View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Hope's Institutional Research Archive

Language generating alphabetic flat splicing P systems

Linqiang Pan[a], Bosheng Song[b,*], Atulya K. Nagar[c], K.G. Subramanian[d]

[a] *School of Electric and Information Engineering, Zhengzhou University of Light Industry, Zhengzhou 450002, Henan, China*
[b] *Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China*
[c] *Department of Mathematics and Computer Science, Faculty of Science, Liverpool Hope University, Liverpool, L16 9JD, UK*
[d] *Department of Mathematics, Madras Christian College, Tambaram, Chennai 600059 India*

Abstract

An operation on strings, called flat splicing was introduced, inspired by a splicing operation on circular strings considered in the study of modelling of the recombinant behaviour of DNA molecules. A simple kind of flat splicing, called alphabetic flat splicing, allows insertion of a word with a specified start symbol and/or a specified end symbol, between two pre-determined symbols in a given word. In this work, we consider a P system with only alphabetic flat splicing rules as the evolution rules and strings of symbols as objects in its regions. We examine the language generative power of the resulting alphabetic flat splicing P systems (AFS P systems, for short). In particular, we show that AFS P systems with two membranes are more powerful in generative power than AFS P systems with a single membrane. We also construct AFS P systems with at most three membranes to generate languages that do not belong to certain other language classes and show an application to generation of chain code pictures.

Keywords: Membrane computing, P system, Flat splicing, Chomsky hierarchy

*Corresponding author
Email addresses: lqpan@mail.hust.edu.cn (Linqiang Pan), boshengsong@163.com (Bosheng Song), nagara@hope.ac.uk (Atulya K. Nagar), kgsmani1948@gmail.com (K.G. Subramanian)

1. Introduction

Ever since the new computability model based on membrane structure with a generic name of P system was introduced by Păun [25, 26], several theoretical as well as application-oriented P system models have been introduced with different motivations and investigated extensively, in the area of membrane computing [28, 36]. As a language generating mechanism, P system has been investigated initially in [24], introducing the notion of rewriting P systems and proving universality for rewriting P systems with three membranes that use a priority relation among the context-free rewriting rules. Such a system involves symbolic computation with structured strings as objects and the evolution rules could be rewriting rules as in formal language theory [29] or other kinds of string transformation rules as for example, contextual rules [23].

From the seminal definition of rewriting P systems by Păun [24], many other different kinds of rewriting P systems with several additional features and/or variants have been introduced. In [2], string rewriting P systems with different parallel methods of rewriting were considered and the computational power was analyzed. String rewriting P systems with communication between regions taking place by the use of query symbols were considered in [5] and computational completeness was established. Three variants of rewriting P systems were studied in [8] by considering leftmost rewriting, use of permitting and forbidding conditions.

Language generation power of rewriting P systems has also been investigated. In [9], rewriting P systems with membranes of variable thickness were considered and using such a P system with only four membranes, a characterization of recursively enumerable languages is obtained. In [14], a hybrid variety of language generating P systems that make use of both context-free and context-adjoining rules, is dealt with while in [15], the generative power of P systems with string-objects and contextual rules was investigated. In [21], a variant of rewriting P systems with global rules was considered besides improving the universality result in [24] by reducing the number of membranes to two while

in [22], P systems with partially parallel rewriting were considered and it was shown that such systems with six membranes generate recursively enumerable languages. Language generation using replicated rewriting P systems is considered in [16, 17]. In the recent past, language generation has been considered in the context of other kinds of P systems [13, 32, 33, 34, 35] such as spiking neural P systems and numerical P systems.

On the other hand, related to the operation of splicing introduced by Head [11, 12] in the area of DNA computing and extensively investigated [27], subsequently, Berstel et al. [1] introduced an operation on strings called *flat splicing*. This operation is done by "cutting" a string $x = u\alpha\beta v$ between α and β and inserting in x, a string $y = \gamma w \delta$ between α and β as directed by a flat splicing rule of the form $(\alpha|\gamma - \delta|\beta)$, thus yielding a word $u\alpha\gamma w\delta\beta v$. A flat splicing rule is called *alphabetic*, when $\alpha, \beta, \gamma, \delta$ are letters of an alphabet or the empty word.

