Fundamenta Informaticae 170 (2019) 1–23
DOI 10.3233/FI-2019-1850
IOS Press

Word Blending in Formal Languages*

Srujan Kumar Enaganti
Department of Computer Science
University of Western Ontario, Canada
srujankumar@gmail.com

Lila Kari & Timothy Ng & Zihao Wang†
School of Computer Science
University of Waterloo, Canada
lila.kari@uwaterloo.ca, tim.ng@uwaterloo.ca, z465wang@uwaterloo.ca

Abstract. In this paper we define and investigate a binary word operation that formalizes an experimentally observed outcome of DNA computations, performed to generate a small gene library, and implemented using a DNA recombination technique called Cross-pairing Polymerase Chain Reaction (XPCR). The *word blending* between two words $\alpha w \gamma_1$ and $\gamma_2 w \beta$ that share a non-empty overlap w, results in $\alpha w \beta$. Interestingly, this phenomenon has been observed independently in linguistics, under the name "blend word" or "portmanteau", and is responsible for the creation of words in the English language such as smog (*smo*ke + f*og*), labradoodle (*labrado*r + p*oodle*), and Brangelina (*Bra*d + *Angelina*). Technically, word blending is related to the binary word operation Latin product, the crossover operation, and simple splicing. We study closure properties of the families in the Chomsky hierarchy under word blending, language equations involving this operation, and its descriptional state complexity when applied to regular languages. We also define iterated word blending and show that, for a given alphabet, there are finitely many languages that can be obtained from an initial language by iterated word blending.

*This research was supported by the NSERC (Natural Sciences and Engineering Research Council of Canada) Discovery Grant R2824A01, and a University of Waterloo School of Computer Science Grant to L.K.
†Address for correspondence: School of Computer Science, University of Waterloo, Canada

1. Introduction

Cross-pairing Polymerase Chain Reaction (XPCR) is an experimental DNA protocol introduced in [1] for extracting, from a heterogeneous pool of DNA strands, all the strands containing a given substrand. XPCR was then employed to implement several DNA recombination algorithms [2], for the creation of the solution space for a SAT problem [3], and for mutagenesis [4]. The combinatorial power of such a technique has been explained by logical-symbolic schemes in [5], while algorithms to create combinatorial libraries were improved and experimented in [4, 6].

The formal language operation called *overlap assembly*, introduced in [7] under the name of self-assembly, and further investigated in [8, 9, 10], also models a special case of XPCR: The overlap assembly of two strings αx and $x\beta$ that share a non-empty overlap x results in the string $\alpha x\beta$. A particular case of overlap assembly, called "chop operation", where the overlap consists of a single letter, was studied in [11, 12], and generalized to an arbitrary-length overlap in [13]. Other similar operations have been studied in the literature, such as the "short concatenation" [14], which uses only the maximum-length (possibly empty) overlap x between operands, the "Latin product" of words [15] where the overlap occurs at the extremities of words and consists of only one letter, the "crossover" operation [16] where an overlap consisting of only one letter may occur in the middle of the words, and the operation $\otimes$ which imposes the restriction that at least one of the non-overlapping parts α, β is not empty [17]. Overlap assembly can also be considered as a particular case of "semantic shuffle on trajectories" with trajectory $0^*\sigma^+1^*$ or as a generalization of the operation $\odot_N$ from [18] which imposes the length of the overlap to be at least N. Many similar biological phenomena and operations can also be modelled using splicing systems [19, 20]. However, modeling these operations often does not require the full power of splicing. Properties of splicing languages under restrictions such as symmetry and reflexivity have been studied in [21, 22].

Returning to the biological process that motivated the study of overlap assembly, the XPCR procedure has been successfully used to join two different genes if they are attached to compatible primers [6]. Formally, $\alpha A\gamma$ and $\gamma D\beta$ were combined to produce $\alpha A\gamma D\beta$ (here A and D are gene sequences and α, γ and β are primers). However, when $A = D$, that is, when two sequences containing the same gene were combined by XPCR, the result was not as expected. More specifically, when using XPCR with the input $\alpha A\gamma$, $\gamma A\beta$, α and β, instead of obtaining the expected $\alpha A\gamma A\beta$, the experiments repeatedly produced the result $\alpha A\beta$.

In this paper[1], we define and investigate a binary word and language operation called *word blending*, that formalizes a generalization of this experimentally observed outcome of XPCR: The *word blending* of two words xAy_1 and y_2Az that share a non-empty overlap A results in xAz. Note that while $y_1 = y_2$ was experimentally observed, we do not require this for our definition of the operation. Interestingly, this phenomenon has been observed independently in linguistics [24], under the name "blend word" or "portmanteau", and is responsible for the creation of words in the English language such as smog (*smo*ke + f*og*), labradoodle (*labrado*r + p*oodle*), emoticon (*emoti*on + *icon*), and Brangelina (*Bra*d + *Angelina*).

[1]The paper is an extended and modified version of [23]. In particular, this paper contains complete proofs for all results, numerous examples, expanded discussion of inverses of blending, new results concerning the iterated blending of languages, revised state complexity results, and remarks on open problems.

The paper is organized as follows. Section 2 details the biological motivation behind the study of word blending, and introduces the main definitions and notations. Section 3 studies closure properties of the families in the Chomsky hierarchy under word blending, its right- and left-inverses, as well as iterated word blending. Section 4 investigates decision problems, such as the decidability of existence of solutions to some language equations involving word blending. In this section we also investigate iterated word blending, and prove that for a given alphabet there are finitely many languages that can be obtained by applying iterated word blending to a given initial language. Section 5 studies the descriptional state complexity of the word blending operation when applied to regular languages.

2. Preliminaries

An alphabet Σ is a finite non-empty set of symbols. We denote by Σ^* the set of all words over Σ, including the empty word λ, and Σ^+ denotes the set of all non-empty words over Σ. The length of the word w is denoted by $|w|$, and the number of occurrences of the letter a in a word w is denoted by $|w|_a$. For words $w, x, y, z \in \Sigma^*$, where $w = xyz$, we call the subwords x, y, and z *prefix*, *infix*, and *suffix* of w, respectively. The sets $\mathrm{pref}(w)$, $\mathrm{inf}(w)$, and $\mathrm{suff}(w)$ contain, respectively, all prefixes, infixes, and suffixes of w. This notation is extended to languages, e.g., $\mathrm{suff}(L) = \bigcup_{w \in L} \mathrm{suff}(w)$. The mirror image of a word $w \in \Sigma^*$ is defined as $\mathrm{mi}(\lambda) = \lambda$, and $\mathrm{mi}(w) = a_k \dots a_2 a_1$ iff $w = a_1 a_2 \dots a_k$. The definition is extended to languages in the natural way, by $\mathrm{mi}(L) = \bigcup_{w \in L} \mathrm{mi}(w)$. The complement of a language $L \subseteq \Sigma^*$ is $L^c = \Sigma^* \backslash L$. For two languages L_1 and L_2, the right quotient of L_1 by L_2 is defined as $L_1 \div_r L_2 = \{u \in \Sigma^* \mid uv \in L_1, v \in L_2\}$, and the left quotient of L_1 by L_2 is defined as $L_1 \div_l L_2 = \{v \in \Sigma^* \mid uv \in L_1, u \in L_2\}$. We adopted this notation to clearly differentiate between the left and right quotient, and because the multiplication symbol $\cdot$ is sometimes used to denote catenation. We also use the convention that catenation has a higher precedence than left and right quotient.

An *abstract family of languages* (AFL) is a family of languages closed under λ-free morphism, inverse morphism, intersection with regular languages, union, and positive Kleene closure. A full AFL is a family of languages closed under morphism, inverse morphism, intersection with regular languages, union, and Kleene closure.

The biological phenomenon we model in this paper was observed during the XPCR-based experiments, initially intended to achieve the catenation of two or more genes (genomic DNA strands). It was namely observed in [6] that, in the particular case where the two genes to be catenated were one and the same, that is, when the two input DNA strands were $\alpha A \gamma$ and $\gamma A \beta$ (here A represents a gene sequence), the output of an PCR-based amplification with primers α and β was $\alpha A \beta$. This output was different from the expected $\alpha A \gamma A \beta$, which had been the anticipated result. Indeed, experiments using XPCR for the purpose of catenating two different genes A and D flanked by primers, that is, when the two input strands were $\alpha A \gamma$ and $\gamma D \beta$, had resulted in the output $\alpha A \gamma D \beta$. This "expected" output of XPCR was modelled by the previously mentioned operation of overlap assembly $\overline{\odot}$, which would produce $\alpha A \gamma D \beta$ as an element of the set $\alpha A \gamma \ \overline{\odot} \ \gamma D \beta$. (Recall that $x \overline{\odot} y = \{z \in \Sigma^+ \mid \exists u, w \in \Sigma^*, \exists v \in \Sigma^+$ such that $x = uv, y = vw, z = uvw\}$.)

We will generalize this experimentally newly-observed phenomenon by considering the case where the end words of the input strings can also be different. We model this string recombination as follows.

Definition 2.1. Given two words x, y over an alphabet Σ, we define the *word blending*, or simply *blending*, of x with y as

$$x \bowtie y = \{z \in \Sigma^+ \mid \exists \alpha, \beta, \gamma_1, \gamma_2 \in \Sigma^*, \exists w \in \Sigma^+ \text{ such that } x = \alpha w \gamma_1, y = \gamma_2 w \beta, z = \alpha w \beta\}.$$

Note that the result of blending between two words is the empty set (the operation is undefined), if the words do not share a non-empty infix. If one or both words are empty, the operation is similarly undefined. The definition of blending can be extended to languages L_1 and L_2 by

$$L_1 \bowtie L_2 = \bigcup_{x \in L_1, y \in L_2} x \bowtie y.$$

Example 2.2. First, consider $u = bbac$ and $v = caab$. Then $u \bowtie v = \{b, bb, bbab, bbaab, bbacaab\}$ and $v \bowtie u = \{c, cac, caac, caabac, caabbac\}$. Next, consider $L_1 = \{a^*b^*\}$ and $L_2 = \{b^*c^*\}$. Then $L_1 \bowtie L_2 = \{a^*b^+c^*\}$ and $L_2 \bowtie L_1 = \{b^+\}$. It is clear from this that blending is not commutative.

