n \geq 1\} \cup \{a\}$ and the following $AFS.TPS(3)$

$$\Pi = (\{a, x, y, z, t\}, \mu, \{a, xayaza\}, \{a, t\}, \{a\}, R_1, R_2, R_3, 3),$$

where $\mu = (\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}, \{3, 1\}\})$. R_1 contains the rule $r_1 : (1, (x|a - \lambda|a), 2)$; R_2 contains $r_2 : (2, (y|a - \lambda|a), 3)$; $r_3 : (2, (z|t - \lambda|a), 3)$; and R_3 contains $r_4 : (3, (z|a - \lambda|a), 1)$. The language generated is $L(\Pi) = L$. The computation starts in membrane 1 with $(xayaza, a) \vdash_{r_1} xa^2yaza$ and continues executing n times the sequence of rules r_2 in R_2, r_4 in R_3, and r_1 in R_1, $n \geq 0$. This computation leads to the string $xa^{n+1}ya^nza^n$, $n \geq 1$. After that, r_3 is applied in R_2 introducing t between z and a and the string is sent to membrane 3, where the computation stops with the string $xa^{n+1}ya^nzta^n$, $n \geq 1$.

We prove now that $L \notin \mathcal{L}(AFS.PS(3))$.

Let us suppose the opposite, $L \in \mathcal{L}(AFS.PS(3))$. In this case there is an $AFS.PS(3)$, Π', with three membranes such that $L(\Pi') = L$. The membrane structure may be either (i) $[_1[_2[_3]_3]_2]_1$ or (ii) $[_1[_2]_2[_3]_3]_1$, with one of the membranes being the output membrane. The output membrane contains initial string a and possibly $xa^{p+1}ya^pzta^p$, for p from a finite set of positive integer numbers. If we denote by α the maximum length of the initial strings, ρ the total number of rules of Π', and τ the maximum number of symbols inserted by any of the rules, then let us denote $k = \alpha + \tau\rho$. Let us now consider the string, $w = xa^{n+1}ya^nzta^n$, from L, where $n > k$. The string is obtained in a halting computation starting from an initial string, w_0, and applying a finite sequence of rules. Let us also denote by v one of the following strings z, t, zt. In any halting computation starting from w_0 and leading to w, one must get $w_0 \models^* w' \models^+ w$, where $w' = x'a^{h_1}ya^{h_2}va^{h_3}$, $x' = x$ or $x' = \lambda$ and $0 \leq h_i < k$, $1 \leq i \leq 3$. In the sequel we refer to each of the substrings between x' and y, y and v, and after v as substrings in positions 1, 2 and 3, respectively. Indeed, if one of the substrings a^{h_i}, $1 \leq i \leq 3$, has more than k occurrences of a and one, both or none of y, v are present then there is a rule, r, that is applied twice when the substring is obtained and this can be repeated an arbitrary number of times increasing arbitrarily that substring and leading to a halting computation producing a string with more occurrences of a on that position i than w and the other positions with the same occurrences as w.

In the computation $w' \models^+ w$, one must get $w'' = x'a^{h'_1}ya^{h'_2}va^{h'_3}$, with at least one h'_i, $1 \leq i \leq 3$, such that $h'_i \geq k$. Similar to the case above, a rule, r, is applied a least twice to position i, and the sequence of rules between the two occurrences of r can be repeated. In the sequence of rules between the two occurrences of r there must be at least two rules, r', r'', applied to the other two positions. Otherwise, repeating the sequence will increase the number of occurrences only for some positions. Each of these rules must the have target indicator different from the label of the membrane where it belongs to. Otherwise, they can be indefinitely applied.

In the case of the membrane structure (i) $[_1[_2[_3]_3]_2]_1$, one distinguishes two cases: (i.1) if two of the rules appear in the same membrane then in the sequence above one can replace one of the rules with the other one and consequently creating an unbalance between the corresponding positions and leading to a string that is not in L; (i.2) if the rules appear in distinct membranes then one considers the rules in membranes 2 and 3; let us assume the first rule is in 2 and the second in 3 and let them be either r', r'' or r'', r. In any of these two circumstances before the rule in 2 and between those in membranes 2 and 3, there must be a rule from membrane 1, r_1', with target indicator 2 and another rule from membrane 1, r_1'', with target indicator 3, respectively. One can replace one of the r_1', r_1'' with the other one and this will increase the number of occurrences for one position and decrease for the other one, leading to the same conclusion as above.

In the case of the membrane structure (ii) $[_1[_2]_2[_3]_3]_1$ the rules r, r', r'' appear in distinct membranes and one of the rules is in membrane 2; then there must be another rule in membrane 2 within the sequence r, r', r'', r such that these two occurrence have different targets, 1 and 3. These can then be replaced one with the other one and leading to the same conclusion as above.