In this work, we consider a variant of language generating P systems with strings as objects and a simple kind of evolution rules, namely, alphabetic flat splicing rules in the regions of the P systems and examine the language generative power of the resulting P systems. We show that alphabetic flat splicing P systems (AFS P systems, for short) with two membranes are more powerful than AFS P systems with a single membrane. We also construct AFS P systems with two membranes to generate languages not in the language classes generated by flat splicing systems with a finite number of initial words, pure context-free grammars and regular grammars. On the other hand, we show that an AFS P system with three membranes generates a context-sensitive language which is not context-free.

A *chain code picture* (also called *line picture*) is made of unit lines in the two-dimensional grid and is encoded by words over the alphabet $\{l, r, u, d\}$ with the symbols l, r, u, d respectively interpreted as instructions to draw a horizontal or vertical unit line to the left, right, up or down directions from the current position in the chain code picture. Chain code picture grammars generating chain code pictures are known [18] as early as 1982 while recently, chain code picture grammars were linked with P systems [3, 31]. Here as an application,

we construct an AFS P system to generate a language of chain code pictures describing "double stairs".

2. Preliminaries

For notions and results related to formal languages, we refer to [29] and to [25, 28] for P systems.

A *word* (also called a *string*) w is a finite sequence of symbols belonging to an alphabet V, which is a finite and nonempty set of symbols. We denote by V^*, the set of all words over V. The set V^* includes the empty word λ and we write $V^+ = V^* - \{\lambda\}$. The *length* $|w|$ of a word w is the number of symbols in w counting repetitions.

We denote by CSL, CFL, LIN, REG respectively, the families of languages generated by *context-sensitive*, *context-free*, *linear* or *regular* Chomsky grammars [29]. The family of finite languages is denoted by FIN.

We now recall the operation of *flat splicing* on words [1]. A flat splicing rule r is of the form $(\alpha|\gamma - \delta|\beta)$, where $\alpha, \beta, \gamma, \delta$ are words over an alphabet V. The words $\alpha, \beta, \gamma, \delta$ are called the *handles* of the rule. When all the four handles of the rule r are symbols in V or the empty word, then the flat splicing rule r is called *alphabetic*. For two words $x = u\alpha\beta v$, $y = \gamma w \delta$, $u, v, w \in V^*$, an application of the flat splicing rule $r = (\alpha|\gamma - \delta|\beta)$ to the pair (x, y) yields the word $z = u\alpha\gamma w\delta\beta v$ and we write $(x, y) \vdash_r z$. Informally expressed, an application of the rule r inserts the second word y between α and β in the first word x yielding the word z.

A *flat splicing system* (FSS) [1] is a triple $\mathcal{S} = (\Sigma, I, R)$, where Σ is an alphabet, I is an initial set of words over Σ and R is a finite set of flat splicing rules. The FSS $\mathcal{S}$ is respectively called *finite*, *regular* or *context-free* according as I is a finite set, a regular set or a context-free language. The language L generated by $\mathcal{S}$ is the smallest language containing I and such that for any two words $u, v \in L$ and any rule $r \in R$, if the rule r is applicable to the pair (u, v) and if the word w is obtained on applying the rule r to the pair (u, v), that is,

if $(u, v) \vdash_r w$, then w is also in L. An alphabetic flat splicing system ($AFSS$) has all the flat splicing rules alphabetic. The families of languages generated by FSS and $AFSS$ are respectively denoted by $\mathcal{L}(FSS, X)$ and $\mathcal{L}(AFSS, X)$ for $X = FIN, REG$ or CF according as the initial set is finite, regular or context-free.

Alphabetic flat splicing rules and context-free initial sets are known [1] to produce only context-free languages.

Theorem 2.1. *[1] The language generated by an alphabetic flat splicing system with context-free initial set is context-free.*

We also recall here the notion of a pure context-free grammar [19] which makes use of only one kind of symbols, namely terminal symbols and context-free kind of rules.