Remark 2.3. We emphasize that the definition of blending is a generalization of the experimentally observed outcome of XPCR on DNA strings. Indeed, to match the experimentally-observed process, we would have to take $\gamma_1 = \gamma_2$ in Definition 2.1. We also note that for a realistic model we would need additional restrictions, such as the fact that the words w, γ_1 and γ_2 should be of a sufficient length, and that these words should not appear as a substring in the other strings involved.

The blending operation resembles the Latin product [15] and the crossover operation [16, 25]. The Latin product, denoted by $\diamond$ was defined as $u \diamond v = u'av'$ for $u = u'a$, $v = av'$ with $a \in \Sigma$, $u', v' \in \Sigma^*$ and, by definition, $u \diamond \lambda = \lambda \diamond u = u$. Even though, as will be proven in Lemma 3.1, word blending is equivalent to single-letter-overlap word blending, the Latin product differs from blending: In the Latin product the overlap and blending can only occur at the extremities of the words, while in word blending it can happen anywhere inside the two operand words. Given a subset of the alphabet $M \subseteq \Sigma$, the crossover operation $\sharp_M$ was defined as $L_1 \sharp_M L_2 = \mathrm{pref}(L_1) \diamond_M \mathrm{suff}(L_2)$, where $\diamond_M$ is a restriction of the Latin product, defined by

$$u \diamond_M v = \begin{cases} u'av' & \text{if } u = u'a,\ v = av',\ a \in M,\ u', v' \in \Sigma^*, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

By definition, $u \diamond_M \lambda = \lambda \diamond_M u = u$, and the crossover on languages is defined as $L_1 \diamond_M L_2 = \bigcup_{u \in L_1, v \in L_2} (u \diamond_M v)$. The blending operation resembles the crossover operation $\sharp_M$ with $M = \Sigma$. However, due to its biological motivation, unlike the Latin product and the crossover operation, blending is not defined when one of the operands is λ, and this leads to additional differences. For example, the crossover using $M = \Sigma = \{a, b\}$ between ab and ba is $ab \sharp_\Sigma ba = \mathrm{pref}(ab) \diamond \mathrm{suff}(ba) = \{\lambda, a, ab, ba, aba\}$, while the Latin product of the same words is $ab \diamond ba = \{aba\}$, and the blending between the same words is $ab \bowtie ba = \{a, aba\}$. Also, both the Latin product and the crossover operation $\sharp_M$ for a given M, are associative, while word blending is not. For example, $(ba \bowtie a) \bowtie b^*a = ba \bowtie b^*a = b^+a$, while $ba \bowtie (a \bowtie b^*a) = ba \bowtie a = ba$.

One can extend the blending operation to an iterated version, as follows. For $L \subseteq \Sigma^*$, the *iterated (word) blending* of L is defined by $L^{\bowtie_0} = L$, and $L^{\bowtie_i} = L \bowtie L^{\bowtie_{i-1}}$, $i \geq 1$. We define the iterated blending closure of L by

$$L^{\bowtie_*} = \bigcup_{i \geq 0} L^{\bowtie_i}.$$

We observe that the result of the iterated blending operation resembles the result of a splicing system. A splicing rule is of the form $(u_1, u_2; u_3, u_4)$ [20]. The splicing of two strings $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$, with $x_1, x_2, y_1, y_2, u_1, u_2, u_3, u_4 \in \Sigma^*$, using the splicing rule $(u_1, u_2; u_3, u_4)$, results in the word $x_1u_1u_4y_2$. It is easy to see that the language $L^{\bowtie_*}$ can be generated by using a set of rules of the form $(w, \lambda; w, \lambda)$ for every word $w \in \Sigma^+$. Splicing rules of this form are called null context splicing rules [26]. In fact, we can show that iterated blending can be expressed as a simple splicing system [25], where the splicing rules are restricted to rules of the form $(a, \lambda; a, \lambda)$ for $a \in \Sigma$. Simple splicing systems were first considered in [25], where the crossover operation described above was also introduced as a "one step" of a simple splicing system. The relationship between iterated blending and splicing will be discussed in greater detail in Section 3.

3. Closure properties

In this section, we prove that the families of regular, context-free and recursively enumerable languages are closed under blending, but that the family of context-sensitive languages is not. The section also contains closure properties of the Chomsky families under the right- and left-inverse of blending, as well as under iterated blending.

The following lemma shows that blending is equivalent to a restricted version where only one-letter overlaps are utilized. A similar such simplification was made for synchronized insertion and deletion and hairpin inversion by Daley et al. [27].

Lemma 3.1. If x, y are non-empty words over Σ, then

$$x \bowtie y = \{z \in \Sigma^+ \mid \exists \alpha, \beta, \gamma_1, \gamma_2 \in \Sigma^*, \exists a \in \Sigma : x = \alpha a \gamma_1, y = \gamma_2 a \beta, z = \alpha a \beta\}.$$

Proof:
Let A denote the right hand side of the equality. The inclusion $A \subseteq x \bowtie y$ is obvious by the definition of blending. To prove the converse, let $z \in x \bowtie y$. Then $z = \alpha w \beta$ for some $\alpha, \beta, \gamma_1, \gamma_2 \in \Sigma^*$ and $w \in \Sigma^+$ such that $x = \alpha w \gamma_1$, $y = \gamma_2 w \beta$. As $w \in \Sigma^+$, w can be written as $w = w_1 a$, where $w_1 \in \Sigma^*$, $a \in \Sigma$. It follows that $x = \alpha w_1 a \gamma_1$, $y = \gamma_2 w_1 a \beta$ and $z = \alpha w_1 a \beta$, that is, $x = \alpha' a \gamma_1$, $y = \gamma_2' a \beta$ and $z = \alpha' a \beta$ with $\alpha' = \alpha w_1 \in \Sigma^*$ and $\gamma_2' = \gamma_2 w_1 \in \Sigma^*$. Thus, $z \in A$, and the equality is proven. □

From the above lemma it follows that word blending, $\bowtie$, is a generalization of the chop operation (also called fusion), as studied in [11, 12], whereby $u \odot v$ equals $u'av'$ if $u = u'a$, $v = av'$, $u', v' \in \Sigma^*$, $a \in \Sigma$, and undefined otherwise. Also note that the chop operation differs from the Latin product only when one of the operands is λ, in which case the result of the Latin product equals the other operand, while the result of the chop operation (fusion) is undefined. Lemma 3.1 can be extended to languages

in the natural way. From this lemma, we can show that the blending of two languages can be obtained by combining the right quotient, catenation, left quotient and union operations, as follows.

Proposition 3.2. Given languages $L_1, L_2 \subseteq \Sigma^*$, $L_1 \bowtie L_2 = \bigcup_{a\in\Sigma}(L_1 \div_r a\Sigma^*)a(L_2 \div_l \Sigma^* a)$.

Proof:
Let $z \in L_1 \bowtie L_2$. Then, by Lemma 3.1, $z = \alpha a\beta$, for some $x \in L_1$ and $y \in L_2$ such that $x = \alpha a\gamma_1$, $y = \gamma_2 a\beta$ where $a \in \Sigma, \alpha, \beta, \gamma_1, \gamma_2 \in \Sigma^*$. It is clear that $\alpha \in L_1 \div_r a\Sigma^*$ and $\beta \in L_2 \div_l \Sigma^* a$, so $z = \alpha a\beta \in (L_1 \div_r a\Sigma^*)a(L_2 \div_l \Sigma^* a)$.

Conversely, let $z \in \bigcup_{a\in\Sigma}(L_1 \div_r a\Sigma^*)a(L_2 \div_l \Sigma^* a)$. Then there exist a letter $a \in \Sigma$ and words $\alpha, \gamma_1, \gamma_2, \beta \in \Sigma^*$, such that $z = \alpha a\beta$, where $x = \alpha a\gamma_1 \in L_1$, $y = \gamma_2 a\beta \in L_2$, which implies that $z \in L_1 \bowtie L_2$. □

Corollary 3.3. The families of regular, context-free and recursively enumerable languages are closed under blending.

Proof:
It follows from Proposition 3.2, the fact that every full AFL is closed under left/right quotient with regular languages, catenation and union, and the fact that the families of regular, context-free and recursively enumerable languages are all full AFLs [28]. □

Proposition 3.4. The family of context-sensitive languages is not closed under blending.

Proof:
Let L_0 be a recursively enumerable language over Σ, that is not context-sensitive. It is known that a context-sensitive language L_1 over $\Sigma \cup \{a, b\}$ with $a, b \notin \Sigma$, can be constructed such that L_1 is a subset of $\{Pba^i \mid P \in L_0, i \geq 0\}$ and, in addition, for every $P \in L_0$ there is an $i \geq 0$ such that $Pba^i \in L_1$ (see, e.g., [28]).

Since it is obvious that $L_1 \bowtie \{b\} = \{Pb \mid P \in L_0\} = L_0 b$, which is not context sensitive, it follows that the family of context-sensitive languages is not closed under blending. □

A binary word operation $\oplus$ is a mapping $\oplus : \Sigma^* \times \Sigma^* \longrightarrow 2^{\Sigma^*}$. Recall that a *right-inverse* of a binary operation $\oplus$ is a binary operation $\oplus_r^{-1}$ defined as follows [29]:

$$w \in (x \oplus y) \text{ iff } y \in (x \oplus_r^{-1} w), \text{ for all } x, y, w \in \Sigma^*.$$

The binary word operation $\oplus_r^{-1}$ was called right-inverse of $\oplus$ because it can be used to "recover" the right operand y of $x \oplus y$, from the other operand x and a word $w \in (x \oplus y)$ in the result. Note that the definition of right-inverse can be rewritten as $x \oplus_r^{-1} w = \{y \in \Sigma^* \mid w \in (x \oplus y)\}$, and that $(\oplus_r^{-1})_r^{-1} = \oplus$. For example, the binary word operations catenation and "reverse left quotient" are right-inverses of each other (given a binary operation $*$, its reverse $*'$ is defined as $u *' v = v * u$).

