Word Blending in Formal Languages: The Brangelina Effect

Srujan Kumar Enaganti[1], Lila Kari[2], Timothy Ng[2(✉)], and Zihao Wang[1]

[1] Department of Computer Science, The University of Western Ontario, London, ON N6A 3K7, Canada
srujankumar@gmail.com, zwang688@uwo.ca
[2] School of Computer Science, University of Waterloo, Waterloo, ON N2L 3GL, Canada
{lila,tim.ng}@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 xwy_1 and y_2wz that share a non-empty overlap w, results in xwz. We study closure properties of families in the Chomsky hierarchy under word blending, language equations involving this operation, and its descriptional state complexity when applied to regular languages. 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*).

1 Introduction

Cross-pairing Polymerase Chain Reaction (XPCR) is an experimental DNA protocol introduced in [11] 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 [13], for the creation of the solution space for a SAT problem [9], and for mutagenesis [12]. The combinatorial power of such a technique has been explained by logical-symbolic schemes in [23], while algorithms to create combinatorial libraries were improved and experimented in [10,12].

The formal language operation called *overlap assembly*, introduced in [5] under the name of self-assembly, and further investigated in [3,7,8], 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 [18,19], and generalized to an arbitrary length overlap in [20]. Other similar operations have been studied in the literature, such as the "short concatenation" [4], which uses only the maximum-length (possibly empty)

S. Stepney and S. Verlan (Eds.): UCNC 2018, LNCS 10867, pp. 72–85, 2018.
https://doi.org/10.1007/978-3-319-92435-9_6

overlap y between operands, the "Latin product" of words [14] where the overlap consists of only one letter, and the operation $\bigotimes$ which imposes the restriction that the non-overlapping part x is not empty [21]. 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 [6] 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 [26,27]. 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 [2,15].

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 [10]. 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 used). 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 two strings $\alpha A\gamma$ and $\gamma A\beta$, instead of obtaining the expected $\alpha A\gamma A\beta$, the experiments repeatedly produced the result $\alpha A\beta$.

In this paper, we define and investigate a formal language operation called *word blending*, that formalizes 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. Interestingly, this phenomenon has been observed independently in linguistics [16], 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 + po*odle*), emoticon (*emoti*on + *icon*), and Brangelina (*Bra*d + *Angelina*).

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 the decidability of existence of solutions to some language equations involving word blending, and Sect. 5 studies the descriptional state complexity of this operation when applied to regular languages.

2 Preliminaries

An alphabet Σ is a finite non-empty set of symbols. Σ^* denotes 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 $lg(w)$. For words $w, x, y, z \in \Sigma^*$ such that $w = xyz$ we call the subwords x, y, and z *prefix*, *infix*, and *suffix* of w, respectively. The sets $\text{pref}(w)$, $\text{inf}(w)$, and $\text{suff}(w)$ contain, respectively, all prefixes, infixes, and suffixes of w. This notation is extended to languages as $\text{suff}(L) = \bigcup_{w\in L} \text{suff}(w)$. The mirror image of a word $w \in \Sigma^*$ is defined as $\text{mi}(\lambda) = \lambda$, and $\text{mi}(w) = a_k \dots a_2 a_1$ if $w = a_1 a_2 \dots a_k$. The definition is extended to languages in the natural way, by $\text{mi}(L) = \bigcup_{w\in L} \text{mi}(w)$.

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 $L_1 L_2^{-1} = \{u \in \Sigma^* | \exists uv \in L_1, v \in L_2\}$, and the left quotient of L_1 by L_2 is defined as $L_2^{-1} L_1 = \{v \in \Sigma^* | \exists uv \in L_1, u \in L_2\}$.

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 [10] 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 a 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, given by $\alpha A \gamma + \gamma D \beta = \alpha A \gamma D \beta$).

Generalizing this experimentally newly-observed phenomenon to the case where the end words of the input strings are different, we model this string recombination as follows. Given two non-empty words x, y over an alphabet Σ, we define the *word blending*, or simply *blending*, of x with y as

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

The definition of blending can be extended to languages L_1 and L_2 by

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

Note that, for a realistic model, we would need additional restrictions such as the fact that the w, γ_1 and γ_2 should be of a sufficient length and should not appear as a substring in the other strings involved.