Definition 1. *[19] A pure context-free grammar is $G = (\Sigma, P, \Omega)$, where Σ is a finite alphabet, Ω is a set of axiom words and P is a finite set of context-free rules of the form $a \to \alpha, a \in \Sigma, \alpha \in \Sigma^*$. Derivations are done as in a context-free grammar except that unlike a context-free grammar, there is only one kind of symbol, namely the terminal symbol. The language generated consists of all words generated from each axiom word. The family of languages generated by pure context-free grammars is denoted by PCF.*

For example, the pure context-free grammar $G = (\{a, b, c, d, e\}, \{e \to caebd\}, \{caebd\})$ generates the language $\{(ca)^n e(bd)^n \mid n \geq 1\}$.

3. Flat splicing P systems

We now introduce language generating P systems with internal output computing languages of structured strings. The regions of P systems considered can have only alphabetic flat splicing rules.

Definition 2. *A language generating P system of degree $m \geq 1$, with alphabetic flat splicing rules, ($AFSP_m$) is*

$$\Pi = (V, \mu, F_1, \cdots, F_m, R_1, \cdots, R_m, i_o),$$

where

(i) V is an alphabet of the system;

(ii) μ is a membrane structure consisting of m membranes labelled in a one-to-one way with $1, \cdots, m$;

(iii) $F_1, \cdots, F_m$ are initial finite subsets of V^* associated with the m regions of μ;

(iv) $R_1, \cdots, R_m$ are finite sets of alphabetic flat splicing rules associated with the m regions of μ and the alphabetic flat splicing rules have attached targets here, out, in (in general, here is omitted);

(v) i_o is the label of the output membrane of μ.

A computation in an $AFSP_m$ is defined in a way similar to a string rewriting P system [24, 25]. A computation starts from an initial configuration defined by the membrane structure with the initial words, if any, in the m regions. The rules in a region are used in a nondeterministic maximally parallel manner, which means that the strings to evolve and the rules to be applied to them are chosen in a nondeterministic manner, but all strings in all the regions which can evolve at a given step should do it. On the other hand, the application of an alphabetic flat splicing rule to a pair of strings in a region, is sequential in the sense only one rule is applied to a pair of strings, resulting in an evolved string which is placed in the region indicated by the target associated with the rule used. As usual the target *here* means that the evolved string remains in the same region, *out* means that the evolved string exits the current membrane, (thus, if the rule was applied in the skin membrane, then it can exit the system such that strings leaving the system are "lost" in the environment), and *in* means that

the string is sent to one of the directly lower membranes, nondeterministically chosen if there exist several of them (thus, if no internal membrane exists, then a rule with the target indication *in* cannot be used). Note that in a region, if a flat splicing rule with target *in* or *out* is applied to a pair of strings (x, y), then x transforms into the string z, which is sent out of the region into an inner region or an outer region. Also note that a string passes through the membranes as a unique entity as in a rewriting P system [24].

A computation is successful only if it stops reaching a configuration where no rule can be applied to the existing strings. The result of a halting computation consists of strings in the output membrane in the halting configuration. The set of all such strings computed (also called generated) by the system Π is denoted by $L(\Pi)$.

The family of all languages $L(\Pi)$ generated by systems Π as above, with at most m membranes, is denoted by $\mathcal{L}(AFSP_m)$.

We give an example to illustrate $AFSP_2$ and its work.

Example 3.1. *Consider the* $AFSP_2$

$$\Pi_1 = (\{x, y, a, b, c, d\}, [_1\, [_2\,]_2\,]_1, \{axb, ca\}, \{bd, ybd\}, R_1, R_2, 2),$$

where R_1 *consists of the alphabetic flat splicing rule* r_1*, while* R_2 *consists of the alphabetic flat splicing rules* r_2*,* r_3*, where*

