ELSEVIER

Theoretical Computer Science 262 (2001) 117–131

Theoretical Computer Science

www.elsevier.com/locate/tcs

Formal properties of PA-matching ☆

Satoshi Kobayashi[a], Victor Mitrana[b], Gheorghe Păun[c], Grzegorz Rozenberg[d]

[a] *Department of Information Sciences, Tokyo Denki University, Ishizaka, Hatoyama-machi, Hiki-gun, Saitama 350-0394, Japan*

[b] *Faculty of Mathematics, Bucharest University, Str. Academiei 14, 70109 Bucureşti, Romania*

[c] *Institute of Mathematics of the Romanian Academy, P.O. Box 1 – 764, 70700 Bucureşti, Romania*

[d] *Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands*

Received February 2000; accepted March 2000

Abstract

We consider the PA-matching operation, used in DNA computing, as a formal operation on strings and languages. We investigate the closure of various families of languages under this operation, representations of recursively enumerable languages and decision problems. We also consider the dual operation of overlapping strings. All closure properties of families in the Chomksy hierarchy under both non-iterated and iterated PA-matching and overlapping operations are settled.

Keywords: DNA computing; PA-matching operation; Overlapping operation; Chomsky hierarchy

1. Introduction

In the fast emerging area of DNA computing, many new computability models are considered, where many of the operations used are inspired by the DNA behavior in vivo or in vitro. Examples of such operations are the splicing operation (used in H systems), the annealing (used in sticker systems), and the insertion–deletion operations. These and other operations are discussed in [7].

☆ This work was supported by the Leiden Center for Natural Computing and the Lorentz Centre of Leiden University. The first author was supported in part by "Research for the Future" Program No. JSPS-RFTF 96I00101 from the Japan Society for the Promotion of Science and Grants-in-Aid for Scientific Research No. 09878059 from the Ministry of Education, Science and Culture, Japan.

E-mail addresses: satoshi@j.dendai.ac.jp (S. Kobayashi), mitrana@funinf.math.unibuc.ro (V. Mitrana), gpaun@imar.ro (G. Păun), rozenber@wi.leidenuniv.nl (G. Rozenberg).

0304-3975/01/$ - see front matter
PII: S0304-3975(00)00185-7

Here, we investigate yet another operation suggested by operations on DNA molecules, the so-called PA-matching operation, used in [10]. It is related to both the splicing and the annealing operations: starting from two single-stranded molecules x, y, such that a suffix w of x is complementary to a prefix $\bar{w}$ of y, by annealing we can form the molecule with the double-stranded part $\binom{w}{\bar{w}}$ and the remaining sticky ends specified by x and y. The matching part is then ignored (removed), so that the resulting string consists of the prefix of x and the suffix of y which were not matched.

This operation is considered here as an abstract operation on formal languages. We relate it to other operations in formal language theory and we settle the closure properties of families in the Chomsky hierarchy under it. A dual operation is that of overlapping, where we keep a matching part of two strings. Also in this case we settle all closure properties of Chomsky families.

Once again, it turns out that manipulation of DNA molecules leads to operations interesting from formal language theory point of view.

2. Formal language theory prerequisites

In this section we recall/introduce some basic notions and notations necessary for the rest of the paper. For details we refer to [11].

For an alphabet V, we denote by V^* the set of all strings of symbols in V, including the empty string λ. The length of a string $x \in V^*$ is denoted by $|x|$, while $V^* - \{\lambda\}$ is denoted by V^+.

The *mirror image* of a string $x = a_1 a_2 \dots a_k,\ a_i \in V,\ 1 \leqslant i \leqslant k$, is $x^R = a_k \dots a_2 a_1$. The *shuffle* of two strings $x, y \in V^*$ is defined by

$$x \sqcup\!\sqcup y = \{x_1 y_1 x_2 y_2 \dots x_n y_n \mid x = x_1 x_2 \dots x_n,\ y = y_1 y_2 \dots y_n,$$
$$\text{for some } n \geqslant 1,\ x_i, y_i \in V^*,\ 1 \leqslant i \leqslant n\}.$$

Both these operations are extended from strings to languages in the usual way.

The sets of prefixes, suffixes, proper prefixes, and proper suffixes of a language $L \subseteq V^*$ are denoted by $Pref(L)$, $Suf(L)$, $PPref(L)$, $PSuf(L)$, respectively.

For $L_1, L_2 \subseteq V^*$ we define the *left quotient* of L_1 with respect to L_2 by $L_2 \backslash L_1 = \{w \in V^* \mid xw \in L_1 \text{ for some } x \in L_2\}$. The *right quotient* is defined in the symmetric way. When L_2 is a singleton, $L_2 = \{x\}$, then we write $\partial_x^{\ell}(L)$ instead of $\{x\} \backslash L_1$ and this operation is called the *left derivative* of L_1 with respect to x. The *right derivative* is denoted by $\partial_x^{\mathrm{r}}(L_1)$.

A *finite transducer* is a 6-tuple $M = (Q, V_{\mathrm{i}}, V_{\mathrm{o}}, q_0, F, \delta)$ where $Q, V_{\mathrm{i}}, V_{\mathrm{o}}$ are finite and non-empty alphabets (the set of states, the input alphabet, and the output alphabet, respectively), $q_0 \in Q$ (the initial state), $F \subseteq Q$ (the set of final states), and δ is the (transition-and-output) function from $Q \times (V_{\mathrm{i}} \cup \{\lambda\})$ to finite subsets of $Q \times V_{\mathrm{o}}^*$. This function is extended in a natural way to $Q \times V_{\mathrm{i}}^*$. Each finite transducer M as above

defines a *finite transduction*

$$M(\alpha) = \{\beta \in V_o^* \mid (q, \beta) \in \delta(q_0, \alpha),\ q \in F\}.$$

The finite transduction M is extended to languages $L \subseteq V_i^*$ in the usual way.