We can also extend the blending operation to an iterated version on a language. Let $L \subseteq \Sigma^*$ be a language. We define the *iterated (word) blending* of L by $L^{\bowtie_0} = L$ and $L^{\bowtie_i} = L \bowtie L^{\bowtie_{i-1}}$. We define the iterated blending closure of L by

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

We observe that the result of the iterated blending operation can be generated by a splicing system with null context splicing rules [17]. Splicing rules in [17] are of the form $(u_1, z, u_2; u_3, z, u_4)$. For such a rule, if we have strings $x = x_1 u_1 z u_2 x_2$ and $y = y_1 u_3 z u_4 y_2$, we obtain the word $x_1 u_1 z u_4 y_2$. A splicing rule is a null context rule when $u_1, u_2, u_3, u_4 = \lambda$. It is easy to see that the language $L^{\bowtie_*}$ can be generated from a splicing scheme with rules of the form $(\lambda, w, \lambda; \lambda, w, \lambda)$ for every word $w \in \Sigma^+$. The relationship between iterated blending and splicing will be discussed in greater detail in Sect. 3.

3 Closure Properties

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

The following lemma shows that word-blending is equivalent to a restricted version where only one-letter overlaps are utilized.

Lemma 1. *If x, y are non-empty words in Σ^+, then*

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

This result can be extended to languages in the natural way. Then from this lemma, we can show that the word blending of two languages can be obtained by combining the right quotient, catenation, left quotient and union operations, as follows.

Proposition 2. *Given languages $L_1, L_2 \subseteq \Sigma^+$,*

$$L_1 \bowtie L_2 = \bigcup_{a \in \Sigma} \left(L_1 (a\Sigma^*)^{-1}\right) a \left((\Sigma^* a)^{-1_l} L_2\right).$$

Corollary 3. *Every full AFL is closed under word blending.*

We note that the families of regular languages, context-free languages and recursively enumerable languages are all full AFLs [28].

Proposition 4. *The family of context-sensitive languages is not closed under word 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 consists of words of the form Pba^i where $i \geqslant 0$ and $P \in L_0$ and, in addition, for every $P \in L_0$ there is an $i \geqslant 0$ such that $Pba^i \in L_1$ (see, e.g., [28]).

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

Recall that, given a binary word operation $\diamond$, the binary word operation $\square$ is called the *right-inverse of* $\diamond$ [22] if and only if for every triplet of words $u, y, w \in \Sigma^*$ the following relation holds: $w \in (u \diamond y)$ if and only if $y \in (u \square w)$. In other words, the operation $\square$ is called the right-inverse of $\diamond$ if it can be used to recover the right operand y in $u \diamond y$, from the other operand u and a word $w \in (u \diamond y)$ in the result. Define now the binary word operation $\bowtie^r$ as $u \bowtie^r w = \bigcup_{a \in \Sigma} \Sigma^* a \left(\left(u(a\Sigma^*)^{-1} a\right)^{-1_l} w\right)$. Informally, given a word $w = \alpha a \beta \in (\alpha a \gamma_1 \bowtie \gamma_2 a \beta)$, the operation $\bowtie^r$ outputs the right operand $y = \gamma_2 a \beta$ of word blending, if it is given as inputs the result $w = \alpha a \beta \in (u \bowtie y)$ and the left operand $u = \alpha a \gamma_1$. The definition of $\bowtie^r$ can be extended to languages naturally.

Proposition 5. *The operation $\bowtie^r$ 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 1. Then, we have that $y = \gamma_2 b \beta \in \Sigma^* b \beta = \Sigma^* b \left((\alpha b)^{-1_l} (\alpha b \beta)\right) \subseteq \Sigma^* b \left(\left(((\alpha b \gamma_1)(b \Sigma^*)^{-1}) b\right)^{-1_l} (\alpha b \beta)\right) \subseteq \bigcup_{a \in \Sigma} \Sigma^* a \left(\left(((\alpha b \gamma_1)(a \Sigma^*)^{-1}) a\right)^{-1_l} (\alpha b \beta)\right)$.