$$r_1 = (a|c - a|x), in; r_2 = (x|b - d|b), out; r_3 = (x|y - d|b).$$

Computation in Π_1 *takes place as follows: The region 1 has axiom strings* axb, ca *and the rule* r_1 *with target indication in, is applicable to the pair* (axb, ca)*, which transforms* axb *into* $acaxb$ *and sends it to region 2 while at the same time no string can evolve in region 2 although it has the strings* bd, ybd *initially, as no rule is applicable to any pair in this region. Once the string* $acaxb$ *arrives in region 2, if the rule* r_2 *with target indication out is applied to the pair* $(acaxb, bd)$ *then the string* $acaxb$ *is transformed into the string* $acaxbdb$ *and sent back to region 1. The process can be repeated and at any moment strings*

of the form $a(ca)^{p-1}x(bd)^{p-1}b$, $p \geq 2$ evolve in the region 2 and go to region 1. But when the string $a(ca)^{p}x(bd)^{p-1}b$, $p \geq 2$ arrives in region 2, if the rule r_3 (with target indication here) is applied to the pair $(a(ca)^{p}x(bd)^{p-1}b, ybd)$ then this yields the string $a(ca)^{p}xy(bd)^{p}b$, $p \geq 1$ which remains in region 2 itself. The computation reaches a halting configuration and the string $a(ca)^{p}xy(bd)^{p}b$ is collected in the output region 2, thus generating the context-free language $\{bd, ybd\} \cup \{a(ca)^{p}xy(bd)^{p}b \mid p \geq 1\}$. Note that bd, ybd are axiom strings in region 2, which are also collected in the language.

We now examine the generative power of the alphabetic flat splicing P system.

Theorem 3.1. $\mathcal{L}(AFSP_1) \subset \mathcal{L}(AFSP_2)$.

Proof. The inclusion is by definition. In order to prove the proper inclusion, consider the language $L_1 = \{c\} \cup \{x(cad)^{n}y \mid n \geq 0\}$ over the alphabet $\{a, c, d, x, y\}$. The language L_1 is generated by the $AFSP_2$

$$\Pi_2 = (\{a, c, d, x, y\}, [_1\, [_2\,]_2\,]_1, \{xy, ad\}, \{c, xy\}, R_1, R_2, 2),$$

where R_1 consists of the alphabetic flat splicing rule r_1, r_2, while R_2 consists of the alphabetic flat splicing rules r_3, r_4, where

$$r_1 = (x|a - d|y), in; r_2 = (x|a - d|c), in; r_3 = (x|c - \lambda|a), out; r_4 = (x|\lambda - c|a).$$

In a computation of this system, initially, in region 1, the string xy transforms into $xady$ by the application of the rule r_1 which "inserts" ad into xy. Due to the target indication in of the rule r_1, the string $xady$ is sent to region 2 where it is transformed into $xcady$ by the application of the rule r_3. Due to the target indication out of the rule r_3, the string $xcady$ is sent back to region 1. The process can be continued by repeated application of the rule r_2 in region 1 and r_3 in region 2, sending the generated string back and forth between regions 1 and 2. At some stage when the string of the form $xad(cad)^{n-1}y$, $n \geq 1$ arrives in region 2, an application of the rule r_4 generates a string of the form $x(cad)^{n}$, $n \geq 1$ which remains in region 2 and the computation halts as no rule

is applicable in any of the regions. Thus the language generated is L_1. But L_1 cannot be generated by any $AFSP_1$. If we assume the contrary, we note that the only membrane in $AFSP_1$ can have only a finite number of initial words (which are also in the language L_1) and a finite number of alphabetic flat splicing rules. Every word (other than c) in the language L_1 has only one x and one y. It can be seen that this property cannot be maintained in the generated words and hence $L_1 \notin \mathcal{L}(AFSP_1)$. □

Theorem 3.2. *$\mathcal{L}(AFSP_2) - \mathcal{L}(FSS, FIN) \neq \emptyset$. As a consequence $\mathcal{L}(AFSP_2) - \mathcal{L}(AFSS, FIN) \neq \emptyset$.*