If we ignore V_o and the output, then we obtain a *finite automaton* (with λ moves). A language is *regular* iff it is accepted by a finite automaton. If L is a regular language and M is a finite transducer, then $M(L)$ is also regular.

A *context-free grammar* is a construct $G = (N, T, S, P)$, where N is the non-terminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom, and P is the set of production rules. The rules are written in the form $A \to z$, where $A \in N$ and $z \in (N \cup T)^*$. If for all rules $A \to z \in P$ the string z contains at most one nonterminal, then the grammar is said to be *linear*. If all rules are of the form $A \to aB$, $A \to a$, for $A, B \in N$, $a \in T$, then the grammar is said to be *regular*. For $x, y \in (N \cup T)^*$, we write $x \Rightarrow y$ if and only if $x = x_1 A x_2$, $y = x_1 z x_2$, for some $x_1, x_2 \in (N \cup T)^*$ and $A \to z \in P$. The reflexive and transitive closure of the relation $\Rightarrow$ is denoted by $\Rightarrow^*$. The language generated by G is $L(G) = \{x \in T^* \mid S \Rightarrow^* x\}$.

By *REG*, *LIN*, *CF*, *CS*, *RE* we denote the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively.

Let V be an alphabet; an *instance of the Post Correspondence Problem*, denoted $PCP(x, y)$, is an ordered pair (x, y), $x = (x_1, \ldots, x_k)$, $y = (y_1, \ldots, y_k)$, of nonempty strings over V. $PCP(x, y)$ has a solution if there are $i_1, i_2, \ldots, i_n$, for $n \geqslant 1$, $1 \leqslant i_j \leqslant k$, $1 \leqslant j \leqslant n$, such that $x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n}$. It is known that the Post Correspondence Problem is undecidable, i.e., there is no algorithm which can decide whether or not an arbitrary instance has a solution.

3. The PA-matching operation

3.1. The non-iterated case

We introduce a new operation on strings, inspired by the operation used in [10]. This operation, called the PA-matching, belongs to "cut-and-paste" operations much investigated as basic operations for theoretical models of DNA computing (see details in [7]). Informally speaking, our operation consists of cutting two strings in two segments such that the prefix of one of them matches the suffix of another, removing these two matching pieces, and pasting the remaining parts.

Formally, given an alphabet V, a subset X of V^+, and two strings $u, v \in V^+$, one defines

$$PAm_X(u, v) = \{wz \mid u = wx,\ v = xz,\ \text{for } x \in X,\ \text{and } w, z \in V^*\}.$$

The operation is naturally extended to languages over V by

$$PAm_X(L_1, L_2) = \bigcup_{u \in L_1, v \in L_2} PAm_X(u, v).$$

When $L_1 = L_2 = L$ we write $PAm_X(L)$ instead of $PAm_X(L_1, L_2)$. Since we shall only deal either with finite sets X or with $X = V^+$, we use the notation $fPAm$ for finite PA-matching and the notation PAm for arbitrary PA-matching PAm_{V^+}.

The reader familiar with the splicing operation ([4, 5, 7]) may easily recognize a special variant of splicing in the finite PA-matching case.

A *splicing rule* over V is a quadruple $r = (u_1, u_2, u_3, u_4)$, with $u_i \in V^*$, $1 \leqslant i \leqslant 4$.

Given a finite set R of splicing rules and the strings $x, y \in V^*$, we write

$$\sigma_R(x, y) = \{x_1u_1u_4y_2 \mid x = x_1u_1u_2x_2, \quad y = y_1u_3u_4y_2, \\ (u_1, u_2, u_3, u_4) \in R,\ x_1, x_2, y_1, y_2 \in V^*\}.$$

For $L_1, L_2, L \subseteq V^*$, we define

$$\sigma_R(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} \sigma_R(x, y)$$

$$\sigma_R(L) = \sigma_R(L, L),$$

$$\sigma_R^0(L) = L,$$

$$\sigma_R^{i+1}(L) = \sigma_R^i(L) \cup \sigma_R(\sigma_R^i(L)), \quad i \geqslant 0,$$

$$\sigma_R^*(L) = \bigcup_{i \geqslant 0} \sigma_R^i(L).$$

Note that in the splicing case we cannot check the suffix–prefix matching; this is the main difference between the two operations. However, with the use of other operations, the two operations can simulate each other.

Lemma 1. *If a family F of languages is closed under concatenation with symbols and non-iterated splicing, then F is closed under the operation $fPAm$.*

Proof. For $L_1, L_2 \subseteq V^*$, consider two symbols c_1, c_2 not in V. For a finite set $X \subseteq V^+$, consider the set of splicing rules $R = \{(\lambda, xc_2, c_1x, \lambda) \mid x \in X\}$. Then we obviously have

$$PAm_X(L_1, L_2) = \sigma_R(L_1\{c_2\}, \{c_1\}L_2),$$

which implies the lemma. □

Lemma 2. *If a family F of languages is closed under finite transductions and the operation $fPAm$, then it is closed under non-iterated splicing.*

Proof. Let $L_1, L_2 \subseteq V^*$ be two languages and R be a finite set of splicing rules over V. For each rule $r = (u_1, u_2, u_3, u_4)$ consider a new symbol a_r and let $X = \{a_r \mid r \in R\}$ be the set of these symbols.