The right-inverse of an operation does not always resemble a binary operation in the traditional sense. For example, if we define the operation $u*v = \{a^{|u|} \mid a \in \Sigma\}$, where Σ is the alphabet, then the

right-inverse of this operation, $x *_r^{-1} y$, equals Σ^* if $y = a^{|x|}$, and equals $\emptyset$ if $y \neq a^{|x|}$. However, the relation $y \in x *_r^{-1} (x * y)$ still holds, and this will turn out to be sufficient for solving some language equations involving binary word operations, as detailed in Section 4.

Also note that the right-inverse of a binary word operation inherits some of the properties of the original binary operation. For example, a standard property of many familiar binary word operations (catenation, left/right quotient, insertion, deletion, shuffle, etc) is that they have a right (left) identity: A binary word operation with *right identity* is a binary word operation $\oplus$ with the property $u \oplus \lambda = \{u\}$ for all $u \in \Sigma^*$. The notion of binary word operation with *left identity* is defined similarly and, for example, catenation, insertion and shuffle are binary operations with both right and left identity, while deletion and left/right quotient are binary word operations with right identity only. The operation of word blending has neither right identity nor left identity. In general, the following result holds.

Proposition 3.5. If $\oplus$ is a binary word operation with left identity, then its right-inverse $\oplus_r^{-1}$ is also a binary operation with left identity.

Proof:
Let v be a word in $(\lambda \oplus_r^{-1} w)$. By definition of right-inverse, $v \in (\lambda \oplus_r^{-1} w)$ iff $w \in (\lambda \oplus v)$ which, in turn, equals $\{v\}$, as $\oplus$ is an operation with left identity. Thus, for all $v \in (\lambda \oplus_r^{-1} w)$, we have that $v = w$, which means that $\{w\} = (\lambda \oplus_r^{-1} w)$, and therefore $\oplus_r^{-1}$ is a binary operation with left identity. □

Proposition 3.6. The right-inverse of a binary word operation $\oplus$ is unique.

Proof:
Assume the contrary, that is, assume that a binary word operation $\oplus$ has two distinct right-inverses $\oplus_1$ and $\oplus_2$. By definition, for all words $u, y, w \in \Sigma^*$, the following relations hold: $w \in (u \oplus y)$ iff $y \in (u \oplus_1 w)$, and $w \in (u \oplus y)$ iff $y \in (u \oplus_2 w)$.

Since $\oplus_1$ and $\oplus_2$ are distinct, there exist some words α, β such that $(\alpha \oplus_1 \beta) \neq (\alpha \oplus_2 \beta)$. Without loss of generality, assume that there exists a word $\gamma \in (\alpha \oplus_1 \beta)$ such that $\gamma \notin (\alpha \oplus_2 \beta)$.

By definition of $\oplus_1$, from $\gamma \in (\alpha \oplus_1 \beta)$ we have that $\beta \in (\alpha \oplus \gamma)$, which by definition of $\oplus_2$ implies that $\gamma \in (\alpha \oplus_2 \beta)$. This contradicts our assumption about γ, therefore the right-inverse of a binary word operation is unique. □

Definition 3.7. For two words $u, w \in \Sigma^*$ the binary word operation $\bowtie_r^{-1}$ is defined as

$$u \bowtie_r^{-1} w = \bigcup_{a \in \Sigma} \Sigma^* a (w \div_l (u \div_r a\Sigma^*) a).$$

The definition of $\bowtie_r^{-1}$ can be extended to languages by

$$L_1 \bowtie_r^{-1} L_2 = \bigcup_{u \in L_1, w \in L_2} \left(\bigcup_{a \in \Sigma} \Sigma^* a (w \div_l (u \div_r a\Sigma^*) a) \right).$$

We observe that, similar to insertion, deletion, and other binary word operations [29], the result of $u \bowtie_r^{-1} w$ is in general a set of words (not necessarily a singleton word). In general, the operation $\bowtie_r^{-1}$ can be used to recover the right operand of blending two words: The right operand (a word) belongs to the set obtained by applying $\bowtie_r^{-1}$ to the other operand and one of the words in the result of blending.

Example 3.8. Let $\Sigma = \{a, b\}$. We have that $aabb \bowtie_r^{-1} aab = \Sigma^* aab \cup \Sigma^* ab \cup \Sigma^* b = \Sigma^* b$. The result of blending $aabb$ with ab is $aabb \bowtie ab = \{ab, \underline{aab}, aabb\}$. The right operand ab is an element of the set obtained by applying $\bowtie_r^{-1}$ to the other operand and one of the words of the result of $\bowtie$: $ab \in aabb \bowtie_r^{-1} \underline{aab} = \Sigma^* b$.

The next proposition shows that the operation $\bowtie_r^{-1}$ is the right-inverse of $\bowtie$.

Proposition 3.9. The operation $\bowtie_r^{-1}$ is the right-inverse of $\bowtie$.

Proof:
If $w \in u \bowtie y$, there exist $\alpha, \beta, \gamma_1, \gamma_2 \in \Sigma^*, b \in \Sigma$ such that $w = \alpha b \beta, u = \alpha b \gamma_1, y = \gamma_2 b \beta$ by Lemma 3.1. Then, we have that

$$\begin{aligned} y = \gamma_2 b \beta \in \Sigma^* b \beta &= \Sigma^* b (\alpha b \beta \div_l \alpha b) \\ &\subseteq \Sigma^* b (\alpha b \beta \div_l (\alpha b \gamma_1 \div_r b \Sigma^*) b) \\ &\subseteq \bigcup_{a \in \Sigma} \Sigma^* a (\alpha b \beta \div_l (\alpha b \gamma_1 \div_r a \Sigma^*) a) = u \bowtie_r^{-1} w. \end{aligned}$$

Conversely, if $y \in u \bowtie_r^{-1} w = \bigcup_{a \in \Sigma} \Sigma^* a (w \div_l (u \div_r a \Sigma^*) a)$, then there exist $b \in \Sigma$, $\gamma_2 \in \Sigma^*$, and $\gamma_3 \in (u \div_r b \Sigma^*) b$ such that $y = \gamma_2 b (w \div_l \gamma_3)$. This implies that

$$\begin{aligned} w &\in (u \div_r b \Sigma^*) b (w \div_l \gamma_3) \\ &= (u \div_r b \Sigma^*) b (\gamma_2 b (w \div_l \gamma_3) \div_l \gamma_2 b) \\ &\subseteq (u \div_r b \Sigma^*) b (y \div_l \Sigma^* b) \\ &\subseteq \bigcup_{a \in \Sigma} (u \div_r a \Sigma^*) a (y \div_l \Sigma^* a) = u \bowtie y. \end{aligned}$$

□

Corollary 3.10. The operation $\bowtie_r^{-1}$ is the unique right-inverse of the blending operation $\bowtie$.

Corollary 3.11. The families of regular and recursively enumerable languages are closed under the right-inverse of blending. Moreover, if L_1 is an arbitrary language and L_2 is a regular language, then $L_1 \bowtie_r^{-1} L_2$ is regular; if L_1 is a regular language and L_2 is a context-free language, then $L_1 \bowtie_r^{-1} L_2$ is context free.

Proof:
The proof follows from Proposition 3.9. The family of regular languages is closed under right quotient and catenation, and the family of context-free languages is closed under union, catenation and left quotient with regular languages [28, 30], so the right-inverse of blending of a regular language with a context-free language is context free. □

Proposition 3.12. The family of context-free languages is not closed under the right-inverse of blending.

Proof:
Consider the context-free languages $L_1 = \{a\$(b^{i_1}a^{i_1}\$)\cdots(b^{i_n}a^{i_n}\$) \mid n \geq 1, i_m \geq 1 \text{ for } 1 \leq m \leq n\}$, $L_2 = \{(a^{j_1}\$b^{2j_1})\cdots(a^{j_k}\$b^{2j_k})(a^j\$c^{2j}) \mid j \geq 1, k \geq 1, j_m \geq 1 \text{ for } 1 \leq m \leq k\}$ and the regular language $R = \{\$c^*\}$.

We now show that $(L_1 \bowtie_r^{-1} L_2) \cap R = \{\$c^{2^n} \mid n \geq 2\}$. Since words in R start with \$ and contain only one symbol \$, the only cases in which the words in $L_1 \bowtie_r^{-1} L_2$ have the pattern of the words in R are the cases of word pairs where the overlap letter is \$, and a prefix ending in \$ in the word from L_1 matches the prefix ending in the last occurrence of \$ in the word from L_2. More precisely, let $u = a\$b^{i_1}a^{i_1}\$b^{i_2}a^{i_2}\$\cdots b^{i_m}a^{i_m}\$\cdots b^{i_n}a^{i_n}\$ \in L_1$ and $v = a^{j_1}\$b^{2j_1}a^{j_2}\$b^{2j_2}\cdots a^{j_m}\$b^{2j_m}a^j\$c^{2j} \in L_2$. For a word $w \in (L_1 \bowtie_r^{-1} L_2)$ to belong to R, we must have

$$a\$b^{i_1}a^{i_1}\$b^{i_2}a^{i_2}\$\cdots b^{i_m}a^{i_m}\$ = a^{j_1}\$b^{2j_1}a^{j_2}\$b^{2j_2}\cdots a^{j_m}\$b^{2j_m}a^j\$,$$

which implies $j_1 = 1, j_2 = i_1 = 2j_1 = 2, \ldots, j = i_m = 2j_m = 2^m$. Thus, $w = \$c^{2j} = \$c^{2^{m+1}}$, which implies $(L_1 \bowtie_r^{-1} L_2) \cap R = \{\$c^{2^n} \mid n \geq 2\}$.

Since the family of context-free languages is closed under intersection with regular languages, it follows that it is not closed under the right-inverse of blending. □

Proposition 3.13. The family of context-sensitive languages is not closed under the right-inverse of blending.