Theorem 3. *The following relationships hold*
(i) $\mathcal{L}(AFS.PS(3)) \subset \mathcal{L}(AFS.TPS(3))$.
(ii) $\mathcal{L}(AFS.TPS(2)) \subset \mathcal{L}(AFS.TPS(3))$.

Proof. The first part follows directly from Theorem 2 and Lemma 1.

For the second part, from Theorem 1, we have $\mathcal{L}(AFS.ZPS(2)) \subseteq \mathcal{L}(AFS.ZPS(3))$, $Z \in \{T, \lambda\}$. Also, $\mathcal{L}(AFS.PS(2)) = \mathcal{L}(AFS.TPS(2))$, as $AFS.PS(2)$ coincides with $AFS.TPS(2)$. These combined with the first part of the Theorem, lead to the result (ii).

The next two results show relationships between particular classes of $AFS.PS$, $AFS.TPS$ and $AFS.EPS$.

Theorem 4. $\mathcal{L}(AFS.PS(2)) \subset \mathcal{L}(AFS.EPS(2,3,3))$.

Proof. The inclusion follows from Theorem 2. Now, we show the first inclusion.

First, we show that $L = \{xa^nzta^n | n \geq 1\}$ is in $\mathcal{L}(AFS.EPS(2,3,3))$ and not in $\mathcal{L}(AFS.PS(2))$.

Let us consider the following $AFS.EPS$ from $AFS.EPS(2,3,3)$

$$\Pi^e = (\{a, x, z, t\}, [_1[_2]_2]_1, \{q_0, q_1, q_2\}, \{xaza, a\}, \emptyset, \mathcal{C}, \delta, q_0, \{q_2\}, 2),$$

with the set of rules in membrane 1, R_1, consisting of $r_1 : (1, (x|a - \lambda|a))$, $r_2 : (1, (z|a - \lambda|a))$, and $r_3 : (1, (z|t - \lambda|a), 2)$; and the empty set of rules in membrane 2. The components are $C_1 = (\{r_1\}, \emptyset)$, $C_2 = (\{r_2\}, \emptyset)$, $C_3 = (\{r_3\}, \emptyset)$; and the transition function $\delta(q_0, C_1) = q_1$, $\delta(q_1, C_2) = q_0$, $\delta(q_0, C_3) = q_2$. Clearly, $L(\Pi^e) = L$.

Let us suppose $L \in \mathcal{L}(AFS.PS(2))$. In this case there is an $AFS.PS(2)$, Π' with two membranes, labelled 1 and 2. One can assume that the membrane

structure is $[_1[_2]_2]_1$, with 2 being the output membrane. Similar to the proof of Lemma 1 in any halting computation from an axiom to $w = xa^nzta^n$, for a large enough n, each rule from a membrane increasing the number of occurrences of a of one of the two positions, must alternate with a rule from the other membrane increasing the occurrences of a from the other position. Hence, the output membrane must contain at least an initial string a^p, for some $p \geq 1$, that will be then in the language. But, L does not contain such a string.

Corollary 1. $\mathcal{L}(AFS.TPS(2)) \subset \mathcal{L}(AFS.EPS(2,3,3))$.

Theorem 5. $(\mathcal{L}(AFS.TPS(3)) \setminus \mathcal{L}(AFS.TPS(2))) \cap \mathcal{L}(AFS.EPS(2,4,4)) \neq \emptyset$.

Proof. Let us consider the language from the proof of Lemma 1, $L = \{xa^{n+1}ya^nzta^n | n \geq 1\} \cup \{a\}$, which satisfies $L \in \mathcal{L}(AFS.TPS(3)) \setminus \mathcal{L}(AFS.TPS(2))$.

In order to complete the proof we show that $L \in \mathcal{L}(AFS.EPS(2,4,4))$.

Let us consider the following $AFS.EPS$ from $AFS.EPS(2,4,4)$,

$$\Pi^e = (\{a,x,y,z,,\}[_1[_2]_2]_1, \{q_0,q_1,q_2,q_3\}, \{xayaza,a,t\}, \{a\}, \mathcal{C}, \delta, q_0, \{q_3\}, 2),$$

with the set of rules in membrane 1, R_1, consisting of $r_1 : (1, (x|a - \lambda|a))$, $r_2 : (1, (y|a - \lambda|a))$, $r_3 : (1, (z|a - \lambda|a))$, $r_4 : (1, (z|t - \lambda|a), 2)$; and no rules in membrane 2. The components are $C_i = (\{r_i\}, \emptyset)$, $1 \leq i \leq 4$; and the transition function $\delta(q_0, C_1) = q_1$, $\delta(q_1, C_2) = q_2$, $\delta(q_2, C_3) = q_0$, $\delta(q_1, C_4) = q_3$. Clearly, $L(\Pi^e) = L$.

We consider now a class of $AFS.ZPS$, $Z \in \{T, \lambda\}$, called *one flow and rule distinguishable.* Subsequently we refer to Definition 1.