If $y \in u \bowtie^r w = \bigcup_{a \in \Sigma} \Sigma^* a \left(\left((u(a\Sigma^*)^{-1})a\right)^{-1_l} w\right)$, then there exist $b \in \Sigma$, and $\gamma_2 \in \Sigma^*, \gamma_3 \in (u(b\Sigma^*)^{-1})b$ such that $y = \gamma_2 b(\gamma_3^{-1_l} w)$. This implies that $w \in \left(u(b\Sigma^*)^{-1}\right) b(\gamma_3^{-1_l} w) = \left(u(b\Sigma^*)^{-1}\right) b\left((\gamma_2 b)^{-1_l} (\gamma_2 b(\gamma_3^{-1_l} w))\right)$ which is included in $\left(u(b\Sigma^*)^{-1}\right) b\left((\Sigma^* b)^{-1_l} y\right) \subseteq \bigcup_{a \in \Sigma} \left(u(a\Sigma^*)^{-1}\right) a\left((\Sigma^* a)^{-1_l} y\right) = u \bowtie y$. □

Corollary 6. *The families of regular languages and recursively enumerable languages are closed under the right inverse of the blending. Moreover, if L_1 is an arbitrary language and L_2 is a regular language, then $L_1 \bowtie^r L_2$ is regular; if L_1 is a regular language and L_2 is a context-free language, then $L_1 \bowtie^r L_2$ is context-free.*

Proposition 7. *The family of context-free languages is not closed under the right inverse of blending.*

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

We now show that $(L_1 \bowtie^r L_2) \cap R = \{\$c^{2^n} \mid n \geqslant 2\}$. Since words in R start with \$ and contain only one symbol \$, the only cases in which the words in $L_1 \bowtie^r L_2$ 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 $w \in (L_1 \bowtie^r L_2)$ 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 $(L_1 \bowtie^r L_2) \cap R = \{\$c^{2^n} \mid n \geqslant 2\}$.

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

Recall that given a binary word operation $\diamond$, the binary word operation $\square$ is called the *left-inverse of* $\diamond$ iff for every triplet of words $x, v, w \in \Sigma^*$ the following relation holds: $w \in (x \diamond v)$ if and only if $x \in (w \square v)$ [22].

Proposition 9. *The left inverse of blending can be expressed using the right inverse of blending, and mirror image as* $w \bowtie^l v = mi(mi(v) \bowtie^r mi(w))$.

Because all families of languages in the Chomsky hierarchy are closed under mirror image, their closure properties under the left-inverse of word blending are the same as their closure properties under the right-inverse of word blending.

We now consider the iterated blending operation $\bowtie_*$. Recall that, as mentioned in Sect. 2, for any language $L \subseteq \Sigma^*$, the language $L^{\bowtie_*}$ can be generated by a splicing system with null-context splicing rules defined as 6-tuples, as in [17]. As shown in [1], every splicing system where the rules are defined by 6-tuples, can also be implemented by a splicing system as defined in [27], which uses 4-tuple rules (see Definition 10). This connection, together with Proposition 2, allows us to express iterated word blending using so-called simple splicing systems [24], themselves a particular case of splicing systems based on 4-tuple splicing rules.

Definition 10 ([27]). *Let* $\sigma = (\Sigma, R)$ *be a splicing scheme, where* Σ *is the alphabet and* R *is a set of rules* $R \subseteq \Sigma^*\#\Sigma^*\$\Sigma^*\#\Sigma^*$. *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*

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

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* $\sigma^*(L) = \bigcup_{i \geqslant 0} \sigma^i(L)$ *with* $\sigma^0(L) = L$ *and* $\sigma^{i+1}(L) = \sigma(\sigma^i(L))$.

Simple splicing schemes are splicing schemes as above, but 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 $R = \Sigma\#\lambda\$\Sigma\#\lambda$. This observation together with Proposition 2 which showed that the word blending of two languages can be written $L_1 \bowtie L_2 = \bigcup_{a \in \Sigma}(L_1(a\Sigma^*)^{-1})a((\Sigma^*a)^{-1_l}L_2)$, gives us the following result.