Proof. In the proof of Theorem 3.1, the language $L_2 = \{c\} \cup \{x(cad)^n y \mid n \geq 0\}$ over the alphabet $\{a, c, d, x, y\}$ was shown to be in $\mathcal{L}(AFSP_2)$. But L_2 cannot be in the family $\mathcal{L}(FSS, FIN)$. In fact if L_2 can be generated by a flat splicing system with a finite number of axioms, then the axioms which will also belong to L_2, should also be words with only one x in the beginning and one y at the end, the only exception being the word c. But then the flat splicing rules of the system will have to allow these axiom words to be inserted into other axiom words and the process of such an insertion will have to continue with the resulting words. This means that the property of having only one x in the beginning and one y at the end, of the words in L_2, cannot be maintained, as the generated words will have more than one x and more than one y. Note that insertion of the axiom word c into other words is not enough to generate the words of the form $x(cad)^n y$. Also, an alphabetic flat splicing system is a special kind of FSS and hence $L_2 \notin \mathcal{L}(AFSS, FIN)$. □

Theorem 3.3. *(i)* $\mathcal{L}(AFSP_2) - REG \neq \emptyset$.
(ii) $\mathcal{L}(AFSP_2) - PCF \neq \emptyset$.

Proof. Consider the language $L_3 = \{ca, cad\} \cup \{(ca)^n (db)^n \mid n \geq 1\}$. Clearly this language is not regular [29]. It cannot also be generated by any

pure context-free grammar [19] since no context-free kind of rule for any of the letters a, b, c, d can generate an equal number of $(ca)'s$ and $(db)'s$. But L_3 can be generated by the $AFSP_2$ given by

$$(\{a, b, c, d\}, [_1\, [_2\,]_2\,]_1, \{cacabdb, db\}, \{ca, cad, cadb, cacadbdb\}, R_1, R_2, 2),$$

where R_1 consists of the alphabetic flat splicing rule $r_1 = (b|d-b|d), in$; while R_2 consists of the rules $r_2 = (a|c-a|b), out$ and $r_3 = (a|c-d|b)$. The computation takes place as follows: While no rule is applicable to any pair of words in region 2 initially, at the same time in region 1, application of rule r_1 inserts db between b and d in the word $cacabdb$ generating the string $cacabdbdb$ which is then sent to region 2. Note at this moment only the word db remains in region 1 and so no rule is applicable. Now in region 2, if the rule r_2 is applied to the pair of words $(cacabdbdb, ca)$ then $cacacabdbdb$ is generated which is then sent back to region 1. The process can be repeated. When a word of the form $(ca)^{n-1}b(db)^{n-1}$, $n \geq 3$ reaches region 2, if the rule r_3 is applied on the pair $((ca)^{n-1}b(db)^{n-1}, cad)$, then the string generated is $(ca)^n(db)^n$, which remains in region 2 with a halting computation. □

Theorem 3.4. $\mathcal{L}(AFSP_3) - CF \neq \emptyset$.

Proof. Consider the context-sensitive language $L_4 = \{d, ed\} \cup \{a^n b^n e(dc)^{n-1} \mid n \geq 2\}$, which is not context-free. The language L_4 is generated by the following $AFSP_3$

$$(\{a, b, c, d, e\}, [_1\, [_2\, [_3\,]_3\,]_2\,]_1, \{a, abc\}, \{b, c\}, \{d, ed\}, R_1, R_2, R_3, 3),$$

where R_1 consists of the alphabetic flat splicing rule $r_1 = (a|\lambda - a|b), in$, while R_2 consists of the rules $r_2 = (b|b - \lambda|c), in$ and $r_4 = (b|c - \lambda|d), out$ and R_3 consists of the rules $r_3 = (b|d - \lambda|c), out$ and $r_5 = (b|e - d|c)$. The computation takes place as follows: Initially, application of the rule r_1 in region 1 to the pair (abc, a) inserts a into abc between a and b and the resulting word $aabc$ enters region 2. Here the application of the rule r_2 to the pair $(aabc, b)$ inserts b