We define two finite transducers, M_1, M_2, such that, for each $x \in V^*$,

$$M_1(x) = \{x_1 u_1 a_r \mid x = x_1 u_1 u_2 x_2, \text{ for } r = (u_1, u_2, u_3, u_4) \in R \text{ and } x_1, x_2 \in V^*\},$$
$$M_2(x) = \{a_r u_4 x_2 \mid x = x_1 u_3 u_4 x_2, \text{ for } r = (u_1, u_2, u_3, u_4) \in R \text{ and } x_1, x_2 \in V^*\}.$$

Clearly, the equality

$$\sigma_R(L_1, L_2) = PAm_X(M_1(L_1), M_2(L_2))$$

holds (the PA-matching just puts together the strings marked by the two tranducers) which proves the lemma. □

Of course, the concatenation with symbols can also be performed by finite transducers, therefore, by combining the above two lemmas we get:

Theorem 1. *If F is a family of languages closed under finite transductions, then F is closed under the operation $fPAm$ if and only if it is closed under non-iterated splicing.*

Then by Theorem 7.1 from [7], we get the following corollary:

Corollary 1. *The families REG, CF, RE are closed under the $fPAm$ operation, but LIN is not closed.*

Also the family CS is closed under the operation $fPAm$ (although it is not closed under splicing), as a consequence of the following result.

Lemma 3. *If a family F of languages is closed under concatenation, union, and right and left derivatives, then F is closed under the operation $fPAm$.*

Proof. The following equality is obvious:

$$PAm_X(L_1, L_2) = \bigcup_{x \in X} \partial_x^r(L_1) \partial_x^\ell(L_2).$$

The required closure properties of F imply then the lemma. □

Corollary 2. *The family CS is closed under the $fPAm$ operation.*

We move now to investigate the properties of arbitrary PA-match operation.

Lemma 4. *If a family F of languages is closed under the shuffle and finite transductions, then F is closed under PAm.*

Proof. Let $L_1, L_2 \in F, L_1, L_2 \subseteq V^*$. Consider the alphabet $V' = \{a' \mid a \in V\}$ and the morphism h defined by $h(a) = a'$, for $a \in V$. Since each morphism can be realized by a finite transducer, $h(L_1) \in F$.

We construct now a finite transducer M which, informally speaking, works as follows on the strings from the language $L_1 \perp\!\!\!\perp h(L_2)$:

- M reads a prefix of the input string formed exclusively by non-primed letters and leaves it unchanged;
- then, starting from a new state, M checks for a while if the input contains only pairs of letters of the form aa', and writes nothing to the output;
- then, starting from another state, M reads only primed symbols and writes as output the non-primed versions of them.

It is easy to see that M defines a transduction that satisfies the equation

$$M(L_1 \perp\!\!\!\perp h(L_2)) = PAm(L_1, L_2).$$

Thus the lemma holds. □

Lemma 5. *If a family F of languages such that $REG \subseteq F$ is closed under concatenation with symbols, left derivatives, and PAm, then F is closed under Pref.*

Proof. Let $L \subseteq V^*$ and let c_1, c_2 be two new symbols. Then obviously

$$Pref(L) = \partial^{\ell}_{c_1} PAm(\{c_1\}L\{c_2\}, V^*\{c_2\}),$$

and so the lemma holds. □

Theorem 2. 1. *The families REG and RE are closed under PAm*
2. *The families LIN, CF, and CS are not closed under PAm.*

Proof. Let us consider the languages

$$L_1 = \{c_1 w d_1 w^R d_2 \mid w \in \{a,b\}^+\},$$
$$L_2 = \{d_1 w d_2 w^R c_2 \mid w \in \{a,b\}^+\}.$$

Clearly, both of them are linear languages. It is easy to see that

$$\partial^{\ell}_{c_1}(\partial^{r}_{c_2}(PAm(L_1, L_2))) = \{ww \mid w \in \{a,b\}^+\},$$

which is not a context-free language. Consequently, the families *LIN* and *CF* are not closed under *PAm*.

The family *CS* is not closed under *Pref*; the families *REG*, *RE* are closed under shuffle and finite transductions. Thus, the theorem follows from the previous lemmas. □

A language L is said to be a *fixed point* of the PA-match operation iff $PAm(L) = L$.

If L is a regular language, then by Theorem 2 we have that $PAm(L)$ is regular. The equivalence problem for regular languages is decidable. Therefore, we can decide

whether or not a given regular language is a fixed point of the PA-match operation. As expected, this is not true for the family of context-free languages.

Theorem 3. *The problem whether or not a given context-free language is a fixed point of the PA-match operation is undecidable.*

Proof. Take two arbitrary n-tuples of nonempty strings over the alphabet $\{a,b\}$, $x=(x_1, x_2,\ldots,x_n)$, $y=(y_1, y_2,\ldots, y_n)$, $n\geqslant 1$, and consider the languages

$$L_z = \{ba^{t_1}ba^{t_2}\ldots ba^{t_k}cz_{t_k}\ldots z_{t_2}z_{t_1} \mid k\geqslant 1, 1\leqslant t_i\leqslant n, 1\leqslant i\leqslant k\}, \text{ for } z\in\{x,y\},$$

$$L_s = \{w_1cw_2cw_2^Rcw_1^R \mid w_1,w_2\in\{a,b\}^*\},$$

$$L(x,y) = \{a,b,c\}^* - (L_x\{c\}L_y^R \cap L_s).$$

It is known, see, e.g., [12], that $L(x,y)$ is a context-free language. If $PCP(x,y)$ has no solution, then $L(x,y)=\{a,b,c\}^*$ and $PAm(L(x,y))=\{a,b,c\}^*$. If $PCP(x,y)$ has solutions, then $L(x,y)\neq\{a,b,c\}^*$ but still $PAm(L(x,y))=\{a,b,c\}^*$. (For each $w\in\{a,b,c\}^*$, the strings c^4 and c^4w are in $\{a,b,c\}^*$ but not in L_s; hence, these strings are in $L(x,y)$. This means that $w\in PAm(c^4,c^4w)$, that is, $\{a,b,c\}^*\subseteq PAm(L(x,y))$. The converse inclusion is trivial.)