Proposition 11. *For any language* $L \subseteq \Sigma^*$, *we have* $\sigma_{\bowtie}(L) = L \bowtie L$ *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 it is unary, since the rules use words of length at most 1. We also note that, even though in [24] 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 12. *Every full AFL is closed under iterated word 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 invisible sites. In [26], 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. □

Now, we will give an explicit construction for $L^{\bowtie_*}$ when L is a regular language. We will require the following lemma concerning the structure of words generated by the iterated blending operation.

Lemma 13. *Let $L \subseteq \Sigma^+$ be a language. Then for each word $w \in L^{\bowtie_*}$, there exists $n \in \mathbb{N}$ such that there are words $u_i \in \inf(L), 1 \leqslant i \leqslant n$ and $\alpha_j \in \Sigma^*, 1 \leqslant j \leqslant n$ and symbols $a_k \in \Sigma, 1 \leqslant k \leqslant n-1$ where*

1. *for $n > 1$,*
   (a) *$w = \alpha_1 a_1 \alpha_2 a_2 \cdots a_{n-1}\alpha_n$,*
   (b) *$u_i = a_{i-1}\alpha_i a_i \in \inf(L)$ for all $2 \leqslant i \leqslant n-1$,*
   (c) *$u_1 = \alpha_1 a_1 \in pref(L)$ and $u_n = a_{n-1}\alpha_n \in suff(L)$,*
2. *$u_1 = w \in L$ for $n = 1$.*

Proposition 14. *Given an NFA A, there exists an NFA A' recognizing the language $L(A)^{\bowtie_*}$ which is effectively constructible.*

This construction gives us a way to test whether a regular language L is closed under iterated blending.

Proposition 15. *Let L be a regular language. It is decidable whether or not L is closed under $\bowtie_*$.*

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 [24], 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. Here, we extend the result to consider the case when the base need not be finite.

Theorem 16. *It is decidable whether or not a regular language has a base over $\bowtie_*$.*

As a consequence, we are able to not only decide whether a regular language is closed under $\bowtie_*$, but if it is, we know there always exists a finite base that generates it.

Corollary 17. *Let L be a regular language closed under $\bowtie_*$. Then L can be generated by a finite base.*

Note that in [24] languages generated by simple splicing schemes are assumed to have finite bases by definition. There it was also shown that the class of languages generated by these simple splicing schemes is a subclass of the family of regular languages. Here we do not have the finite base restriction, and Corollary 17 shows that allowing regular bases does not give simple splicing schemes and iterated word blending any more power than restricting bases to be finite.

4 Decision Problems

This section investigates 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, X, Y are unknown languages, and $\bowtie$ is the word blending operation.

Proposition 18. *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 [22], since $\bowtie^r$ is the right-inverse of word blending, if there exists a solution Y to the given equation, then $Y' = (L \bowtie^r R^c)^c$ 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 $\bowtie^r$ 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 [25], 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 solution. □

Proposition 19. *The existence of a solution X to the equation $X \bowtie L = R$ is decidable for regular languages L and R.*

Proposition 20. *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 if and only if 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 solution exists.

We now consider the case when R is not empty. If there is a singleton solution $\{w\}$ to the equation $L \bowtie \{w\} = R$, where $L, R \subseteq \Sigma^+, w \in \Sigma^+$ then there is a shortest singleton solution of length $k \geqslant 1$, 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 finite state automaton that accepts R is at least k.

If $lg(w_s) = 1$, then $\lambda \notin R$, so the number of states of any finite state machine that recognizes R is at least 2, which is greater than the length of w_s.

Suppose $k \geqslant 2$. Define $L_i = (L \bowtie a_i) a_{i+1} \cdots a_k$ for $1 \leqslant i < k$, 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—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 $2 \leqslant j \leqslant k$, then $\alpha_1 a_1 \cdots a_{j-1}(a_j \cdots a_k)^+ \subseteq R$, and 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 $2 \leqslant j < h \leqslant k$, then $\alpha_1 a_1 \cdots a_{j-1}(a_j \cdots a_{h-1})^+ a_h \cdots a_k \subseteq R$, and 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 finite state machine for R is at least k. If the minimal deterministic finite automaton that generates R has k states, the algorithm for deciding the existence of a singleton solution will check all the words β, where $lg(\beta) \leqslant k$. The answer is "yes" if this algorithm finds a string β such that $L \bowtie \{\beta\} = R$, and "no" otherwise. □