between b and c and the resulting word $aabbc$ is sent to region 3. If rule r_3 in this region is applied to the pair $(aabbc, d)$, then the resulting word $aabbdc$ is sent out to region 2, the only rule applicable is r_4 to the pair $(aabbdc, c)$ yielding the word $aabbcdc$ which is sent to region 1. The process can repeat. Note that by the construction of the P system, when a rule is applicable to a pair of words in a region, no other rule in any other region becomes applicable to any available pair of words. In general when a word of the form $a^n b^n c(dc)^{n-2}$, $n \geq 2$ reaches region 3 and if the rule r_5 is applied to the pair $(a^n b^n c(dc)^{n-2}, ed)$, then the computation halts and the generated word $a^n b^n e(dc)^{n-1}$ remains in the output membrane 3, which is included in the language of the system. □

4. Application to chain-code pictures

There are many applications of the P system models in the area of membrane computing [4], in particular in the generation of digitized pictures in the two-dimensional plane [30]. Description of pictures given by chain codes has been of great interest and investigation [6, 7, 10, 18], in view of the fact that the chain-code pictures in the two-dimensional plane are described by string grammars. Recently, P systems for chain code picture languages have been proposed and investigated [3, 31]. Here we illustrate, by an example, an application of flat splicing P systems in generating chain code picture languages.

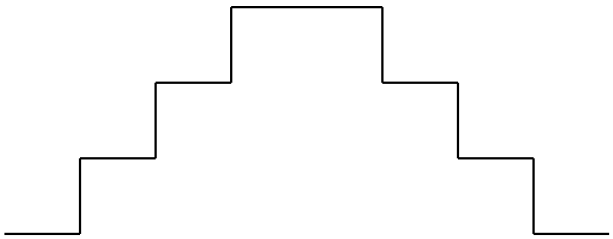

Figure 1: A chain code picture of double "stairs" of equal height

A chain code picture [18] p is composed of unit horizontal and vertical lines in the two-dimensional plane and is described by words over the alphabet $\{l, r, u, d\}$

with the symbols l, r, u, d respectively interpreted as instructions to draw a horizontal or vertical unit line to the left, right, up or down directions from the present position in the chain code picture. A chain code picture language is a set of chain code pictures.

Consider the non-regular language $L_c = \{(ru)^n rr(dr)^n l \mid n \geq 1\}$, whose words correspond to chain code pictures of double "stairs" of equal height. For $n = 3$, the chain code picture of the word $(ru)^3 rr(dr)^3 l$ is shown in Fig. 1. Here we give a flat splicing P system Π_c with two membranes to generate the chain code picture language L_c. The P system Π_c is given by

$$\Pi_c = \{l, r, u, d\}, [_1\, [_2\,]_2\,]_1, \{ru, rr, rudrl\}, \{dr\}, R_1, R_2, 2),$$

where R_1 consists of the alphabetic flat splicing rules $r_1 = (u|r - u|d), in$ and $r_3 = (u|r - r|d), in$, while R_2 consists of the rules $r_2 = (u|d - r|d), out$. The computation takes place as follows: Initially, application of the rule r_1 in region 1 to the pair $(rudrl, ru)$ inserts ru into $rudrl$ between u and d and the resulting word $rurudrl$ enters region 2. Here the application of the rule r_2 to the pair $(rurudrl, dr)$ inserts dr between u and d and the resulting word $rurudrdrl$ is sent back to region 1. The process can repeat. But in region 1, if the rule r_3 is applied to the pair $(rurudrdrl, rr)$, then the resulting word $rururrdrdrl$ is sent into the region 2 and is collected in the language.

5. Conclusions and discussions

In this work, we have considered P systems of the language generating variety making use of a simple kind of rule, namely, alphabetic flat splicing rules. We have shown that alphabetic flat splicing P systems with two membranes are more powerful in generative power than alphabetic flat splicing P systems with a single membrane. We also proved that alphabetic flat splicing P systems with at most three membranes generate languages that do not belong to certain other language classes. Finally, an application of alphabetic flat splicing P systems to generation of chain code pictures was given.