Consequently, $PAm(L(x,y))=L(x,y)$ if and only if $PCP(x,y)$ has no solution. Since PCP is undecidable, the theorem holds. □

3.2. The iterated case

We will investigate now the iterated version of the PA-match operation.

It is defined as follows. For a language $L\subseteq V^*$ and a finite set $X\subseteq V^+$, we define:

$$PAm_X^0(L) = L,$$

$$PAm_X^{k+1}(L) = PAm_X^k(L)\cup PAm_X(PAm_X^k(L)), \quad k\geqslant 0,$$

$$PAm_X^*(L) = \bigcup_{k\geqslant 0} PAm_X^k(L).$$

When X is finite, the iterated PA-matching operation is denoted by $fPAm^*$; in the case $X=V^*$, the corresponding operation is denoted by PAm^*.

Lemma 6. *If a family F of languages is closed under concatenation with symbols, iterated splicing, and left and right derivatives, then F is closed under iterated finite PA-matching.*

Proof. Let $L\subseteq V^*$ be a language in F and X be a finite subset of V^+. Let c_1, c_2 be two new symbols. We associate with X the set of splicing rules $R=\{(\lambda, xc_2, c_1x, \lambda) \mid x\in X\}$.

Clearly,

$$PAm^*_X(L) = \partial^{\ell}_{c_1}(\partial^{\mathrm{r}}_{c_2}(\sigma^*_R(\{c_1\}L\{c_2\}))).$$

Hence the lemma holds. □

Lemma 7. *Let F be a family of languages closed under concatenation with symbols, union, left and right derivatives.*
1. *If F is closed under $fPAm^*$, then F is closed under $fPAm$.*
2. *If F is closed under PAm^*, then F is closed under PAm.*

Proof. For $L_1, L_2 \subseteq V^*$, let c_1, c_2 be two new symbols. It is easy to see that the following equation holds:

$$PAm_X(L_1, L_2) = \partial^{\ell}_{c_1}(\partial^{\mathrm{r}}_{c_2}(PAm^*_X(\{c_1\}L_1 \cup L_2\{c_2\}))).$$

(The derivatives require that at least one PAm operation is performed, while the markers c_1, c_2 prevent performing more than one such operation.) Note that this relation holds also for PAm. □

Theorem 4.
1. *The families REG and RE are closed under both $fPAm^*$ and PAm^*.*
2. *The family LIN is not closed under $fPAm^*$ and PAm^*.*
3. *The family CF is closed under $fPAm^*$ but it is not closed under PAm^*.*
4. *The family CS is closed neither under $fPAm^*$ nor under PAm^*.*

Proof.
1. The closure under $fPAm^*$ follows from Lemma 6 and the fact that the family of regular languages is closed under iterated splicing (see [1, 8, 7]).

A more involved argument is required for proving the closure under PAm^* (remember that the regularity is not preserved by an iterated splicing with respect to a regular set of splicing rules – see [6]).

Let $R \subseteq V^*$ be a regular language recognized by a finite automaton $M = (Q, V, q_0, F, \delta)$, which satisfies the following conditions:

$$F = \{q_f\},\ q_0 \neq q_f,$$
$$\delta(q_f, a) = \emptyset \quad \text{for all } a \in V,$$
$$q_0 \notin \delta(q_0, x) \quad \text{for each } x \in V^+.$$

Clearly, each regular language is accepted by a finite automaton satisfying the above conditions.

We construct now iteratively a sequence of finite automata with λ-moves, $M_0, M_1, \ldots, M_i, \ldots$ with $M_i = (Q, V, q_0, \{q_f\}, \delta_i)$ as follows:
- $M_0 = (Q, V, q_0, \{q_f\}, \delta_0) = M$.
- $M_{i+1} = (Q, V, q_0, \{q_f\}, \delta_{i+1})$ is obtained from M_i as follows.
  - $\delta_{i+1}(s, a) = \delta_i(s, a)$, for all $s \in Q, a \in V \cup \{\lambda\}$.

- For all pairs of different states $q, q' \in Q - \{q_0, q_f\}$ such that:
  (i) $q \notin \delta_i(q', \lambda)$,
  (ii) $L(M_q) \cap L(M^{q'}) \neq \emptyset$,
  where

$$M_q = (Q, V, q_0, \{q\}, \delta_i), \quad \text{and} \quad M^{q'} = (Q, V, q', \{q_f\}, \delta_i),$$

we set

$$\delta_{i+1}(q', \lambda) = \delta_{i+1}(q', \lambda) \cup \{q\}.$$

Obviously, the above sequence is finite, because there exists k such that $M_{k+1} = M_k$ (the set of states is not changed, only new transitions are added); hence $M_{k+p} = M_k$, for all $p \geqslant 0$. Note also that the construction is effective due to the decidability of the emptiness problem for the intersection of two regular languages. Furthermore,

$$R = L(M_0) \subseteq L(M_1) \subseteq L(M_2) \subseteq \cdots \subseteq L(M_k) = L(M_{k+1}) = \cdots \subseteq PAm^*(R)$$

holds. On the other hand, one may easily prove by induction that $PAm^j(R) \subseteq L(M_j)$, for all $j \geqslant 0$; therefore $PAm^*(R) = L(M_k)$.