Proof:
Let L_0 be a recursively enumerable language over Σ, that is not context-sensitive, and L_1 be the language consisting of words of form $a^ibP, P \in L_0, i \geq 0$, constructed similarly as in the proof of Proposition 3.4.

The result now follows as $(L_1 \bowtie_r^{-1} \{a^*b\}) \cap b\Sigma^* = \{bP \mid b \notin \Sigma, P \in L_0\} = bL_0$, which is not context sensitive, and L_1 be the language containing words of form $a^ibP, P \in L_0$ and constructed similarly as in the proof of Proposition 3.4. □

Recall that given a binary word operation $\oplus$, the binary word operation $\oplus_l^{-1}$ was called the *left-inverse of* $\oplus$ iff for all words $x, y, w \in \Sigma^*$ the following relation holds: $w \in (x \oplus y)$ iff $x \in (w \oplus_l^{-1} y)$, [29]. The left-inverse of $\oplus$ can be used to "recover" the left operand x of $x \oplus y$, from a word $w \in (x \oplus y)$ in the result, and the other operand y. Similarly to Proposition 3.5, we can prove that:

Proposition 3.14. If $\oplus$ is a binary word operation with right identity, then its left-inverse $\oplus_l^{-1}$ is also a binary operation with right identity.

Define now the binary word operation $w \bowtie_l^{-1} v = \text{mi}(\text{mi}(v) \bowtie_r^{-1} \text{mi}(w))$. The operation $\bowtie_l^{-1}$ is the left-inverse of $\bowtie$, and this can be proven similarly to Proposition 3.9 by invoking the mirror image operation. Because all the families of languages in the Chomsky hierarchy are closed under

mirror image, their closure properties under the left-inverse of blending are the same as their closure properties under the right-inverse of blending. Lastly, similar to Proposition 3.6, we can also prove that the left-inverse of a binary operation is unique and, as a consequence, the operation $\bowtie_l^{-1}$ is the unique left-inverse of the blending operation.

We now consider the iterated blending operation $\bowtie_*$. Recall that, as mentioned in Section 2, for any language $L \subseteq \Sigma^*$, the language $L^{\bowtie_*}$ can be generated by a splicing system with null-context splicing rules. This connection, together with Proposition 3.2, allows us to express iterated blending using so-called simple splicing systems [25].

Definition 3.15. ([20])
Let $\sigma = (\Sigma, R)$ be a splicing scheme, where Σ is the alphabet and R is a set of rules $R \subseteq \Sigma^*\#\Sigma^*\$\Sigma^*\#\Sigma^*$, where $\#, \$ \notin \Sigma$. A rule $(u_1, u_2; u_3, u_4)$ is a word $u_1\#u_2\$u_3\#u_4 \in R$. For two strings $x, y \in \Sigma^*$, we have that

$$\sigma(x, y) = \{x_1u_1u_4y_2 \mid x = x_1u_1u_2x_2, y = y_1u_3u_4y_2; x_1, x_2, y_1, y_2 \in \Sigma^*, u_1\#u_2\$u_3\#u_4 \in R\}.$$

For a language L, we define $\sigma(L) = L \cup \bigcup_{x,y\in L} \sigma(x, y)$ and we define the iterated splicing of L by $\sigma^*(L) = \bigcup_{i\geq 0} \sigma^i(L)$ with $\sigma^0(L) = L$ and $\sigma^{i+1}(L) = \sigma(\sigma^i(L))$, $i \geq 0$.

Simple splicing schemes are splicing schemes as above, but they are restricted to rules of the form $(a, \lambda; a, \lambda)$ for $a \in \Sigma$. Note that for two languages L_1 and L_2 over Σ, we now have that

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

where $\sigma_{\bowtie}$ is the simple splicing scheme $\sigma_{\bowtie} = (\Sigma, R)$ with $R = \{a\#\lambda\$a\#\lambda \mid a \in \Sigma\}$. Note that in a simple splicing scheme that expresses word blending, each and every letter of the alphabet must be used in a splicing rule. This observation, together with Proposition 3.2 which showed that the blending of two languages can be written as $L_1 \bowtie L_2 = \bigcup_{a\in\Sigma}(L_1 \div_r (a\Sigma^*))a(L_2 \div_l (\Sigma^*a))$, gives us the following result.

Proposition 3.16. For any language $L \subseteq \Sigma^*$, we have $\sigma_{\bowtie}(L) = L \cup (L \bowtie L)$ and $\sigma^*_{\bowtie}(L) = L^{\bowtie_*}$.

We note that the splicing scheme $\sigma_{\bowtie}$ is finite, since the number of rules depends only on the number of symbols in Σ, and is unary (its rules use words of length at most 1). We also note that, even though in [25] consideration is restricted to the case when L is a finite language, the properties of the splicing systems obtained therein imply the following closure properties.

Proposition 3.17. The families of regular, context-free and recursively enumerable languages are closed under iterated blending.

Proof:
Recall that $L^{\bowtie_*} = \sigma^*_{\bowtie}(L)$ and that $\sigma_{\bowtie}$ is finite and unary. For a splicing rule $u_1\#u_2\$u_3\#u_4$, the words u_1 and u_4 are called visible sites and u_2 and u_3 are called invisible sites. In [19], it is shown that full AFLs are closed under regular splicing systems with finitely many visible sites. Since $\sigma_{\bowtie}$ is

finite, the rules of $\sigma_{\bowtie}$ contain only finitely many visible sites. Since the families of regular, context-free and recursively enumerable are all full AFLs [28], we have that they are closed under iterated blending. □

It was shown in [25] that the class of languages generated by simple splicing systems is a subclass of the class of strictly locally testable languages, which is a subregular language class. The authors also showed that the class of languages generated by simple splicing systems contains the class of finite languages. This is done by constructing a simple splicing system with a finite language L as the initial language and introducing a new alphabet symbol to be used with a splicing rule. The effect is that no splicing can be performed in this system, since no words in L contain the new symbol, and therefore no new words are generated. This approach does not work in the case of iterated blending because we cannot restrict the use of blending, which must occur whenever two words have a one-letter overlap. The following example shows that, even though iterated blending is related to simple splicing, there are differences between the two: In contrast to the case of simple splicing (whereby every finite language can be generated by a simple splicing scheme), there exist finite languages that cannot be generated via iterated blending.

Example 3.18. Let $L = \{aa\}$. We will show that L cannot be generated via iterated blending. Suppose that there exists a language B such that $B^{\bowtie_*} = L$. Then there exist words $u, v \in B$ such that $aa \in u \bowtie v$. This means either $u = aau'$ and $v = v'a$ for some $u', v' \in \Sigma^*$ or $u = au'$ and $v = v'aa$ for some $u', v' \in \Sigma^*$. In either case, together with $aa \in B^{\bowtie_*}$, we have $a^+ \subseteq B^{\bowtie_*} = L$, a contradiction.

4. Decision problems

This section investigates some decision problems related to the blending operation, such as the existence of solutions to language equations of the type $X \bowtie L = R$ and $L \bowtie Y = R$ (where L, R are given known languages and X, Y are unknown languages), the closure of languages under $\bowtie_*$, and the existence of a (finite) language base for a given language. We also show that, for a given alphabet, there exist finitely many languages that can be obtained by iterated blending from an initial language.

Proposition 4.1. The existence of a solution Y to the equation $L \bowtie Y = R$ is decidable for given regular languages L and R.

Proof:
According to [29], since $\bowtie_r^{-1}$ is the right-inverse of word blending, if there exists a solution Y to the given equation, then $Y' = (L \bowtie_r^{-1} R^c)^c$ is also a solution. Moreover, in this case Y' is the maximal solution, in the sense that it includes all the other solutions to the equation. Since the family of regular languages is closed under $\bowtie_r^{-1}$ and complement, the algorithm for deciding the existence of a solution starts with constructing $L \bowtie Y'$, which is also regular, and checking whether $L \bowtie Y'$ equals R. As equality of regular languages is decidable [31], if the answer to the question "Is $L \bowtie Y'$ equal to R?"

is "yes", then a solution to the equation exists, and Y' is such a solution. If the answer is "no", then the equation has no solutions. □

We note that the solution to a language equation of the type $L \bowtie Y = R$ need not be unique.

Example 4.2. Let $\Sigma = \{a, b\}$. The equation $\{a^n b^n | n \geq 0\} \bowtie Y = a^+ b^+$, has the maximal solution $Y_{max} = a^* b^+ \cup \{\lambda\}$, but it also has the solution $Y = b^+ \subseteq Y_{max}$.

As another example, the equation $\{a^n b^n | n \geq 0\} \bowtie Y = a^+$ has the maximal solution $Y_{max} = (\{a^n b^n | n \geq 0\} \bowtie_r^{-1} (a^+)^c)^c = (\{a^n b^n | n \geq 0\} \bowtie_r^{-1} (\{\lambda\} \cup \Sigma^* b \Sigma^*))^c = (\{a^n b^n | n \geq 0\} \bowtie_r^{-1} \Sigma^* b \Sigma^*)^c = (\Sigma^* b \Sigma^*)^c = a^*$, but it also has the solution $Y = a^+ \subseteq Y_{max}$.

Proposition 4.3. The existence of a singleton solution $\{w\}$ to the equation $L \bowtie \{w\} = R$ is decidable for regular languages L and R.

Proof:
If R is empty, a singleton solution $\{w\}$ to the equation $L \bowtie \{w\} = R$ exists iff L does not use all the letters from the alphabet Σ. The decision algorithm will check the emptiness of all regular languages $L \cap \Sigma^* a \Sigma^*$, where $a \in \Sigma$: If any of them is empty, then $\{w\} = \{a\}$ is a singleton solution, otherwise no singleton solutions exist.