Definition 3. *An $AFS.ZPS$, $Z \in \{T, \lambda\}$, is called* one flow *$AFS.ZPS$ if at any step of the computation there is a membrane containing all the evolving strings.*

The membrane containing the evolving strings during the computation is not necessarily unique. Not all the strings stop evolving at the same moment. A rule that leads to a non-evolving string is called *final rule.*

The membrane where any computation starts is called *starting membrane.*

Definition 4. *An $AFS.ZPS$, $Z \in \{T, \lambda\}$, is called* rule distinguishable *$AFS.ZPS$ if in any rule one of the handles is an initial string and for any rule $r_i : (i, (\alpha|\gamma - \delta|\beta), j)$ from R_i, and any initial string $x_i = \gamma w \delta$ of a membrane i, $1 \leq i \leq m$, there is no rule $r_{i'} : (i', (\alpha|\gamma' - \delta'|\beta), j')$ from $R_{i'}$, and any initial string $x_{i'} = \gamma' w' \delta'$ of a membrane i', $1 \leq i' \leq m$, $i' \neq i$, such that $x_{i'} \neq x_i$..*

An AFS.ZPS is rule distinguishable when any two rules from different membranes, i, i', that are applicable between the same symbols α, β, must not use different initial strings, $x_i, x_{i'}$.

An $AFS.ZPS$, $Z \in \{T, \lambda\}$, satisfying the above definitions is called *one flow and rule distinguishable $AFS.ZPS$.*

Example 4. Let the following one flow and rule distinguishable $AFS.TPS(3)$

$$\Pi_{o,d} = (\{x, a, c, y, b\}, \mu, F_1, F_2, F_3, R_1, R_2, R_3, 3)$$

where $\mu = (\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}, \{3, 1\}\})$; $F_1 = \{xa, c, yb\}$, $F_2 = \{d, t\}$, $F_3 = \{a, b, t\}$; $R_1 = \{r_1 : (1, (x|c - \lambda|a, 2), r_2 : (1, (y|c - \lambda|b, 2)\}$, $R_2 = \{r_3 : (2, (x|d - \lambda|c, 3), r_4 : (2, (x|t - \lambda|c, 3), r_5 : (2, (y|d - \lambda|c, 3)\}$, $R_3 = \{r_6 : (3, (x|a - \lambda|d, 1), r_7 : (3, (y|b - \lambda|d, 1), r_8 : (3, (y|t - \lambda|d))\}$; and 3 is the output membrane.

$\Pi_{o,d}$ is one flow $AFS.TPS$, with starting membrane 1. Indeed, the computation starts with xa and yb in membrane 1 and in the first step one gets xca and ycb, respectively, by using r_1 and r_2. These strings are then sent to membrane 2, as per the target indicators of r_1 and r_2. The computation leads to $xtca(dca)^n, n \geq 0$, and $yt(dcb)^n, n \geq 1$. This computation develops in parallel for the two strings until for one of them a final rule is applied, r_4, for the first string, or r_8, for the second one. These final rules are not applied at the same step, i.e., one of the strings may stop in membrane 3, whereas the other one continues to travel through the system until the final rule will stop the computation in the same output membrane. One can notice that when the final rules are applied, the result may not follow the flow of the other string, as it does no longer evolve. The language computed by $\Pi_{o,d}$ is $L(\Pi_{o,d}) = \{xtca(dca)^n | n \geq 0\} \cup \{yt(dcb)^n | n \geq 1\} \cup \{a, b, t\}$, i.e., the strings generated in one flow manner, plus the initial strings from the output membrane.

$\Pi_{o,d}$ is also rule distinguishable $AFS.TPS$ as rules $(i, \alpha|\gamma - \delta|\beta), j)$ with the same symbols α, β appear only in the same membrane, $\alpha = x, \beta = c$ in 2 (r_3, r_4) and $\alpha = y, \beta = d$ in 3 (r_7, r_8).

The following $AFS.TPS(3)$

$$\Pi_{o,nd} = (\{x, a, c, y, b\}, \mu, F_1, F_2, F_3, R_1, R_2, R_3, 3)$$

where $\mu = (\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}, \{3, 1\}\})$; $F_1 = \{xa, ac, yb, c\}$, $F_2 = \{a, d, t\}$, $F_3 = \{a, b, t\}$; $R_1 = \{r_1 : (1, (x|a - c|a, 2), r_2 : (1, (y|c - \lambda|b, 2)\}$, $R_2 = \{r_3 : (2, (x|a - \lambda|a, 3), r_4 : (2, (x|t - \lambda|a, 3), r_5 : (2, (y|d - \lambda|c, 3)\}$, $R_3 = \{r_6 : (3, (x|a - \lambda|a, 1), r_7 : (3, (y|b - \lambda|d, 1), r_8 : (3, (y|t - \lambda|d))\}$; and 3 is the output membrane.