2. Because the family *LIN* is closed neither under $fPAm$ (Corollary 1) nor under *PAm* (Theorem 2), by Lemma 7 it follows that it is not closed under the iterated versions of these operations.

3. It is known that the family *CF* is closed under iterated splicing [9]; thus, the closure of *CF* under $fPAm^*$ follows from Lemma 6. By Lemma 7 and Theorem 2, we get the non-closure of *CF* under PAm^*.

4. Consider now a language $L \in RE - CS$, $L \subseteq V^*$. There are $a_1, a_2 \notin V$ and a context-sensitive language $L' \subseteq L\{a_1\}\{a_2\}^*$ such that for each $w \in L$ there is $i \geqslant 0$ with $wa_1a_2^i \in L'$. We have then

$$\partial^{\mathrm{r}}_{a_1}(PAm^*_{\{a_2, a_2c\}}(L'\{a_2, a_2^2\} \cup \{a_2c\}) = L.$$

Indeed, the first *PAm* operation transforms strings $wa_1a_2^n \in L'$ into $wa_1a_2^{n-1}c$. The next step leads to $wa_1a_2^{n-2}$ and the process can be iterated. The right derivative with respect to a_1 selects from $PAm^*_{\{a_2, a_2c\}}(L'\{a_2, a_2c\} \cup \{a_2c\})$ the strings of the form wa_1. Since we nondeterministically concatenate L' with both a_2 and a_2^2, in this way we can get wa_1 for all $w \in L$. Thus, the equality follows.

If the family *CS* was closed under the operation $PAm^*_{\{a_2, a_2c\}}$, then $L \in CS$, which is a contradiction. □

As a matter of fact, the non-closure of the families *CF* and *CS* under iterated arbitrary PA-matching may be obtained from a more general result.

Theorem 5. *Each recursively enumerable language $L \subseteq V^*$ can be written as $L = \partial^{\ell}_{c_1}(\partial^{\mathrm{r}}_{c_2}(PAm^*(L') \cap \{c_1\}V^*\{c_2\}))$, where L' is a context-free language and c_1, c_2 are two new symbols.*

Proof. Assume that L is generated by a type-0 grammar $G=(N,V,S,P)$ in the Geffert normal form, [3], that is, with $N=\{S,A,B,C\}$ and P having only context-free rules of the form $S\to x, x\in(N\cup V)^+$, and a single extra rule $ABC\to\lambda$. Consider the context-free grammar $G'=(\{S\},V\cup\{A,B,C,X\},S,\{S\to h(x)\mid S\to x\in P\})$, where X is a new symbol, and h is a morphism that replaces A by XA leaving all the other symbols unchanged. Consider the language

$$L'=\{c_1\}L(G')\{c_2\}\cup\{XABCwc_2Yw^RZ\mid w\in(V\cup\{B,C\})^*\}$$
$$\cup\{YwZw^Rc_2\mid w\in(V\cup\{B,C\})^*\},$$

where Y,Z are two new symbols. Clearly, L' is a context-free language.

Let $c_1w_1XABCw_2c_2$ be a string in $\{c_1\}L(G')\{c_2\}$, with $w_2\in(V\cup\{B,C\})^*$ (that is, this is the rightmost occurrence of $XABC$ in our string). The only possible PA-matching operation is

$$PAm(c_1w_1XABCw_2c_2, XABCw_2c_2Yw_2^RZ)=c_1w_1Yw_2^RZ.$$

The obtained string can again "enter" only one operation:

$$PAm(c_1w_1Yw_2^RZ, Yw_2^RZ(w_2^R)^Rc_2)=c_1w_1w_2c_2.$$

In this way, one occurrence of $XABC$ has been removed. By iterating the PAm operation, all such substrings can be removed – therefore $\{c_1\}L\{c_2\}=PAm^*(L')\cap\{c_1\}V^*\{c_2\}$ holds. The left and the right derivatives lead now to L. □

As a direct consequence of the above result, we find that every family of languages that contains all context-free languages but not all recursively enumerable languages, and is closed under intersection with regular sets and right and left derivatives, is not closed under PAm^*. This is the case for most of the language families in the regulated rewriting area [2]. Moreover, the above result implies some undecidability results.

Corollary 3. *The following problems are undecidable:*
1. *For an arbitrary $L\in CF$, is $PAm^*(L)$ regular/context-free?*
2. *For an arbitrary $L\subseteq V^*$, $L\in CF$, does $w\in V^*$ belong to $PAm^*(L)$?*

4. The overlapping operation

In this section we consider another operation on languages that may be viewed as the dual of PA-matching. While the PA-matching operation removes the matched part, the *overlapping* operation preserves the matched part and removes the rest.

More precisely, for strings x,y we define

$$Ov(x,y)=PSuf(x)\cap PPref(y).$$

Then,

$$Ov(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} Ov(x, y) = PSuf(L_1) \cap PPref(L_2).$$

We write $Ov(L)$ instead of $Ov(L, L)$. The closure properties of the language families in the Chomsky hierarchy under the overlapping operation are the same as for the PA-matching operation.