If R is not empty and $\lambda \in R$, then there are no singleton solutions by the definition of blending. We now consider the case when R is not empty and $\lambda \notin R$. If there is a singleton solution $\{w\}$ to the equation $L \bowtie \{w\} = R$, where $L \subseteq \Sigma^*$, $R \subseteq \Sigma^+$, $w \in \Sigma^+$, then there is a shortest singleton solution of length $k \geq 1$, denoted by $w_s = a_1 a_2 \cdots a_k$, with $a_1, a_2, \ldots, a_k \in \Sigma$. We now want to show that the number of states in any DFA that recognizes R is at least k.

If $|w_s| = 1$, then $\lambda \notin R$, so the number of states of any DFA that recognizes R is at least 2, which is greater than the length of w_s.

Suppose $k \geq 2$. Define $L_i = (L \bowtie a_i) a_{i+1} \cdots a_k$ for $1 \leq i < k$, and define $L_k = L \bowtie a_k$. Then, we have $R = \bigcup_{i=1}^{k} L_i$. Note that $L_1 \not\subseteq \bigcup_{i=2}^{k} L_i$, as otherwise $a_2 a_3 \cdots a_k$ would be a shorter singleton solution than w_s, which is a contradiction.

Let $\alpha \in L_1 \subseteq R$; α can be represented as $\alpha = \alpha_1 a_1 a_2 \cdots a_k$, where $\alpha_1 \in \Sigma^*$. Assume now that R is recognized by a DFA $M = (Q, \Sigma, \delta, q_0, F)$ with $n < k$ states. Then there is a derivation

$$q_0 \alpha_1 a_1 a_2 \cdots a_k \Longrightarrow^* q_{i_1} a_1 a_2 \cdots a_k \Longrightarrow q_{i_2} a_2 \cdots a_k \Longrightarrow \cdots \Longrightarrow q_{i_k} a_k \Longrightarrow q_{i_{k+1}}.$$

Because M has $n < k$ states, there is a state that occurs twice in the set $\{q_{i_2}, q_{i_3}, \ldots, q_{i_{k+1}}\}$.

If $q_{i_j} = q_{i_{k+1}}$ where $2 \leq j \leq k$, then $\alpha_1 a_1 \cdots a_{j-1} (a_j \cdots a_k)^+ \subseteq R$, so there exists a word $\alpha_2 \in \Sigma^*$ such that $\alpha_1 a_1 \cdots a_{j-1} (a_j \cdots a_k)^+ \alpha_2 \subseteq L$. Thus, we have

$$\alpha \in \alpha_1 a_1 \cdots a_{j-1} (a_j \cdots a_k)^+ \alpha_2 \bowtie a_k \subseteq L_k \subseteq \bigcup_{i=2}^{k} L_i.$$

If $q_{i_j} = q_{i_h}$ where $2 \leq j < h \leq k$, then $\alpha_1 a_1 \cdots a_{j-1}(a_j \cdots a_{h-1})^+ a_h \cdots a_k \subseteq R$, so there exists a word $\alpha_2 \in \Sigma^*$ such that $\alpha_1 a_1 \cdots a_{j-1}(a_j \cdots a_{h-1})^+ \alpha_2 \subseteq L$. Then

$$\alpha \in \big(\alpha_1 a_1 \cdots a_{j-1}(a_j \cdots a_{h-1})^+ \alpha_2 \bowtie a_{h-1}\big) a_h \cdots a_k \subseteq L_{h-1} \subseteq \bigcup_{i=2}^{k} L_i.$$

In either case, for all words $\alpha \in L_1$, $\alpha \in \bigcup_{i=2}^{k} L_i$. Thus, we have that $L_1 \subseteq \bigcup_{i=2}^{k} L_i$, which is a contradiction.

For the equation $L \bowtie Y = R$, if there is a singleton solution, there is a singleton solution w_s of minimal length k, and the number of states in any DFA recognizing R is at least k. If the language R is given as a DFA with k states, the algorithm for deciding the existence of a singleton solution will check all the words β, where $|\beta| \leq k$. The answer is"yes" if this algorithm finds a string β such that $L \bowtie \{\beta\} = R$, and "no" otherwise. □

Proposition 4.4. The existence of a singleton solution $\{w\}$ to the equation $L \bowtie \{w\} = R$ is undecidable for regular languages R and context-free languages L.

Proof:
Assume, for the sake of contradiction, that the existence of a singleton solution $\{w\}$ to the equation $L \bowtie \{w\} = R$ is decidable for regular languages R and context-free languages L.

Given an arbitrary context-free language L' over an alphabet Σ, the context-free language $L_1 = \#\Sigma^+\# \cup L'\$$ can be constructed where $\#, \$ \notin \Sigma$. Note now that the equation $L_1 \bowtie \{w\} = \Sigma^*\$$ has a singleton solution $\{w\}$ iff $L' = \Sigma^*$ and the solution is $\{w\} = \{\$\}$. Thus, if we could decide the problem in the proposition, we would be able to decide whether or not $L' = \Sigma^*$ for arbitrary context-free languages L', which is impossible [31]. □

Corollary 4.5. The existence of a solution Y to the equation $L \bowtie Y = R$ is undecidable for regular languages R and context-free languages L.

Proposition 4.6. The existence of a (singleton) solution X to the equation $X \bowtie L = R$ is decidable for regular languages L and R, and undecidable for regular languages R and context-free languages L.

Proof:
Similar to those of Proposition 4.1, Proposition 4.3, Proposition 4.4 and Corollary 4.5. □

Next, we consider the decidability of the question of whether a language is closed under iterated blending.

Proposition 4.7. Let L be a regular language. It is decidable whether or not L is closed under $\bowtie_*$.

Proof:
By Proposition 3.16 and [32], we can construct an NFA A' that recognizes $L^{\bowtie_*}$. Testing equivalence of two NFAs is known to be decidable [31] and therefore, testing whether $L = L^{\bowtie_*}$ is decidable. □

Proposition 4.8. It is undecidable whether or not L is closed under $\bowtie_*$ where L is context free.

Proof:
Assume the contrary. Under this assumption, given an arbitrary context-free language L over an alphabet Σ, and a letter $\#$ not in Σ, we claim that the context-free language $B = \#L(\#\Sigma^*)^*$ is closed under $\bowtie_*$ iff $L = \Sigma^*$ or $L = \emptyset$.

It is clear that if $L = \Sigma^*$ or $L = \emptyset$, then B is closed under $\bowtie_*$. Now, we consider the other implication and assume that B is closed under $\bowtie_*$. If $B = \emptyset$ then $L = \emptyset$. If $B \neq \emptyset$ then $L \neq \emptyset$. Consider the languages $\#L$ and $\#L(\#\Sigma^*)^+$, both included in B. As B is closed under $\bowtie_*$, the blending of these two languages should be included in $B^{\bowtie_*}$, that is, $\#L \bowtie \#L(\#\Sigma^*)^+ = (\#\Sigma^*)^+ \subseteq B^{\bowtie_*}$. Thus, $B = (\#\Sigma^*)^+$, and this implies that $L = \Sigma^*$.

Thus, if we could decide the problem in the proposition, since emptiness is decidable for context-free languages [31], we would be able to decide whether or not $L = \Sigma^*$ for arbitrary context-free languages L, which is impossible [31]. □

Let $L, B \subseteq \Sigma^*$ be two languages. We say that B is a base of L (with respect to $\bowtie_*$) if $L = B^{\bowtie_*}$. In [25], it is shown that it is decidable whether or not a regular language is generated by a simple splicing scheme and a finite language base, and we can prove similar results with respect to $\bowtie_*$.

Note that a language L is closed under $\bowtie_*$ iff it has a base with respect to $\bowtie_*$. If L is closed under $\bowtie_*$, L is a base for itself. Otherwise, if L is not closed under $\bowtie_*$, it does not have a base. Indeed, if it had a base B, we would have that $L = B^{\bowtie_*} = \{B^{\bowtie_*}\}^{\bowtie_*} = L^{\bowtie_*}$, which is a contradiction.

A language L is said to be an *iterated blending language* iff it can be generated by iterated blending, that is, iff there exists a base B, such that $L = B^{\bowtie_*}$. Some languages are iterated blending languages and some are not, as shown by the following examples.

Example 4.9. Let $\Sigma = \{0, 1\}$. The language $L_1 = \Sigma^*0$, which consists of the binary representations of all even natural numbers, is an iterated blending language because it can be generated by applying iterated blending to the base $B_1 = L_1$, or to the finite base $B_2 = \{110, 100, 010, 000\}$. On the other hand, the language L_2 consisting of the binary representations of all prime numbers, is not an iterated blending language because it is not closed under $\bowtie_*$.

Propositions 4.7 and 4.8 can now be rephrased as follows: The question of whether or not a language L is an iterated blending language (has a base) is decidable if L is regular, and it is undecidable if L is context free. The next proposition answers the analogous question regarding the existence of a finite base.

Proposition 4.10. The question of whether or not a language L has a finite base with respect to $\bowtie_*$ is decidable if L is regular, and it is undecidable if L is context free.

Proof:
The proof is the same as that of the question of whether or not a language L can be generated by a simple splicing system, which was proven to be decidable if L is regular, and undecidable if L is context free [25]. □

A similar proof idea can also be used to prove the following result, showing that a language has a base iff it has a finite base.

Proposition 4.11. Let $L \subseteq \Sigma^*$ be an arbitrary language. There exists a language $R \subseteq \Sigma^*$ such that $L = R^{\bowtie_*}$ iff $L = B^{\bowtie_*}$, where $B = L \cap \{w \in \Sigma^* \mid |w|_a \leq 2 \text{ for all } a \in \Sigma\}$.

Proof:
It is clear that if $L = B^{\bowtie_*}$, then there exists a language $R \subseteq \Sigma^*$ such that $L = R^{\bowtie_*}$. For the other implication, assume that there exists a language $R \subseteq \Sigma^*$ such that $L = R^{\bowtie_*}$. Since $L = R^{\bowtie_*}$, by definition R is a base for L with respect to $\bowtie_*$ and therefore, L is closed under $\bowtie_*$, and $L^{\bowtie_*} = L = R^{\bowtie_*}$.