Proposition 21. *The existence of a singleton solution $\{w\}$ to the equation $\{w\} \bowtie L = R$ is decidable for regular languages L and R.*

Proposition 22. *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 $\{w\}$ if and only if $L' = \Sigma^*$ 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. □

Corollary 23. *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 24. *1. The existence of a singleton solution $\{w\}$ to the equation $\{w\} \bowtie L = R$ is undecidable for a regular language R and a context-free language L.*

2. The existence of a solution X to the equation $X \bowtie L = R$ is undecidable for a regular language R and a context-free language L.

5 State Complexity

By Proposition 2, the family of regular languages is closed under word blending. Thus, we can consider the state complexity of the blending operation on two

regular languages. Recall from Proposition 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 [29].

First, for illustrative purposes, we will construct an NFA that recognizes the blending of two languages given by DFAs. Let $A_m = (Q_m, \Sigma, \delta_m, s_m, F_m)$ be a DFA with $m \geqslant 1$ states that recognizes the language L_m and let $A_n = (Q_n, \Sigma, \delta_n, s_n, F_n)$ be a DFA with $n \geqslant 1$ states that recognizes the language L_n. We construct an NFA $B' = (Q', \Sigma, \delta', s', F')$, where $Q' = Q_m \cup Q_n$, $s' = s_m$, $F' = F_n$, and the transition function $\delta' : Q' \times \Sigma \to 2^{Q'}$ is defined for all $q \in Q'$ and $a \in \Sigma$ by

$$\delta'(q,a) = \begin{cases} \bigcup_{p \in Q_n} \delta_n(p,a) & \text{if } q \in Q_m \text{ and } \delta_m(q,a) \text{ is not the sink state,} \\ \delta_m(q,a) & \text{if } q \in Q_m \text{ and } \delta_m(q,a) \text{ is the sink state,} \\ \delta_n(q,a) & \text{if } q \in Q_n. \end{cases}$$

In Fig. 1, we define two DFAs A_m and A_n and show the NFA B' resulting from the construction described above. Intuitively, the machine B' operates by first reading the input word assuming that it is the prefix of some word recognized by A_m. Since the blending occurs on only one symbol, the machine guesses at which symbol the blend occurs. Once the blend occurs the machine continues and assumes the rest of the word is the suffix of some word recognized by A_n.

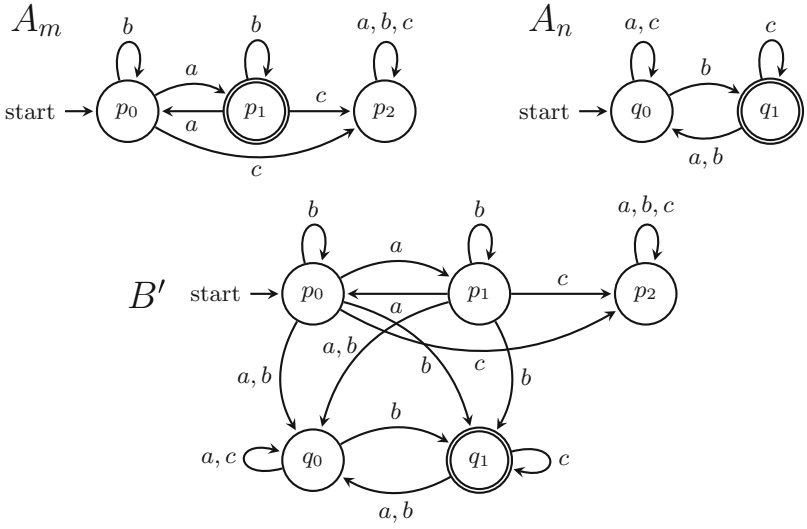

Fig. 1. The NFA B' recognizes the blend of the languages recognized by the DFAs A_m and A_n

Proposition 25. *The NFA B' recognizes the language $L_m \bowtie L_n$.*

Now, using the same basic idea, we will construct a DFA that recognizes the language of the blending of the two languages recognized by two given DFAs A_m and A_n. We construct a DFA $A' = (Q', \Sigma, \delta', s', F')$ where

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

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

Figure 2 shows the DFA A' that results from following the construction described above, where A_m and A_n are the DFAs shown in Fig. 1. 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 $(q_\varnothing, P)$, where $q_\varnothing$ is the sink state of A_m. The second phase only occurs when the blend 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. We will show this formally in the following.