Theorem 6. 1. *The families REG and RE are closed under Ov.*
2. *The families LIN, CF, and CS are not closed under Ov.*

Proof. The first assertion follows from the closure of both families under intersection, $PPref$ and $PSuf$.

It is easy to see that the closure under overlapping, together with other "easy" closure properties (concatenation with symbols, left and right derivatives), implies the closure under intersection ($L_1 \cap L_2 = \partial^{\ell}_{c_1}(\partial^{r}_{c_2}(Ov(\{c_1^2\}L_1\{c_2\}, \{c_1\}L_2\{c_2^2\})))$) and the prefix operation ($Pref(L) = \partial^{\ell}_{c}(\{c^2\}V^*, \{c\}L\{c\})$). These observations imply the second claim. □

From the previous proof it follows that the fixed point problem for Ov is decidable for regular languages. The problem remains undecidable for context-free languages (with the same proof as for PAm).

Now, let us consider the iterated version of the overlapping operation. The usual way of defining an iterated operation (see the case of the splicing and the case of PA-matching) does not work for the iterated overlapping, because $Ov(Ov(L)) \subseteq Ov(L)$, which makes the usual definition ($Ov^{k+1}(L) = Ov^k(L) \cup Ov(Ov^k(L))$) uninteresting. Therefore, we shall define $Ov^{k+1}(L) = Ov(Ov^k(L))$, for all $k \geqslant 1$. Moreover, $Ov^*(L) = L'$ iff the following two conditions are fulfilled:

(i) $L' \subseteq Ov^k(L)$, for all $k \geqslant 1$,

(ii) for each L'' with $L' \subset L''$ there exists $k \geqslant 1$ such that $L'' \not\subseteq Ov^k(L)$.

This means that, $Ov^*(L)$ is the largest language (with respect to inclusion) which is included in all the sets $Ov(L), Ov^2(L), \ldots$

Theorem 7. 1. *For each $k \geqslant 1$ there is a language L_k such that $Ov^*(L_k) = Ov^k(L_k)$.*
2. *There are languages L such that $Ov^{k+1}(L) \subset Ov^k(L)$, for all $k \geqslant 1$.*

Proof. Consider the language $L_k = \{a^i b^j \mid 1 \leqslant i, j \leqslant k\}$. It is easy to see that $Ov(L_m) = L_{m-1}$, $2 \leqslant m \leqslant k$, and $Ov(L_1) = \emptyset$. Therefore, $Ov^*(L_k) = Ov^k(L_k)$.

Consider also the language

$$L_\infty = \bigcup_{n \geqslant 1} \{(ba^n b)^i (ca^n c)^j \mid 1 \leqslant i, j \leqslant n\}.$$

For each n, we can overlap only strings containing blocks $ba^n b, ca^n c$. For given n, we can perform a bounded number of overlappings, because at each step we have to

remove either the prefix ba^nb or the suffix ca^nc. Therefore $Ov^{k+1}(L_\infty) \neq Ov^k(L_\infty)$ for $k \leqslant n$. Because n can be arbitrarily large, the operation can be iterated for an arbitrarily large number of steps. □

Note that $Ov^k(L_\infty) \neq \emptyset$, but $Ov^*(L_\infty) = \emptyset$.

Theorem 8. *The families LIN and CF are not closed under* Ov^*.

Proof. For $L_1, L_2 \subseteq V^*$, let us consider two new symbols, c_1, c_2. We obtain the equality:

$$Ov^*(\{c_1\}^*L_1\{c_2\}^* \cup \{c_1\}^*L_2\{c_2\}^*) \cap \{c_1\}V^*\{c_2\} = \{c_1\}(L_1 \cap L_2)\{c_2\}.$$

Indeed, $\{c_1\}^*(L_1 \cap L_2)\{c_2\}^* \subseteq Ov(\{c_1\}^*L_1\{c_2\}^* \cup \{c_1\}^*L_2\{c_2\}^*)$. Starting from strings in $\{c_1\}^*(L_1 \cap L_2)\{c_2\}^*$, we can iterate the overlapping operation an arbitrarily large number of times.

By this equation, the closure under Ov^* implies the closure under intersection. Since the families *LIN* and *CF* are not closed under intersection (but they are closed under concatenation with regular languages, intersection with regular languages, union, and left and right derivatives), the theorem holds. □

Theorem 9. *The family CS is not closed under* Ov^*.

Proof. Let $L \subseteq V^*$ and let c_1, c_2 be new symbols (not in V). Consider the language

$$L' = \{c_1\}^*V^*\{c_2\}^* \cup \{c_1\}^*((L \sqcup\!\sqcup \{c_2\}^*) \cap V^*\{c_2\}^*V^*).$$

This is a context-sensitive language. It is easy to see that

$$Ov^*(L') \cap \{c_1\}V^*\{c_2\} = \{c_1\}Pref(L)\{c\}.$$

(We have $\{c_1\}^*Pref(L)\{c_2\}^* \subseteq Ov(L')$, hence we can iterate the operation Ov an arbitrarily large number of times.)

Because the family *CS* is closed under right and left derivatives, but not under the operation *Pref*, we obtain the non-closure under Ov^*. □

Clearly, *RE* is closed under the iterated overlapping operation. The case of the family *REG* will be settled below (also in affirmative), after establishing two auxiliary results.