$\Pi_{o,nd}$ is one flow $AFS.TPS$ with starting membrane 1, but is not rule distinguishable as rule r_1 uses initial string ac in membrane 1 and r_3 uses a in 2, between x and a.

The following $AFS.TPS(3)$

$$\Pi_{no,nd} = (\{x, a, c, y, b\}, \mu, F_1, F_2, F_3, R_1, R_2, R_3, 3)$$

where $\mu = (\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}, \{3, 1\}\})$; $F_1 = \{xa, ac, c\}$, $F_2 = \{a, t, yc, d\}$, $F_3 = \{a, b, t\}$; $R_1 = \{r_1 : (1, (x|a - c|a, 2), r_2 : (1, (y|c - \lambda|b, 2)\}$, $R_2 = \{r_3 : (2, (x|a - \lambda|a, 3), r_4 : (2, (x|t - \lambda|a, 3), r_5 : (2, (y|d - \lambda|c, 3)\}$, $R_3 = \{r_6 : (3, (x|a - \lambda|a, 1), r_7 : (3, (y|b - \lambda|d, 1), r_8 : (3, (y|t - \lambda|d))\}$; and 3 is the output membrane.

$\Pi_{no,nd}$ is not one flow $AFS.TPS$ as a computation starts in membrane 1, from xa, and another one in 2, from yc. It is also not rule distinguishable, for the same argument used for $\Pi_{o,nd}$.

Now, one can show that an $AFS.ZPS(m)$, $Z \in \{T, \lambda\}$, $m \geq 2$, can be simulated by and $AFS.EPS$ with two membranes. This result shows that potential hierarchies of the languages generated by $AFS.ZPS(m)$, $Z \in \{T, \lambda\}$, $m \geq 2$, are all generated by $AFS.EPS$ with two membranes.

Theorem 6. *For any one flow and rule distinguishable $AFS.ZPS(m)$, with $Z \in \{T, \lambda\}$, $m \geq 2$, Π, there is an $AFS.EPS(2, m, m^2)$, Π^e, such that $L(\Pi) = L(\Pi^e)$.*

Proof. It is enough to consider the case of one flow and rule distinguishable $AFS.TPS$ with m, $m \geq 2$, membranes.

Let us have the following $AFS.TPS$

$$\Pi = (V, \mu, F_1, \ldots, F_m, R_1, \ldots, R_m, i_0).$$

The following $AFS.EPS$ is constructed

$$\Pi^e = (V, \mu, Q, F_1', F_2', \mathcal{C}, \delta, q_0, F, 2),$$

with 2 membranes, labelled 1 and 2, with 2 corresponding to i_0; $\mu = (\{1, 2\}, \{\{1, 2\}\})$; Q has m states, each of them associated with a membrane of Π, $Q = \{q_1, \ldots, q_m\}$; F_1' contains the initial strings of $F_1, \ldots, F_m$, but F_{i_0}, whereas $F_2' = F_{i_0}$.

For each set of rules R_i, $1 \leq i \leq m$, one considers the sets $R_{i,i_k} = \{r | r \in R_i$, r is not final rule and its target indicator is $i_k\}$, $1 \leq k \leq n_i$. When rules from R_{i_k}, $1 \leq k \leq n_i$, are applied to strings in membrane i, the results are all sent to membrane i_k. $R_{i,f}$ denotes the set of all final rules that appear in R_i. R'_{i,i_k} denotes $R_{i,i_k} \cup R_{i,f}$. Every set R'_{i,i_k} includes the final rules, as they lead to results that do no longer evolve and, when appear, do not have to disturb the flow of those that continue to evolve by moving to membrane i_k. The strings resulted from final rules go to the target indicators of them and these may differ from i_k. R_i is equal to $R'_{i_1} \cup \ldots \cup R'_{i_{n_i}}$. The non-final rules from R'_{i,i_k} have the form $(i, (\alpha|\gamma - \delta|\beta), i_k)$. For the final rules from R'_{i,i_k} the membrane indicators may be different from i_k. One denotes by R''_{i,i_k} the set $\{(e, (\alpha|\gamma - \delta|\beta), f) | (i, (\alpha|\gamma - \delta|\beta), t) \in R'_{i,i_k}$ where $e = 1$ if $i \neq i_0$, otherwise $e = 2$ and $f = 1$ if $t \neq i_0$, otherwise $f = 2\}$; t is i_k for non-final rules and may be different from i_k when the rule is final. In each set R''_{i,i_k} the rules use the membrane labels of Π^e, i.e., either 1 when they indicate a membrane different from i_0, or 2 otherwise.

From the construction so far of the sets of rules for Π^e, it follows that the number of rules of this $AFS.EPS$ is the same as the number of rules of Π.

The set of components, $\mathcal{C}$, is $\{C_{i,i_k} | C_{i,i_k} = (R'_{i,i_k}, \emptyset)$ when $i \neq i_0$, $C_{i,i_k} = (\emptyset, R'_{i,i_k})$ when $i = i_0$, $1 \leq i \leq m, 1 \leq k \leq n_i \leq m\}$.