Let $w \in L$ such that there exists $a \in \Sigma$ with $|w|_a \geq 3$, and let $S_w^0 = \{w\}$. Since $|w|_a \geq 3$, the word w can be decomposed as $w = w_1 a w_2 a w_3 a w_4$, where $w_1 \in \Sigma^*$, and $w_2, w_3, w_4 \in (\Sigma \setminus \{a\})^*$. Consider the words $w' = w_1 a w_2 a w_4$ and $w'' = w_1 a w_3 a w_4$. It is clear that $w \in w' \bowtie w''$, that $w' \in w \bowtie w \subseteq L$, and that $w'' \in w \bowtie w \subseteq L$.

Consider now the set $S_w^1 = (S_w^0 \setminus \{w\}) \cup \{w', w''\}$. We have that $w \in (S_w^1)^{\bowtie_*}$, $(S_w^1)^{\bowtie_*} \subseteq L^{\bowtie_*}$, and $|w'|_a = |w''|_a = |w|_a - 1$. If there exists a word in S_w^1 wherein the number of occurrences of a is at least 3, then we repeat the process for this word to obtain a new set; otherwise, choose another letter $b \in \Sigma \setminus \{a\}$, and repeat the process. By repeating this process, after a finite number of steps, we obtain a set S_w such that $S_w \subseteq B$, where $B = L \cap \{w \mid |w|_a \leq 2 \text{ for all } a \in \Sigma\}$. Moreover, $w \in S_w^{\bowtie_*}$, $S_w^{\bowtie_*} \subseteq B^{\bowtie_*}$, and $S_w^{\bowtie_*} \subseteq L$. It is clear that

$$\bigcup_{w \in L \setminus B} S_w \cup \{w \in L \mid |w|_a \leq 2 \text{ for all } a \in \Sigma\} = B.$$

We claim that $B^{\bowtie_*} = L$. Indeed, since $B \subseteq L$ and L is closed under $\bowtie_*$, we have that $B^{\bowtie_*} \subseteq L$. For the other inclusion, if $w \in L$ and $|w|_a \leq 2$ for all $a \in \Sigma$, then $w \in B \subseteq B^{\bowtie_*}$. Otherwise, that is, if $w \in L$ but w has at least three occurrences of a for some $a \in \Sigma$, we have that $w \in S_w^{\bowtie_*} \subseteq B^{\bowtie_*}$. This proves that $L \subseteq B^{\bowtie_*}$. □

Corollary 4.12. Given a language $L \subseteq \Sigma^*$, the following are equivalent:

(i) L is closed under iterated blending,
(ii) L is an iterated blending language,
(iii) L has a base R with respect to iterated blending, i.e., such that $R^{\bowtie_*} = L$,
(iv) L has a finite base B with respect to iterated blending, i.e., such that $B^{\bowtie_*} = L$.

Corollary 4.13. For an arbitrary language $R \subseteq \Sigma^*$, $R^{\bowtie_*}$ is regular. All the families of languages in the Chomsky hierarchy are closed under $\bowtie_*$.

Corollary 4.14. Given an alphabet Σ, there are finitely many iterated blending languages over Σ.

Proof:
Consider an alphabet Σ of cardinality n. Any word of length at least $2n+1$ must have a letter repeated at least 3 times, so there are at most $\sum_{i=0}^{2n} n^i$ different words containing each letter at most twice. Therefore, by Proposition 4.11, there are at most $2^{\sum_{i=0}^{2n} n^i}$ different bases, and thus there are finitely many iterated blending languages. □

5. State complexity

By Proposition 3.2, the family of regular languages is closed under blending. Thus, we can consider the state complexity of blending on two regular languages. Recall from Proposition 3.2 that the blending of two languages can be expressed as a series of union, catenation, and quotient operations. While the state complexity of each of these operations is known, the state complexity of a combination of operations is not necessarily the same as the composition of the state complexities of the operations [33].

We will construct a DFA that recognizes the language of the blending of the two languages L_m and L_n recognized respectively by DFAs $A_m = (Q_m, \Sigma, \delta_m, s_m, F_m)$ and $A_n = (Q_n, \Sigma, \delta_n, s_n, F_n)$. We construct a DFA $A' = (Q', \Sigma, \delta', s', F')$ where

- $Q' = Q_m \times 2^{Q_n}$,
- $s' = (s_m, \emptyset)$,
- $F' = \{(q, P) \in Q_m \times 2^{Q_n} \mid P \cap F_n \neq \emptyset\}$,
- $\delta'((q, P), a) = (\delta_m(q, a), P')$ for $a \in \Sigma$, where

$$P' = \begin{cases} \bigcup_{p \in P} \delta_n(p, a) & \text{if } \delta_m(q, a) \text{ is the sink state,} \\ \bigcup_{p \in Q_n} \delta_n(p, a) & \text{otherwise.} \end{cases}$$

Each state of A' is a pair consisting of a state of A_m and a subset of states of A_n. Informally, we can divide the computation of a word into two phases. In the first phase, states of the form (q, P) are reached where q is not the sink state of A_m. Here, the set P is determined solely by the input symbol as the machine tries to guess the symbol on which the blending occurs. In the second phase, the machine reaches states $(q_\emptyset, P)$, where $q_\emptyset$ is the sink state of A_m. The second phase only occurs when the blending occurs and the input that has been read is no longer a prefix of a word recognized by A_m. In this phase, the set P is determined by the transition function of A_n. It follows from this construction that A' recognizes the language $L_m \bowtie L_n$. We give an example of the result of this construction in Figure 1.

A simple count shows that the number of states in the state set of A' is $m \cdot 2^n$. We will show that, depending on the size of the alphabet, not all of these states are necessarily reachable. First, we consider the case where the alphabet is unary.

Proposition 5.1. Let L_m and L_n be regular languages defined over a unary alphabet such that L_m is recognized by an m-state DFA and L_n is recognized by an n-state DFA. Then the state complexity of $L_m \bowtie L_n$ is $m + n - 2$ if both L_m and L_n are finite or 2 otherwise. Furthermore, this bound is reachable.

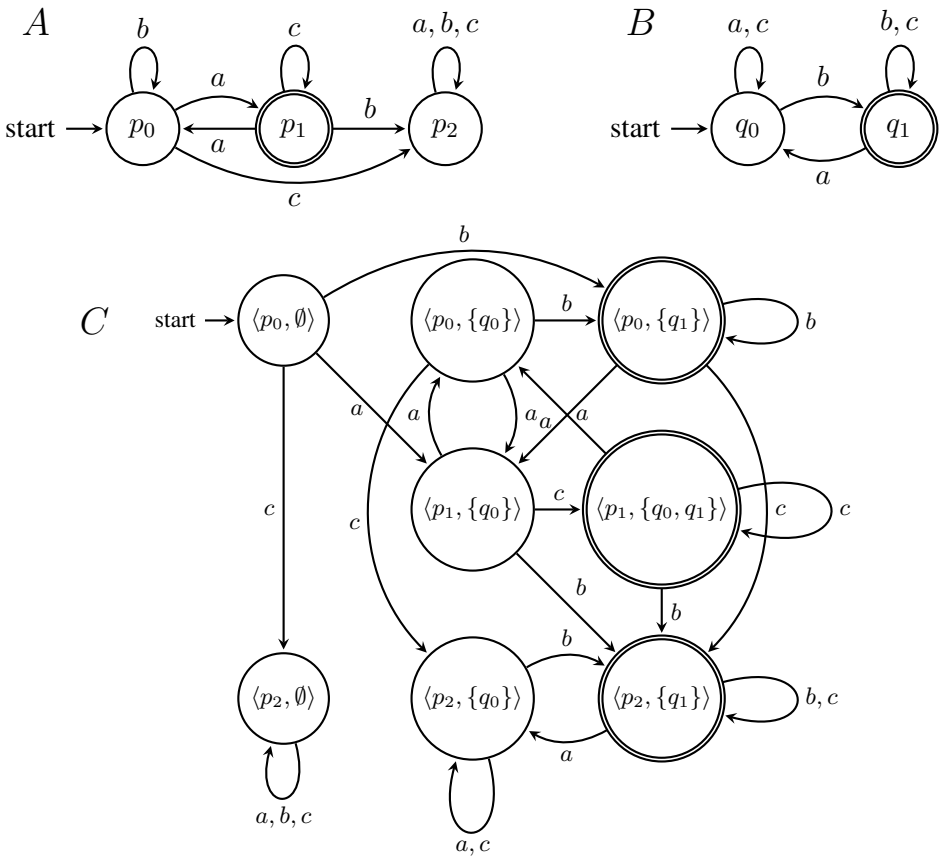

Figure 1. The DFA C is constructed from the DFAs A and B and recognizes the language $L(A) \bowtie L(B)$.

Proof:
Recall that by Proposition 3.2, $L_m \bowtie L_n = (L_m \div_r (a^+))a(L_n \div_l (a^+))$. If either L_m or L_n is infinitely large, then we have $L_m \bowtie L_n = a^+$, in which case the state complexity of $L_m \bowtie L_n$ is 2. If both L_m and L_n are finite, then it is easy to see that the state complexity of $L_m \bowtie L_n$ is $m+n-2$. □

Now, we will consider the state complexity when the languages are defined over alphabets of size greater than 1.

Lemma 5.2. The DFA A' requires at most $(m-1) \cdot (k-1) + 2^n + 1$ states, where $k = |\Sigma| \leq 2^n$.

Proof:
First, observe that in order to maximize the number of reachable states of A', the DFA A_m must contain a state that cannot reach an accepting state. Otherwise, if every state of A_m can reach an accepting state, then by definition of A', we have $L(A_m) \bowtie L(A_n) = L(A') \subseteq \mathrm{pref}(L(A_m))$. One can construct a DFA for $\mathrm{pref}(L(A_m))$ by modifying A_m so that every state of A_m is a final state. In

this case, A' would then require at most m states. Thus, we assume that A_m contains a sink state $q_\emptyset$ which cannot reach an accepting state.