Let $A = (Q, V, q_0, F, \delta)$ be a minimal complete deterministic finite automaton; because the automaton is complete, the mapping δ is total and a dead state exists from which there is no path to a final state. Let $\mathscr{A}$ be the set of all finite automata of the form $A_{p,q} = (Q, V, p, \{q\}, \delta)$, for $p, q \in Q$. Clearly, this also is a finite set. We denote by $L(\mathscr{A})$ the family of all languages recognized by automata in $\mathscr{A}$ and by $CL(\mathscr{A})$ the closure of the family $L(\mathscr{A})$ under finite union and finite intersection operations. Because $L(\mathscr{A})$ is a finite family, $CL(\mathscr{A})$ also is a finite family of languages.

Lemma 8. *The family $CL(\mathscr{A})$ is closed under complementation.*

Proof. Let L be a language in $CL(\mathscr{A})$. It can be written in the form $L=(L_{1,1}\cap\cdots\cap L_{1,n_1})\cup\cdots\cup(L_{m,1}\cap\cdots\cap L_{m,n_m})$, where each language $L_{i,j}, 1\leqslant i\leqslant m, 1\leqslant j\leqslant n_i$, is an element of $L(\mathscr{A})$. The complement of each language $L_{i,j}$ (we denote the complement of a language K by $\overline{K}$) is also in $L(\mathscr{A})$, since A was a complete deterministic automaton. Because $\overline{L}=(\overline{L_{1,1}}\cup\cdots\cup\overline{L_{1,n_1}})\cap\cdots\cap(\overline{L_{m,1}}\cup\cdots\cup\overline{L_{m,n_m}})$, it follows that also the complement of L is in $CL(\mathscr{A})$. □

Lemma 9. *The family $CL(\mathscr{A})$ is closed under the non-iterated overlapping operation.*

Proof. Let L be a language in $CL(\mathscr{A})$. We write it in the form $L=T_1\cup\cdots\cup T_n$, where each $T_i, 1\leqslant, i\leqslant n$ is a finite intersection of languages in $CL(\mathscr{A})$. For every integer $i=1,\ldots,n$, denote:

$$
\begin{aligned}
K_i &= \{x\in V^* \mid \partial^{\ell}_x(T_i)=\{\lambda\}\},\\
M_i &= \{x\in V^* \mid \partial^{\ell}_x(T_i)=\emptyset\},\\
P_i &= \{x\in V^* \mid \partial^{\mathrm{r}}_x(T_i)=\{\lambda\}\},\\
R_i &= \{x\in V^* \mid \partial^{\mathrm{r}}_x(T_i)=\emptyset\}.
\end{aligned}
$$

Note that the following assertions hold for each string $x\in V^*$:

$$
\begin{aligned}
x\in L-Ov(L) &\Leftrightarrow \partial^{\ell}_x(L)=\{\lambda\} \text{ or } \partial^{\mathrm{r}}_x(L)=\{\lambda\}\\
&\Leftrightarrow (\partial^{\ell}_x(T_i)\subseteq\{\lambda\} \text{ for all } i \text{ and there is some} j\\
&\quad \text{such that } \partial^{\ell}_x(T_j)\neq\emptyset), \text{ or}\\
&\quad (\partial^{\mathrm{r}}_x(T_i)\subseteq\{\lambda\} \text{ for all } i \text{ and there is some } j\\
&\quad \text{such that } \partial^{\mathrm{r}}_x(T_j)\neq\emptyset).
\end{aligned}
$$

Consequently, we have

$$
\begin{aligned}
L-Ov(L) = &\left(\bigcap_{1\leqslant i\leqslant n}(K_i\cup M_i)-\bigcap_{1\leqslant i\leqslant n}M_i\right)\\
&\cup\left(\bigcap_{1\leqslant i\leqslant n}(P_i\cup R_i)-\bigcap_{1\leqslant i\leqslant n}R_i\right).
\end{aligned}
$$

By Lemma 8, it suffices to prove that for every $i=1,\ldots,n$ the languages K_i, M_i, P_i, R_i are contained in $CL(\mathscr{A})$.

Consider any $T_i=L_1\cap\cdots\cap L_m$, where $L_j\in L(\mathscr{A}), 1\leqslant j\leqslant m$. For every j there is an automaton $A_j=(Q,V,p_j,\{f_j\},\delta)$ in $\mathscr{A}$ such that $L_j=L(A_j)$.

(a) In the standard manner, construct the product automaton $A_K=(Q^m,V,\vec{p},\{\vec{f}\},\vec{\delta})$ which accepts T_i, where $\vec{p}=(p_1,\ldots,p_m)$ and $\vec{f}=(f_1,\ldots,f_m)$. If A_K has a cycle

which contains the final state $\vec{f}$, then $K_i = \emptyset$ and this is a language in $CL(\mathscr{A})$. Otherwise, $K_i = T_i$, which is again in $CL(\mathscr{A})$.

(b) For the previous automaton A_K, let F_M be the set of states which appear on a path from $\vec{p}$ to $\vec{f}$. Consider the automaton $A_M = (Q^m, V, \vec{p}, F_M, \vec{\delta})$. Then,

$$\overline{M_i} = L(A_M) = \bigcup_{\vec{q} \in F_M} L((Q^m, V, \vec{p}, \{\vec{q}\}, \vec{\delta})).$$

For every state $\vec{q} = (q_1, \ldots, q_m) \in F_M$, the language accepted by the automaton $(Q^m, V, \vec{p}, \{\vec{q}\}, \vec{\delta})$ is an intersection of the languages $L((Q, V, p_j, \{q_j\}, \delta))$, which are in $L(\mathscr{A})$ for all $j = 1, \ldots, m$. Consequently, $M_i \in CL(\mathscr{A})$.