The maximum number of components is m^2.

The transition function is defined by $\delta(q_i, C_{i,i_k}) = q_{i_k}$, $q_i, q_{i_k} \in Q$, $1 \leq i \leq m$, $1 \leq k \leq n_i$.

$q_0 = q_h$, $q_h \in Q$, and h is the starting membrane and $F = \{q_{i_0}\}$.

The idea of the construction above is the following: the contents and rules of all the membranes of Π, but i_0, are available in membrane 1 of Π^e and those from i_0 are contained in membrane 2. The rules of Π are modified such that the labels of the membranes in Π are mapped into the two membranes of Π^e. The AFS.EPS, Π^e, will have a distinct state for each membrane. The component, $C_{s,d}$, utilised by the transition function in a state, q_s, associated with membrane s of Π, contains the rules of the membrane s of Π having the same target indicator, membrane d, and maybe final rules. In this case $\delta(q_s, C_{s,d}) = q_d$, where q_d is the state in Π^e corresponding to membrane d. One flow constraints imposes that all the evolving strings are always in one membrane and the rule distinguishable property of Π allows $AFS.EPS$, Π^e, to use in membrane 1, where the contents of all the membranes of Π, but i_0, are included, exactly the rules that correspond to those of Π.

The computation in Π performs the first step from the starting membrane, denoted above h. As Π is one flow $AFS.TPS$, all the strings that evolve in the first step are in h. The rules applicable to them have the same target indicator h_k, unless are final rules. All the rules are from R'_{h,h_k} and send the results to h_k. In Π^e the computation starts in membrane 1, if $h \neq i_0$, or 2, otherwise. The transition function will execute $\delta(q_h, C_{h,h_k}) = q_{h_k}$, allowing only rules from R''_{h,h_k} to be utilised. If $h_k \neq i_0$ then the first step ends in membrane 1, otherwise in 2. As the current state of Π^e corresponds to the membrane containing the strings and Π is a rule distinguishable AFS.TPS, the rules applicable from R''_{h,h_k} use only initial strings from h. The above is true for any membrane of Π containing the evolving strings and likewise for Π^e. Hence, $L(\Pi^e) = L(\Pi)$.

Remark 3. The $AFS.EPS$ built in the proof of the above theorem, has 2 membranes, m states and at most m^2 components over a set of rules with the cardinal equal to that of the set of rules of the $AFS.ZPS$, $Z \in \{T, \lambda\}$.

6 Applications - Chain Code Picture Languages

Chain code pictures are composed of straight horizontal and vertical unit lines in the 2D plane. These can be codified as words (strings) over the alphabet $V = \{u, d, r, l\}$, where each character stands for a unit line in the corresponding direction: *up, down, right, left.* Since this codification has been introduced in [23], many studies have been developed of families of such pictures, seen as languages, and many generative methods have been proposed for them (see for instance [13] and [14]), and even parallel rewriting in the context of P systems [12].

Here we propose to generate two such languages, the *double stairs* and next the *diamond*, with $AFS.ZPS$ systems, $Z \in \{T, \lambda\}$, and $AFS.EPS$ systems.

The *double stairs* language is $\{(ur)^n(rd)^n \mid n \leq 1\}$.

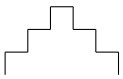

Figure 2: A chain code picture of two "stairs" of equal height, represented by the chain code word $(ur)^3(dl)^2d$

We will generate a slightly modified language of stairs, with a longer "platform" on top. The *diamond* is the above double stairs, completed with its mirror image w.r.t. the horizontal axis.

Let us denote by Π_1 the following $AFS.PS$ with 3 membranes

$$(V, [_1[_2]_2[_3]_3]_1, F_1, F_2, \emptyset, R_1, R_2, \emptyset, 3\}),$$

where $V = \{u, d, r, l\}$, $F_1 = \{urrd, ru, r\}$ and $F_2 = \{dr\}$, $R_1 = \{r_1 : (1, (u|r - u|r), 2), r_2 : (1, (r|\lambda - r|r), 3)\}$ and $R_2 = \{r_3 : (2, (r|d - r|d), 1)\}$. The output membrane is 3, which has no initial strings and rules, just collecting the final results sent by r_2. The computation starts from $urrd$ in membrane 1. Using n times the rules r_1 in membrane 1 and r_3 in membrane 2, one gets $(ur)^{n+1}(rd)^{n+1}$, $n \geq 0$, in membrane 1. Finally, r_2 is applied yielding $(ur)^{n+1}r(rd)^{n+1}$, in membrane 3, $n \geq 0$, where the computation halts. Clearly, $L(\Pi_1) = \{(ur)^n r(rd)^n \mid n \geq 1\}$.