Consider the transition function δ' on a state (q, P), where $q \neq q_\emptyset$. Then for each symbol $a \in \Sigma$, there is only one possible reachable set of states P in A_n. This gives us up to $(m-1)\cdot k$ reachable states. However, we claim that in order for two states (q, P) and (q, P') with $P \neq P'$ to be distinguishable, q must contain a transition to $q_\emptyset$. Otherwise, for every symbol $a \in \Sigma$, we have $\delta'((q, P), a) = \delta'((q, P'), a)$ by definition. Thus, since every state must contain at least one transition to $q_\emptyset$ and A_m is deterministic, A' has only at most $(m-1)\cdot(k-1)$ reachable states of this form.

Next, consider that there are up to 2^n reachable states $(q_\emptyset, P)$ as derived from the subset construction.

Finally, we note that the initial state $s' = (s_m, \emptyset)$ does not belong to any of the above sets. Adding all of these states together, we have at most $(m-1)\cdot(k-1) + 2^n + 1$ reachable states. □

Lemma 5.3. Let $m, n \geq 3$, and $4 \leq k < 2^n$. There exist families of DFAs A_m with m states and B_n with n states defined over an alphabet with k letters such that a DFA recognizing $A_m \bowtie B_n$ requires at least $(m-1)\cdot(k-1) + 2^n + 1$ states.

Proof:
Let $\Sigma = \{a_1, \ldots, a_{k-2}, b, c\}$. We will define the DFAs A_m and B_n over Σ.

Let $A_m = (Q_m, \Sigma, \delta_m, s_m, F_m)$ where $Q_m = \{0, \ldots, m-1\}$, $s_m = 0$, and $F_m = \{m-2\}$. We define the transition function δ_m by

- $\delta_m(p, a_i) = p$ for all $0 \leq p \leq m-2$ and $1 \leq i \leq k-2$,
- $\delta_m(p, b) = p+1$ for $0 \leq p \leq m-2$,
- $\delta_m(p, c) = m-1$ for $0 \leq p \leq m-2$,
- $\delta_m(m-1, a) = m-1$ for all $a \in \Sigma$.

The DFA A_m is shown in Figure 2.

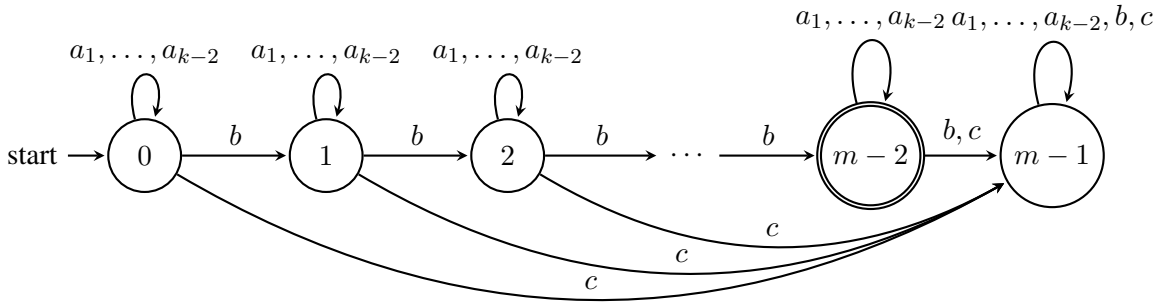

Figure 2. The DFA A_m.

Let $B_n = (Q_n, \Sigma, \delta_n, s_n, F_n)$ where $Q_n = \{0, \ldots, n-1\}$, $s_n = 0$, and $F_n = \{n-1\}$. We will define the transition function δ_n by

- $\delta_n(q, b) = q + 1 \bmod n$ for $0 \leq q \leq n-1$,
- $\delta_n(q, c) = q$ for $0 \leq q \leq n-1$.

For transitions on symbols a_i with $1 \leq i \leq k-2$, we define an enumeration of the subsets of Q_n and let $Q_n[i]$ be the i-th subset of Q_n. Any arbitrary enumeration of subsets of Q_n suffices for this proof subject to the condition that

1. for $0 \leq i \leq k-2$, each i corresponds to a distinct subset of Q_n (that is, $Q_n[i] \neq Q_n[j]$ iff $i \neq j$ for $0 \leq i, j \leq k-2$), and
2. we reserve the following: $Q_n[0] = Q_n$, $Q_n[1] = \{0, 1, \ldots, n-2\}$, $Q_n[2] = \{0\}$.

Also note that while we have defined $Q_n[0]$, there are no symbols a_0. We will show later that, by our definitions, the role of a_0 will be played by b. If $k > 2^n$, then this property cannot hold but it is clear that we can enumerate all 2^n subsets of Q_n.

Then we define transitions on $a_i \in \Sigma$ by

$$\delta_n(q, a_i) = \begin{cases} q & \text{if } q \in Q_n[i], \\ (q + \min_{(q+j \bmod n) \in Q_n[i]} j) \bmod n & \text{if } q \notin Q_n[i]. \end{cases}$$

In other words, for each state $q \in Q_n$, the transition on the symbol a_i goes to the "next" state that is in $Q_n[i]$. If $q \in Q_n[i]$, then that q itself is the "next" state.

The DFA B_3 is shown in Figure 3, with $Q_3[i]$ defined for $0 \leq i \leq 6$ as follows:

$$\begin{aligned} &Q_3[0] = \{0, 1, 2\} \\ &Q_3[1] = \{0, 1\} \qquad Q_3[2] = \{0\} \qquad Q_3[3] = \{1\} \\ &Q_3[4] = \{2\} \qquad Q_3[5] = \{0, 2\} \qquad Q_3[6] = \{1, 2\} \end{aligned}$$

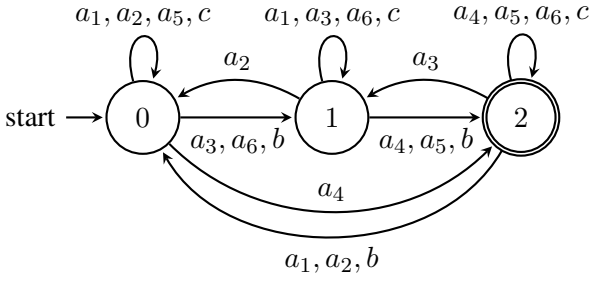

Figure 3. The DFA B_3.

We will show that A' contains $(m-1) \cdot (k-1) + 2^n + 1$ reachable and distinguishable states.

First, to show that the states are reachable, we note that $s' = (s_m, \emptyset)$ is clearly reachable as the initial state. Then, we observe that for $1 \leq i \leq k-2$, the state $(q, Q_n[i])$ with $q \in Q_m \setminus \{m-1\}$

is reachable on the word $b^q a_i$ and $(q, Q_n[0])$ is reachable on the word b^q. Since the only symbol not used here is c, this gives us $(m-1)\cdot(k-1)$ states.

Now we consider states of the form $(m-1, P)$, where $P \subseteq Q_n$. Observe that $(m-1, Q_n)$ can be reached on the word b^{m-1}. Also, note that $(m-1, \emptyset)$ can be reached on the letter c from $(0, \emptyset)$.

Next, we will show that all states of the form $(m-1, P)$, where $P = Q_n \setminus T$ for some $T \subseteq Q_n$ are reachable from $(m-1, Q_n)$ by induction on $|T|$. First, consider $|T| = 1, T = \{t\}, 0 \leq t \leq n-1$. Then, we have $(m-1, Q_n) \xrightarrow{a_1 b^{t+1}} (m-1, Q_n \setminus \{t\})$.

Assume that all states $(m-1, Q_n \setminus T')$ are reachable from $(m-1, Q_n)$, where $u = |T'| \geq 1$. We will show that all states $(m-1, Q_n \setminus T)$ are reachable from $(m-1, Q_n)$, where $|T| = u+1 < |Q_n|$. Let $P = Q_n \setminus T = \{t_1, t_2, \ldots, t_{\ell-1}\}$, where elements in P are ordered in the ascending order. If $t_1 = 0$, $t_{\ell-1} \neq n-1$, then we have

$$(m-1, \{0, t_2, \cdots, t_{\ell-1}, n-1\}) \xrightarrow{a_1} (m-1, \{0, t_2, \ldots, t_{\ell-1}\}) = (m-1, P). \tag{1}$$

Thus, $(m-1, P)$ is reachable from the state $(m-1, P \cup \{n-1\})$, which is reachable from $(m-1, Q_n)$ by assumption.

If $t_1 = 0, t_{\ell-1} = n-1$, there exists a largest integer $v \notin P$, where $1 \leq v < n-1$, then we have

$$(m-1, \{w + n - v - 1 \bmod n \mid w \in P\}) \xrightarrow{b^{v+1}} (m-1, P).$$

Thus, $(m-1, P)$ is reachable from $(m-1, Q_n)$ by (1).

If $t_1 > 0$, then we have

$$(m-1, \{0, t_2 - t_1, \cdots, t_{\ell-1} - t_1\}) \xrightarrow{b^{t_1}} (m-1, \{t_1, t_2, \ldots, t_{\ell-1}\}).$$

That is, $(m-1, P)$ is reachable from $(m-1, Q_n)$ by (1). Thus, all states $(m-1, Q_n \setminus T)$ with $|T| = u+1$ are reachable.

Thus, we have an additional 2^n reachable states of the form $(m-1, P)$, giving us a total of $(m-1)\cdot(k-1) + 2^n + 1$ reachable states.

Next, we will show that these states are pairwise distinguishable. Consider two states (q, P) and (q', P'). First, we consider when $P \neq P'$. In this case, reading c takes the state (q, P) to $(m-1, P)$ and (q', P') to $(m-1, P')$. Then without loss of generality, there exists an element $t \in P$ such that $t \notin P'$. Then these states are distinguished by the word b^{n-1-t}.