(c) The proof of the relation $R_i \in CL(\mathscr{A})$ can be obtained in the same manner as in the case of M_i.

(d) For each automaton $A_j = (Q, V, p_j, \{f_j\}, \delta)$ as above, we construct its reversal, $A_j^R = (Q, V, f_j, \{p_j\}, \delta^R)$ and we make A_j^R deterministic by the usual subset construction technique. Let $A_{j*}^R = (2^Q, V, \{f_j\}, F_j, \delta_*^R)$ be the automaton obtained in this way. Then, by definition, $L(A_{j*}^R) = L_j^R$ holds.

We construct the product $((2^Q)^m, V, \vec{s}, F, \vec{\delta}_*^R)$ of automata $A_{1*}^R, \ldots, A_{m*}^R$, which accepts $T_i^R = L_1^R \cap \cdots \cap L_m^R$, where $\vec{s} = (\{f_1\}, \ldots, \{f_m\})$. Let F_P be the set of all states $\vec{q}$ in F such that $\vec{q}$ does not have a path to any state of F. Let $A_P = ((2^Q)^m, V, \vec{s}, F_P, \vec{\delta}_*^R)$. Then, we have $P_i^R = L(A_P)$. Thus,

$$\begin{aligned} P_i &= L(A_P^R) \\ &= L(((2^Q)^m, V, F_P, \{\vec{s}\}, (\vec{\delta}_*^R)^R)) \\ &= \bigcup_{\vec{q} \in F_P} L(((2^Q)^m, V, \vec{q}, \{\vec{s}\}, (\vec{\delta}_*^R)^R)). \end{aligned}$$

It suffices now to show that for every $\vec{q} \in F_P$, the language $Z_{\vec{q}} = L(((2^Q)^m, V, \vec{q}, \{\vec{s}\}, (\vec{\delta}_*^R)^R))$ is in $CL(\mathscr{A})$. Let $\vec{q} = (E_1, \ldots, E_m) \in F_P$, where $E_j \subseteq Q, 1 \leqslant j \leqslant m$. Note that for any $(X_1, \ldots, X_m)$ and $(Y_1, \ldots, Y_m)$ in $(2^Q)^m$ and for any $a \in V$, the following assertions hold:

$$\begin{aligned} &(Y_1, \ldots, Y_m) \in (\vec{\delta}_*^R)^R((X_1, \ldots, X_m), a) \\ &\quad \Leftrightarrow \vec{\delta}_*^R((Y_1, \ldots, Y_m), a) = (X_1, \ldots, X_m) \\ &\quad \Leftrightarrow \delta_*^R(Y_j, a) = X_j \quad \text{for all } j = 1, \ldots, m, \\ &\quad \Leftrightarrow X_j = \{q \mid q \in \delta^R(p, a) \quad \text{for some } p \in Y_j\}, \text{for all } 1 \leqslant j \leqslant m, \\ &\quad \Leftrightarrow X_j = \{q \mid p = \delta(q, a), \text{for some } p \in Y_j\}, \text{ for all} 1 \leqslant j \leqslant m. \end{aligned}$$

Therefore, we have

$$Z_{\vec{q}} = \bigcap_{1 \leqslant j \leqslant m} \left(\bigcap_{r \in E_j} L((Q, V, r, \{f_j\}, \delta)) - \bigcup_{r \notin E_j} L((Q, V, r, \{f_j\}, \delta)) \right),$$

hence this language is in $CL(\mathscr{A})$ and this completes the proof of the lemma. □

Theorem 10. *The family REG is closed under the operation* Ov^*.

Proof. Starting from a minimal deterministic finite automaton A for a regular language L, we construct the family $CL(\mathscr{A})$ as above. Because this family is closed under non-iterated overlapping, all languages $Ov^k(L), k \geqslant 1$, are in $CL(\mathscr{A})$. Because the family $CL(\mathscr{A})$ is finite, it follows that only finitely many languages $Ov^k(L)$ are different to each other. The smallest of them is equal to $Ov^*(L)$ and it is an element of $CL(\mathscr{A})$. It follows that $Ov^*(L)$ is a regular language. □

References

[1] K. Culik II, T. Harju, Splicing semigroups of dominoes and DNA, Discrete Appl. Math. 31 (1991) 261–277.
[2] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, Springer, Berlin, 1989.
[3] V. Geffert, Normal forms for phrase-structure grammars, RAIRO/Theoret. Inform. Appl. 25 (5) (1991) 473–496.
[4] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bull. Math. Biol. 49 (1987) 737–759.
[5] T. Head, Gh. Păun, D. Pixton, Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, in: Regulated Rewriting in Formal Language Theory, vol. 2, Springer, Berlin, 1989, pp. 295–360 (Chapter 7).
[6] Gh. Păun, Regular extended H systems are computationally universal, J. Automat. Languages Combin. 1 (1) (1996) 27–36.
[7] Gh. Păun, G. Rozenberg, A. Salomaa, DNA Computing. New Computing Paradigms, Springer, Heidelberg, 1998.
[8] D. Pixton, Regularity of splicing languages, Discrete Appl. Math. 69 (1996) 101–124.
[9] D. Pixton, Splicing in abstract families of languages, Tech. Rep., SUNY University, Binghamton, New York, 1997.
[10] J.H. Reif, Parallel molecular computation: models and simulations, Proc. 7th Annual ACM Symp. on Parallel Algorithms and Architectures, Santa Barbara, 1995, pp. 213–223.
[11] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, 3 vols., Springer, Heidelberg, 1997.
[12] A. Salomaa, Formal Languages, Academic Press, New York, 1973.