Proposition 26. *The DFA A' recognizes the language $L_m \bowtie L_n$.*

A simple count of the number of states in the state set of A' gives us as many as $m2^n$ states. 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.

Theorem 27. *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 $m+n-1$ if both L_m and L_n are finite or 1 otherwise. Furthermore, this bound is reachable.*

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

Lemma 28. *The DFA A' requires at most $(m-1)\cdot(k-1)+2^n+1$ states, where $k = |\Sigma| \leqslant 2^n$.*

Lemma 29. *Let $k \geqslant 3$ and $m, n \geqslant 2$. 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.*

These results together give us the following theorem.

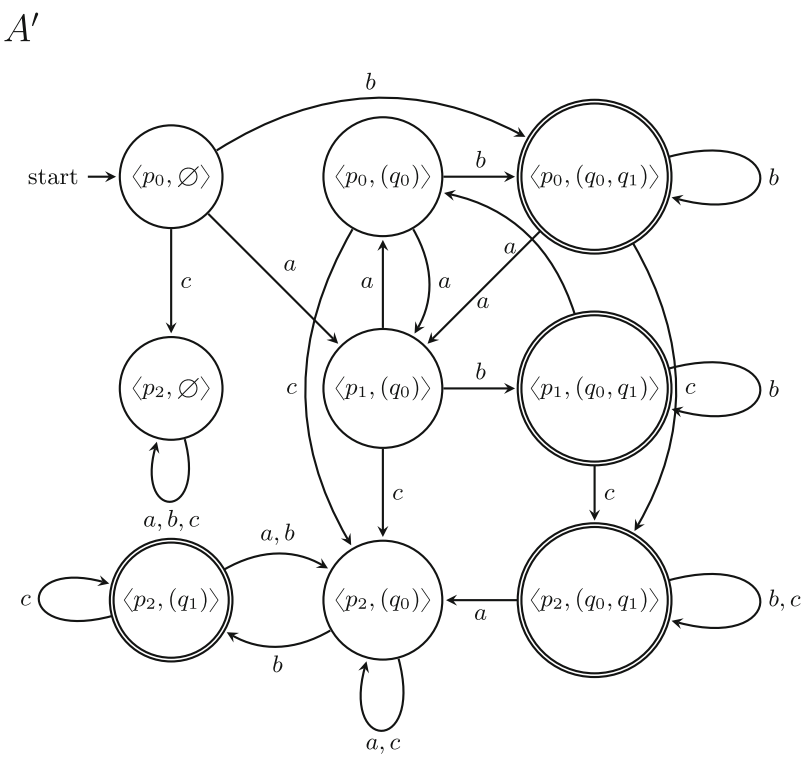

Fig. 2. The DFA A' recognizes the blend of the languages recognized by A_m and A_n from Fig. 1

Theorem 30. *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. Then*

$$sc(L_m \bowtie L_n) \leqslant (m-1) \cdot (k-1) + 2^n + 1,$$

and this bound can be reached in the worst case.

Acknowledgements. We thank Giuditta Franco for fruitful discussions on modelling the outcomes of various XPCR experiments.