One of the problems that can be explored is whether the number of membranes in the stated results, especially in Theorem 3.4, could be reduced. Although there is a strictly context-sensitive language (as shown in Theorem 3.4), which can be generated by an AFS P system with three membranes, the position in the Chomsky hierarchy of the language family of AFS P systems with three membranes at the most, remains to be investigated. It will also be of interest to examine whether the inclusion $AFSP_m \subseteq AFSP_{m+1}$ is proper or not, for $m \geq 2$.

The language generative power of alphabetic flat splicing P systems in comparison with Chomsky hierarchy has been investigated in section 3. It remains open to explore whether alphabetic flat splicing P systems can generate any recursively enumerable language. Moreover, in section 4, a simple picture, double "stairs" of equal height has been generated. It will be interesting to examine whether we can generate more complex pictures (such as chess board patterns) by using alphabetic flat splicing P systems.

The main feature of an alphabetic flat splicing rule is to allow only insertion of a word into another word in the context of certain symbols, which makes it very restrictive. So it may be of interest to study the language generative power of a hybrid variety of language generating P system having certain simple kinds of rules such as the deletion rules of the form $a \to \lambda$ in addition to alphabetic flat splicing rules. Along this direction, a language generating P system of the hybrid variety with only regular or linear Chomsky type of rules along with alphabetic flat splicing rules, is considered in [20] and such a P system with two membranes generates a context-sensitive language.

Acknowledgements

The work of L. Pan and B. Song was supported by National Natural Science Foundation of China (61033003, 61320106005, 61472372, 61472371, 61572446, and 61602192), China Postdoctoral Science Foundation (2016M600592), and the Innovation Scientists and Technicians Troop Construction Projects of Henan

Province (154200510012). The fourth author K.G. Subramanian is grateful to UGC, India, for the award of Emeritus Fellowship (No.F.6-6/2016-17/EMERITUS-2015-17-GEN-5933/(SA-II)) to him to execute his work in the Department of Mathematics, Madras Christian College.

References

References

[1] J. Berstel, L. Boasson, I. Fagnot, Splicing systems and the chomsky hierarchy, Theor. Comput. Sci. 436 (2012) 2–22.

[2] D. Besozzi, C. Ferretti, G. Mauri, C. Zandron, Parallel rewriting P systems with deadlock, in: Proc. of the 8th International Workshop on DNA-Based Computers, in: LNCS, vol. 2568, 2003, pp. 302–314.

[3] R. Ceterchi, K.G. Subramanian, I. Venkat, P systems with parallel rewriting for chain code picture languages, in: Proc. of the 11th Conference on Computability in Europe, in: LNCS, vol. 9136, 2015, pp. 145–155.

[4] G. Ciobanu, M.J. Pérez-Jiménez, Gh. Păun, Applications of Membrane Computing, Springer, Berlin, 2006.

[5] E. Csuhaj-Varjú, G. Vasil, P systems with string objects and with communication by request, in: Proc. of the 8th Workshop on Membrane Computing, in: LNCS, vol. 4860, 2007, pp. 228–239.

[6] J. Dassow, Grammatical Picture generation, Tech. Report, Otto-von-Guericke-Universität Magdeburg, Chapter 4, 101–121, 2011.

[7] J. Dassow, F. Hinz, Decision problems and regular chain code picture languages, Discrete Appl. Math. 45 (1993) 29–49.

[8] C. Ferretti, G. Mauri, Gh. Paun, C. Zandron, On three variants of rewriting P systems, Theor. Comput. Sci. 301 (2003) 201–215.

[9] R. Freund, C. Martin-Vide, Gh. Păun, From regulated rewriting to computing with membranes: collapsing hierarchies, Theor. Comput. Sci. 312 (2004) 143–188.

[10] R. Gutbrod, A transformation system for generating description languages of chain code pictures, Theor. Comput. Sci. 68 (1989) 239–252.

[11] T. Head, Formal language theory and dna: an analysis of the generative capacity of specific recombinant behaviours, B. Math. Biol. 49 (1987) 735–759.