The next solution to the double stairs (half a diamond) chain code picture problem is based on an $AFS.EPS$ with two membranes computing the language $L = \{(ur)^n r(rd)^n \mid n \geq 1\}$.

Let us consider the $AFS.EPS(2, 3, 3)$

$$\Pi_1^e = ([_1[_2]_2]_1, \{u, d, r, l\}, \{q_1, q_2, q_3\}, X_1, X_2, \mathcal{C}, \delta, q_1, 2, \{q_3\}),$$

where $X_1 = \{urrd, ru, r, dr\}$, $X_2 = \emptyset$, the set of components, $\mathcal{C} = \{C_1, C_2, C_3\}$, is based on the following sets of rules: $R_1 = \{r_1 : (1, (u|r - u|r)), r_2 : (1, (r|\lambda - r|r), 2), r_3 : (1, (r|d - r|d))\}$, $R_2 = \emptyset$. The components are $C_1 = (\{r_1\}, \emptyset)$, $C_2 = (\{r_3\}, \emptyset)$, $C_3 = (\{r_2\}, \emptyset)$, and the transition function is $\delta(q_1, C_1) = q_2$, $\delta(q_2, C_2) = q_1$, $\delta(q_1, C_3) = q_3$. In Π_1^e the computation takes place in membrane 1, and membrane 2 is just a place collecting the results, with no initial strings and rules. Clearly, $L(\Pi_1^e) = L$.

In the application above we have used from the alphabet V only the symbols u, d, r allowing to build the upper part of the diamond. Next, we consider generating the entire diamond and for getting the lower part of it we use the symbols u, d, l. So, we make full use of V for generating this chain code picture.

First, we consider an $AFS.TPS$ for generating the diamond chain code picture language.

Consider the $AFS.TPS(6)$

$$\Pi_2 = (V, \mu, F_1, \ldots, F_6, R_1, \ldots, R_6, 6\}),$$

where $\mu = (\{1, \ldots, 6\}\{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}, \{1, 5\}, \{5, 6\}\})$ and the sets of initial strings are $F_1 = \{urrddllu, ru, r\}$, $F_2 = \{dr\}$, $F_3 = \{ld\}$, $F_4 = \{ul\}$, $F_5 = \{l\}$ and $F_6 = \emptyset$. The sets of rules are $R_1 = \{r_1 : (1, (u|r - u|r), 2), r_5 : (1, (r|\lambda - r|r), 5)\}$, $R_2 = \{r_2 : (2, (r|d - r|d), 3)\}$, $R_3 = \{r_3 : (3, (d|l - d|l), 4)\}$, $R_4 = \{r_4 : (4, (l|u - l|u), 1)\}$, $R_5 = \{r_6 : (2, (l|\lambda - l|l), 3)\}$, $R_6 = \emptyset$. The computation starts in membrane 1 with the initial string $urrddllu$,

by applying either r_1 or r_5. If r_1 is applied one gets the string $ururrddllu$ which is sent to membrane 2. The following rules are applied in membranes 2, 3 and 4: r_2, r_3 and r_4, respectively, getting then the string $ururrdrddldllulu = (ur)^2(rd)^2(dl)^2(lu)^2$. The sequence of rules r_1, r_2, r_3, r_4 can be applied several times until one gets $(ur)^{n+1}(rd)^{n+1}(dl)^{n+1}(lu)^{n+1}$, where the sequence above has been applied a total of n times, $n \geq 0$. Finally, if r_5 is applied the string obtained $(ur)^{n+1}r(rd)^{n+1}(dl)^{n+1}(lu)^{n+1}$ is sent to membrane 5 where rules r_6 is applied and the result, $(ur)^{n+1}r(rd)^{n+1}(dl)^{n+1}l(lu)^{n+1}$, is sent to the output membrane 6. Clearly, the diamond chain code picture language is computed by the $AFS.TPS(6)$, Π_2, i.e., $L(\Pi_2) = \{(ur)^n r(rd)^n (dl)^n l(lu)^n | n \geq 1\}$.

One can built an $AFS.EPS$ with two membranes, precisely $AFS.EPS(2, 6, 6)$, that will compute the same diamond chain code picture language.

$$\Pi_2^e = ([_1[_2]_2]_1, \{u, d, r, l\}, \{q_1, q_2, q_3, q_4, q_5, q_6\}, X_1, X_2, \mathcal{C}, \delta, q_1, 2, \{q_6\}).$$