References

1. Bonizzoni, P., Ferretti, C., Mauri, G., Zizza, R.: Separating some splicing models. Inf. Process. Lett. **79**(6), 255–259 (2001)
2. Bonizzoni, P., Felice, C.D., Zizza, R.: The structure of reflexive regular splicing languages via Schützenberger constants. Theor. Comput. Sci. **334**(1), 71–98 (2005)
3. Brzozowski, J., Kari, L., Li, B., Szykuła, M.: State complexity of overlap assembly. arXiv preprint arXiv:1710.06000 (2017)

4. Carausu, A., Păun, Gh.: String intersection and short concatenation. Revue Roumaine de Mathématiques Pures et Appliquées **26**(5), 713–726 (1981)
5. Csuhaj-Varju, E., Petre, I., Vaszil, Gy.: Self-assembly of strings and languages. Theor. Comput. Sci. **374**(1), 74–81 (2007)
6. Domaratzki, M.: Minimality in template-guided recombination. Inf. Comput. **207**(11), 1209–1220 (2009)
7. Enaganti, S.K., Ibarra, O.H., Kari, L., Kopecki, S.: On the overlap assembly of strings and languages. Nat. Comput. **16**(1), 175–185 (2017)
8. Enaganti, S.K., Ibarra, O.H., Kari, L., Kopecki, S.: Further remarks on DNA overlap assembly. Inf. Comput. **253**, 143–154 (2017)
9. Franco, G.: A polymerase based algorithm for SAT. In: Coppo, M., Lodi, E., Pinna, G.M. (eds.) ICTCS 2005. LNCS, vol. 3701, pp. 237–250. Springer, Heidelberg (2005). https://doi.org/10.1007/11560586_20
10. Franco, G., Bellamoli, F., Lampis, S.: Experimental analysis of XPCR-based protocols. arXiv preprint arXiv:1712.05182 (2017)
11. Franco, G., Giagulli, C., Laudanna, C., Manca, V.: DNA extraction by XPCR. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 104–112. Springer, Heidelberg (2005). https://doi.org/10.1007/11493785_9
12. Franco, G., Manca, V.: Algorithmic applications of XPCR. Nat. Comput. **10**(2), 805–819 (2011)
13. Franco, G., Manca, V., Giagulli, C., Laudanna, C.: DNA recombination by XPCR. In: Carbone, A., Pierce, N.A. (eds.) DNA 2005. LNCS, vol. 3892, pp. 55–66. Springer, Heidelberg (2006). https://doi.org/10.1007/11753681_5
14. Golan, J.S.: The Theory of Semirings with Applications in Mathematics and Theoretical Computer Science. Addison-Wesley Longman Ltd., Reading (1992)
15. Goode, E., Pixton, D.: Recognizing splicing languages: syntactic monoids and simultaneous pumping. Discrete Appl. Math. **155**(8), 989–1006 (2007)
16. Gries, S.T.: Shouldn't it be breakfunch? A quantitative analysis of blend structure in English. Linguistics **42**(3), 639–667 (2004)
17. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. Bull. Math. Biol. **49**(6), 737–759 (1987)
18. Holzer, M., Jakobi, S.: Chop operations and expressions: descriptional complexity considerations. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 264–275. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22321-1_23
19. Holzer, M., Jakobi, S.: State complexity of chop operations on unary and finite languages. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 169–182. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31623-4_13
20. Holzer, M., Jakobi, S., Kutrib, M.: The chop of languages. In: Automata and Formal Languages, 13th International Conference, AFL 2011, Debrecen, pp. 197–210 (2011)
21. Ito, M., Lischke, G.: Generalized periodicity and primitivity for words. Math. Logic Q. **53**(1), 91–106 (2007)
22. Kari, L.: On language equations with invertible operations. Theor. Comput. Sci. **132**(1–2), 129–150 (1994)
23. Manca, V., Franco, G.: Computing by polymerase chain reaction. Math. Biosci. **211**(2), 282–298 (2008)
24. Mateescu, A., Păun, Gh., Rozenberg, G., Salomaa, A.: Simple splicing systems. Discrete Appl. Math. **84**(1–3), 145–163 (1998)

25. Mateescu, A., Salomaa, A.: Handbook of Formal Languages. Springer, New York (1997)
26. Pixton, D.: Splicing in abstract families of languages. Theor. Comput. Sci. **234**, 135–166 (2000)
27. Păun, Gh.: On the splicing operation. Discrete Appl. Math. **70**(1), 57–79 (1996)
28. Salomaa, A.: Formal Languages. Academic Press Inc., New York (1977)
29. Salomaa, K., Yu, S.: On the state complexity of combined operations and their estimation. Int. J. Found. Comput. Sci. **18**(4), 683–698 (2007)