[12] T. Head, Circular Suggestions for DNA Computing. In: Pattern Formation in Biology, Vision and Dynamics(Ed. Carbone, A. et al.), World Scientific, (2000) 325–335.

[13] K. Jiang, W. Chen, Y. Zhang, L. Pan, On string languages generated by sequential spiking neural P systems based on the number of spikes, Nat. Comput. 15 (1) (2016) 87–96.

[14] S.N. Krishna, K. Lakshmanan, R. Rama, Hybrid P systems, Rom. J. Inf. Sci. Tech. 4 (2001) 11–123.

[15] S.N. Krishna, K. Lakshmanan, R. Rama, On the power of P systems with contextual rules, Fundam. Inform. 49 (2002) 167–178.

[16] S.N. Krishna, R. Rama, P Systems with Replicated Rewriting. J. Automata, Languages and Combinatorics. 6 (3) (2001) 345–350.

[17] V. Manca, C. Martin-Vide, Gh. Păun, On the power of P systems with replicated rewriting, J. Automata, Languages and Combinatorics, 6 (3) (2001) 359–374.

[18] H.A. Maurer, G. Rozenberg, E. Welzl, Using string languages to describe picture languages, Inf. Contro. 54 (3) (1982) 155–185.

[19] H.A. Maurer, A. Salomaa, D. Wood, Pure Grammars, Inf. Control. 44 (1980) 47–72.

[20] L. Pan, B. Song, A.K. Nagar, K.G. Subramanian, Rewriting P systems with flat-splicing rules, in: Proc. of the 17th Int. Conf. Membrane Computing, (2016) 249–261.

[21] A. Păun, P systems with global rules, Theor. Comput. Syst. 35 (5) (2002) 471–481.

[22] A. Păun, On P systems with partial parallel rewriting, Rom. J. Inf. Sci. Tech. 4 (2001) 203–210.

[23] Gh. Păun, Marcus Contextual Grammar. Studies in Linguistics and Philosophy. Kluwer, Dordrecht, 1997.

[24] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (2000) 108–143.

[25] Gh. Păun, Computing with Membranes: An Introduction. Springer-Verlag, Berlin, 2002.

[26] Gh. Păun, From Cells to Computers: Membrane Computing - A Quick Overview, in: Proc. of the 10th International Workshop on DNA Computing, in: LNCS, vol. 3384, 2005, pp. 268–280.

[27] Gh. Păun, G. Rozenberg, A. Salomaa, DNA Computing–New Computing Paradigms. Texts in Theoretical Computer Science. An EATCS Series, Springer, 1998.

[28] Gh. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing. Oxford University Press, New York, 2010.

[29] G. Rozenberg, A. Salomaa, (Eds.): Handbook of Formal Languages. Vols. 1-3, Springer-Verlag, Berlin, 1997.

[30] K.G. Subramanian, P systems and picture languages, in: Proc. of the 5th International Conference on Machines, Computations and Universality, in: LNCS, vol. 4644, 2007, pp. 99–109.

[31] K.G. Subramanian, I. Venkat, L. Pan, P systems generating chain code picture languages, in: Proc. of Asian Conference on Membrane Computing, Wuhan, (2012) pp. 115–123.

[32] T. Wu, Z. Zhang, L. Pan, On languages generated by cell-like spiking neural P systems, IEEE T. Nanobiosci. 15 (5) (2016) 455–467.

[33] X. Zeng, L. Xu, X. Liu, L. Pan, On languages generated by spiking neural P systems with weights, Inf. Sci. 278 (2014) 423–433.

[34] X. Zhang, X. Zeng, L. Pan, On languages generated by asynchronous spiking neural P systems, Theor. Comput. Sci. 410 (26) (2009) 2478–2488.

[35] X. Zhang, Y. Liu, B. Luo, L. Pan, Computational power of tissue P systems for generating control languages, Inf. Sci. 278 (2014) 285–297.

[36] G. Zhang, M. Gheorghe, L. Pan, M.J. Pérez-Jiménez, Evolutionary membrane computing: a comprehensive survey and new results, Inf. Sci. 279 (2014) 528–551.