The sets of initial strings are $X_1 = \{urrddllu, ru, r, dr, ld, ul, l\}$ and $X_2 = \emptyset$. The sets of rules are $R_1 = \{r_1 : (1, (u|r - u|r)), r_2 : (1, (r|d - r|d)), r_3 : (1, (d|l - d|l)), r_4 : (1, (l|u - l|u)), r_5 : (1(r|\lambda - r|r)), r_6 : (1, (l|\lambda - l|l), 2)\}$ and $R_2 = \emptyset$. The set of components is $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$, where $C_i = (\{r_i\}, \emptyset)$, $1 \leq i \leq 6$. The transition function is $\delta(q_1, C_1) = q_2$, $\delta(q_2, C_2) = q_3$, $\delta(q_3, C_3) = q_4$, $\delta(q_4, C_4) = q_1$, $\delta(q_1, C_5) = q_5$, $\delta(q_5, C_6) = q_6$. q_1 is the initial state, 2 is the output membrane and q_6 is the final state. Clearly, the language computed by Π_2^e through halting computations starting from membrane 1, state q_1 and ending in membrane 2, state q_6, is the diamond chain code picture language, $\{(ur)^n r(rd)^n (dl)^n l(lu)^n \mid n \geq 1\}$.

Remark 4. In the applications above the $AFS.ZPS$, $Z \in \{T, \lambda\}$, are all one flow and rule distinguishable, allowing to build corresponding $AFS.EPS$ with two membranes.

7 Conclusions and Further Work

In this paper we extended the research on P systems with rewriting using regular or linear rules and alphabetic flat splicing rules considered in [26] and that on the generative power of P systems with rewriting, having up to three membranes and using alphabetic splicing rules [25], by considering tissue P system models and a variant of Eilenberg P systems which use alphabetic flat splicing rules.

We have investigated the computational power of these models, relationships amongst classes of languages computed by such models with 2 and 3 membranes and have shown that the families of languages computed by one flow and rule distinguishable $AFS.PS$ and $AFS.TPS$ with m, $m \geq 2$, membranes are included in the family of languages computed by $AFS.EPS$ with 2 membranes. Finally, applications computing chain code picture languages with the models introduced here are presented. We conjecture that the result provided in Theorem 3, $\mathcal{L}(AFS.TPS(2)) \subset \mathcal{L}(AFS.TPS(3))$, can be extended to

$\mathcal{L}(AFS.TPS(n)) \subset \mathcal{L}(AFS.TPS(n+1))$, for arbitrary $n, n \geq 2$, and Theorem 6 remains true for $AFS.EPS$ with m, $m \geq 2$, membranes if they are one flow and rule distinguishable.

There are several future research directions that may be considered in connection with the results reported in this paper. Arbitrary flat splicing rules [6] can be considered together with rewriting P systems. Also, as various other splicing operations have been identified and studied [16, 32, 10, 9, 7], these are worth studying in the context of P systems (and Eilenberg P systems). The tissue P systems investigated in the paper have the membranes connected through a ring graph. What happens when the topology of the graph is different, for instance, nodes with more than two edges connected to them? Insertion and deletion systems have been studied in [19, 22, 35] and insertion systems investigated in connection with flat splicing systems in [6]. Insertion and deletion have been further studied as regulated mechanisms [15, 2] and in the context of P systems research [21, 3]. Connections between such classes of insertion-deletion systems and alphabetic flat splicing tissue P systems (and Eilenberg P systems) represent another line of further research. Another interesting research direction is given by the relationships between various classes of flat splicing P systems and matrix grammars using flat splicing operations investigated in [11].

Acknowledgements. MG's and LK's work has been partially supported by the Royal Society grant IES\R3\213176, 2022-2024.

References

1. Aguado, J., Bălănescu, T., Gheorghe, M., Holcombe, M., Ipate, F.: P systems with replicated rewriting and stream X-machines. Fundamenta Informaticae **49**, 17–33 (2002)
2. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: Regulated insertion-deletion systems. Journal of Automata, Languages and Combinatorics **27**, 15–45 (2022)
3. Alhazov, A., Krassovitskiy, A., Rogozhin, Y., Verlan, S.: P systems with minimal insertion and deletion. Theoretical Computer Science **412**, 136–144 (2011)
4. Bernardini, F., Gheorghe, M., Holcombe, M.: PX systems = P systems + X machines. Natural Computing **2**, 201–213 (2003)
5. Bernardini, F., Gheorghe, M., Holcombe, M.: Eilenberg P systems with symbol-objects. In: Jonoska, N. et al. (ed.) Aspects of Molecular Computing, Essays Dedicated to Tom Head on the Occasion of His 70th Birthday, *LNCS*, vol. 2950, pp. 49–60. Springer (2004)
6. Berstel, J., Boasson, L., Fagnot, I.: Splicing systems and Chomsky hierarchy. Theoretical Computer Science **436**, 2–22 (2012)
7. Bonizzoni, P., de Felice, C., Fici, G., Zizza, R.: On the regularity of circular splicing languages: a survey and new developments. Natural Computing **9**, 397–420 (2010)
8. Bălănescu, T., Gheorghe, M., Holcombe, M., Ipate, F.: Eilenberg P systems. In: Păun, Gh. et al. (ed.) Membrane Computing: WMC - CdeA 2002, *LNCS*, vol. 2597, pp. 43–67. Springer (2002)
9. Ceterchi, R.: An algebraic characterization of semi-simple splicing. Fundamenta Informaticae **73**, 19–25 (2006)