Now, fix $P = P'$ and assume without loss of generality that $q > q'$. First, suppose $q < m-1$. Then we have

$$(q, P) \xrightarrow{b^{m-1-q}} (m-1, Q_n[0]) \xrightarrow{a_2} (m-1, Q_n[2]) \xrightarrow{a_1} (m-1, \{0\}).$$

Recall that we had defined $Q_n[2] = \{0\}$. Now, since $q > q'$, we have $m-1-q+q' < m-1$. Let $q'' = m-1-q+q'$, and we have

$$(q', P') \xrightarrow{b^{m-1-q}} (q'', Q_n[0]) \xrightarrow{a_2} (q'', Q_n[2]) \xrightarrow{a_1} (q'', Q_n[1]).$$

Since $Q_n[1] \neq \{0\}$, the two states are now distinguishable by the prior case.

Now, suppose $q = m - 1$. Then we have

$$(q, P) = (m-1, P) \xrightarrow{a_2} (m-1, Q_n[2]) = (m-1, \{0\}) \xrightarrow{a_1} (m-1, \{0\})$$

and

$$(q', P') = (q', P) \xrightarrow{a_2} (q', Q_n[2]) \xrightarrow{a_1} (q', Q_n[1]).$$

Again, since $Q_n[1] \neq \{0\}$, the two states are now distinguishable by the prior case.

Thus, we have shown that all $(m-1) \cdot (k-1) + 2^n + 1$ states are reachable and pairwise distinguishable. □

These results together give us the following theorem.

Theorem 5.4. Let A_m be a DFA with m states recognizing the language L_m and let A_n be a DFA with n states recognizing the language L_n, where L_m and L_n are defined over an alphabet Σ of size k, and $m, n \geq 3$. Then the state complexity of $L_m \bowtie L_n$ is $(m-1) \cdot (k-1) + 2^n + 1$ if $4 \leq k < 2^n$, and $(m-1) \cdot (2^n - 1) + 2^n + 1$, if $k \geq 2^n$, and this bound can be reached in the worst case.

6. Conclusions

We have defined a word blending operation as a generalization of an experimentally observed outcome of XPCR on DNA strings. Technically, word blending is related to the binary word operation Latin product, the crossover operation, and simple splicing systems. We studied closure properties, decision problems, and the state complexity for this operation, as well as showed that given an alphabet, there are finitely many languages that can be the result of iterated word blending applied to an initial language.

We now mention a few directions for further research. The first is to consider a more realistic definition for word blending. Indeed, the current definition is a generalization of the observed experimental outcome, and various properties may be different if we impose the experimental restriction $\gamma_1 = \gamma_2$ (for example, Lemma 3.1 no longer holds). Another direction is to consider the complexity of the decision problems studied in Section 4, to find the exact number of different iterated blending languages in Corollary 4.14, and to characterize the class of languages generated by iterated blending. Finally, the question of determining the state complexity of iterated word blending remains open. Possible approaches could potentially make use of the connections between iterated blending and simple splicing systems.

Acknowledgements

We thank Giuditta Franco for fruitful discussions on modelling the outcomes of various XPCR experiments. We also thank the anonymous referees whose detailed reviews and comments lead to an expansion of the paper as well as improvements in the presentation.

References

[1] Franco G, Giagulli C, Laudanna C, Manca V. DNA extraction by XPCR. In: Ferretti C, Mauri G, Zandron C (eds.), Proc. DNA Computing, (DNA 10), volume 3384 of *LNCS*. Springer. 2005 pp. 104–112. doi:10.1007/11493785_9.

[2] Franco G, Manca V, Giagulli C, Laudanna C. DNA recombination by XPCR. In: Carbone A, Pierce NA (eds.), Proc. DNA Computing, (DNA 11), volume 3892 of *LNCS*. Springer, 2006 pp. 55–66. doi:10.1007/11753681_5.

[3] Franco G. A Polymerase Based Algorithm for SAT. In: Coppo M, Lodi E, Pinna GM (eds.), Proc. Italian Conference on Theoretical Computer Science, (ICTCS 2005), volume 3701 of *LNCS*. Springer, 2005 pp. 237–250. doi:10.1007/11560586_20.

[4] Franco G, Manca V. Algorithmic applications of XPCR. *Natural Computing*, 2011. **10**(2):805–819. doi:10.1007/s11047-010-9199-8.

[5] Manca V, Franco G. Computing by polymerase chain reaction. *Mathematical Biosciences*, 2008. **211**(2):282–298. doi:10.1016/j.mbs.2007.08.010.

[6] Franco G, Bellamoli F, Lampis S. Experimental Analysis of XPCR-based protocols. 2017. arXiv preprint arXiv:1712.05182.

[7] Csuhaj-Varju E, Petre I, Vaszil G. Self-assembly of strings and languages. *Theoretical Computer Science*, 2007. **374**(1):74–81. URL `https://doi.org/10.1016/j.tcs.2006.12.004`.

[8] Brzozowski JA, Kari L, Li B, Szykuła M. State Complexity of Overlap Assembly. In: Câmpeanu C (ed.), Proc. Implementation and Application of Automata, (CIAA 2018), volume 10977 of *LNCS*. Springer, 2018 pp. 109–120. doi:10.1007/978-3-319-94812-6_10.

[9] Enaganti SK, Ibarra OH, Kari L, Kopecki S. On the overlap assembly of strings and languages. *Natural Computing*, 2017. **16**(1):175–185. doi:10.1007/s11047-015-9538-x.

[10] Enaganti SK, Ibarra OH, Kari L, Kopecki S. Further remarks on DNA overlap assembly. *Information and Computation*, 2017. **253**(1):143–154. URL `https://doi.org/10.1016/j.ic.2017.01.009`.

[11] Holzer M, Jakobi S. Chop Operations and Expressions: Descriptional Complexity Considerations. In: Mauri G, Leporati A (eds.), Proc. Developments in Language Theory, (DLT 2011), volume 6795 of *LNCS*. Springer, 2011 pp. 264–275. doi:10.1007/978-3-642-22321-1_23.

[12] Holzer M, Jakobi S. State Complexity of Chop Operations on Unary and Finite Languages. In: Kutrib M, Moreira N, Reis R (eds.), Proc. Descriptional Complexity of Formal Systems, (DCFS 2012), volume 7386 of *LNCS*. Springer, 2012 pp. 169–182. doi:10.1007/978-3-642-31623-4_13.

[13] Holzer M, Jakobi S, Kutrib M. The Chop of Languages. *Theoretical Computer Science*, 2017. **682**:122–137. URL `https://doi.org/10.1016/j.tcs.2017.02.002`.

[14] Carausu A, Păun G. String intersection and short concatenation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 1981. **26**(5):713–726.

[15] Golan JS. The Theory of Semirings with Applications in Mathematics and Theoretical Computer Science. Addison-Wesley Longman Ltd., 1992. ISBN: 0582078555, 9780582078550.

[16] Ceterchi R. An algebraic characterization of semi-simple splicing. *Fundamenta Informaticae*, 2006. **72**(1–2):19–25.

[17] Ito M, Lischke G. Generalized periodicity and primitivity for words. *Mathematical Logic Quarterly*, 2007. **53**(1):91–106. URL https://doi.org/10.1002/malq.200610030.

[18] Domaratzki M. Minimality in template-guided recombination. *Information and Computation*, 2009. **207**(11):1209–1220. URL https://doi.org/10.1016/j.ic.2009.02.009.

[19] Pixton D. Splicing in abstract families of languages. *Theoretical Computer Science*, 2000. **234**(1):135–166. doi:10.1016/S0304-3975(98)00046-2.

[20] Păun G. On the splicing operation. *Discrete Applied Mathematics*, 1996. **70**(1):57–79. URL https://doi.org/10.1016/0166-218X(96)00101-1.

[21] Bonizzoni P, De Felice C, Zizza R. The structure of reflexive regular splicing languages via Schützenberger constants. *Theoretical Computer Science*, 2005. **334**(1):71–98. doi:10.1016/ j.tcs.2004.12.033.

[22] Goode E, Pixton D. Recognizing splicing languages: Syntactic monoids and simultaneous pumping. *Discrete Applied Mathematics*, 2007. **155**(8):989–1006. URL https://doi.org/10.1016/j.dam.2006.10.006.

[23] Enaganti SK, Kari L, Ng T, Wang Z. Word Blending in Formal Languages: The Brangelina Effect. In: Stepney S, Verlan S (eds.), Proc. Unconventional Computation and Natural Computation, (UCNC 2018), volume 10867 of *LNCS*. Springer, 2018 pp. 72–85. ISBN-10:0126157505, 13:978-0126157505. doi:10.1007/978-3-319-92435-9_6.

[24] Gries ST. Shouldn't it be breakfunch? A quantitative analysis of blend structure in English. *Linguistics*, 2004. **42**(3):639–667. doi:10.1515/ling.2004.021.

[25] Mateescu A, Păun G, Rozenberg G, Salomaa A. Simple splicing systems. *Discrete Applied Mathematics*, 1998. **84**(1–3):145–163. URL https://doi.org/10.1016/S0166-218X(98)00002-X.

[26] Head T. Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 1987. **49**(6):737–759. doi:10.1007/BF02481771.

[27] Daley M, Ibarra OH, Kari L. Closure and decidability properties of some language classes with respect to ciliate bio-operations. *Theoretical Computer Science*, 2003. **306**(1–3):19–38. doi:10.1016/S0304-3975(03)00139-7.

[28] Salomaa A. Formal Languages. Academic Press Professional, Inc., 1977.

[29] Kari L. On language equations with invertible operations. *Theoretical Computer Science*, 1994. **132**(1–2):129–150. URL https://doi.org/10.1016/0304-3975(94)90230-5.

[30] Kari L. On insertion and deletion in formal languages. Ph.D. thesis, University of Turku, 1993.

[31] Hopcroft JE, Motwani R, Ullman JD. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Longman Publishing Co., Inc., 3 edition, 2006. ISBN: 10:0321455363, 13:978-0321455369.

[32] Pixton D. Regularity of Splicing Languages. *Discrete Applied Mathematics*, 1996. **69**(1–2):101–124. URL https://doi.org/10.1016/0166-218X(95)00079-7.

[33] Salomaa K, Yu S. On the state complexity of combined operations and their estimation. *International Journal of Foundations of Computer Science*, 2007. **18**(4):683–698. URL https://doi.org/10.1142/S0129054107004917.