10. Ceterchi, R., Martín-Vide, C., Subramanian, K.G.: On some classes of splicing languages. In: Jonoska, N. et al. (ed.) Aspects of Molecular Computing, Essays Dedicated to Tom Head on the Occasion of His 70th Birthday, *LNCS*, vol. 2950, pp. 84–105. Springer (2004)
11. Ceterchi, R., Pan, L., Song, B., Subramanian, K.G.: Matrix flat splicing. In: Gong, M. et al. (ed.) Bio-inspired Computing - Theories and Applications: BIC-TA Xi'an 2016, *CCIS*, vol. 681, pp. 54–63. Springer (2016)
12. Ceterchi, R., Subramanian, K.G., Venkat, I.: P systems with parallel rewriting for chain code picture languages. In: Beckmann, A. et al. (ed.) Evolving Computability, CiE 2015, *LNCS*, vol. 9136, pp. 145–155. Springer (2015)
13. Dassow, J., Habel, A., Taubenberger, S.: Chain-code pictures and collages generated by hyperedge replacement. In: Cuny, J. et al. (ed.) Graph Grammars 1994, *LNCS*, vol. 1073, pp. 412–427. Springer (1994)
14. Drewes, F.: Some remarks on the generative power of collage grammars and chain-code grammars pp. 1–14 (2000)
15. Fernau, H., Kuppusamy, L., Raman, I.: On path-controlled insertion–deletion systems. Acta Informatica **56**, 35–59 (2019)
16. Head, T.: Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviours. Bulletin of Mathematical Biology **49**(6), 739–756 (1987)
17. Ipate, F., Gheorghe, M.: A model learning based testing approach for spiking neural P systems. Theoretical Computer Science **924**, 1–16 (2022)
18. Ipate, F., Niculescu, I.M., Lefticaru, R., Konur, S., Gheorghe, M.: A model learning based testing approach for kernel P systems. Theoretical Computer Science **966**, 113975 (2023)
19. Kari, L.: On insertion and deletion in formal languages. Tech. rep., Ph. D. Thesis, University of Turku (1991)
20. Kefalas, P., Eleftherakis, G., Holcombe, M., Gheorghe, M.: Simulation and verification of P systems with communicating X-machines. BioSystems **70**, 135–148 (2003)
21. Krishna, S.N., Rama, R.: Insertion-deletion P systems. In: Jonoska, N. et al. (ed.) DNA Computing, 7th International Workshop on DNA-Based Computers, 2001, *LNCS*, vol. 2340, pp. 360–370. Springer (2002)
22. Margenstern, M., Păun, Gh., Rogozhin, Y., Verlan, S.: Context-free insertion-deletion systems. Theoretical Computer Science **330**, 339–348 (2005)
23. Maurer, H.A., Rozenberg, G., Welzl, E.: Using string languages to describe picture languages. Information and Control **54**, 155–185 (1982)
24. Niculescu, I.M., Gheorghe, M., Ipate, F., Ştefănescu, A.: From kernel P systems to X-machines and FLAME. Journal of Automata, Languages and Combinatorics **19**, 239–250 (2014)
25. Pan, L., Song, B., Nagar, A., Subramanian, K.G.: Language generating alphabetic flat splicing P systems. Theoretical Computer Science **724**, 28–34 (2018)
26. Pan, L., Song, B., Subramanian, K.G.: Rewriting P systems with flat splicing rules. In: Leporati, A. et al. (ed.) Membrane Computing: CMC 2016, *LNCS*, vol. 10105, pp. 340–351. Springer (2016)
27. Păun, Gh.: Computing with membranes. Tech. rep., Turku Centre for Computer Science (1998). URL `http://tucs.fi/publications/view/?pub_id=tPaun98a`
28. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693. URL `https://doi.org/10.1006/jcss.1999.1693`

29. Păun, Gh.: Membrane Computing - An Introduction. Springer, Verlag (2002)
30. Păun, Gh., Rozenberg, G., Salomaa, A.: DNA Computing - New Computing Paradigms. Springer (1998)
31. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
32. Pixton, D.: Regularity of splicing languages. Discrete Applied Mathematics **69**, 101–124 (1996)
33. Song, B., Li, K., Orellana-Martín, D., Pérez-Jiménez, M.J., Hurtado, I.P.: A survey of nature-inspired computing: Membrane computing. ACM Computing Surveys **54**(1), 629–649 (2021). https://doi.org/https://doi.org/10.1145/3431234
34. Stamatopoulou, I., Kefalas, P., Gheorghe, M.: Modelling the dynamic structure of biological state-based systems. BioSystems **87**, 142–149 (2007)
35. Verlan, S.: On minimal context-free insertion-deletion systems. Journal of Automata, Languages and Combinatorics **12**, 317–328 (2007)