Bounded Hairpin Completion

Masami Ito[1], Peter Leupold[1†], Florin Manea[2,4‡], and Victor Mitrana[2,3§]

[1] Department of Mathematics, Faculty of Science
Kyoto Sangyo University, Department of Mathematics
Kyoto 603-8555, Japan
ito@cc.kyoto-su.ac.jp, Peter.Leupold@web.de

[2] Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 010014, Bucharest, Romania
flmanea@fmi.unibuc.ro,mitrana@fmi.unibuc.ro

[3] Dept. Organización y Estructura de la Información
Universidad Politécnica de Madrid
Crta. de Valencia km. 7 - 28031 Madrid, Spain

[4] Otto-von-Guericke-University Magdeburg,
Faculty of Computer Science
PSF 4120, D-39016 Magdeburg, Germany.

Abstract. Hairpin completion is a formal operation inspired from biochemistry. Here we consider a restricted variant of hairpin completion called bounded hairpin completion. Applied to a word encoding a single stranded molecule x such that either a suffix or a prefix of x is complementary to a subword of x, hairpin completion produces a new word z, which is a prolongation of x to the right or to the left by annealing.
Although this operation is a purely mathematical one and the biological reality is just a source of inspiration, it seems rather unrealistic to impose no restriction on the length of the prefix or suffix added by the hairpin completion. The restriction considered here concerns the length of all prefixes and suffixes that are added to the current word by hairpin completion. They cannot be longer than a given constant. Closure properties of some classes of formal languages under the non-iterated and iterated bounded hairpin completion are investigated. We consider the bounded hairpin completion distance between two words and generalize this distance to languages and discuss algorithms for computing them. Finally also the inverse operation, namely bounded hairpin reduction, as well as the set of all primitive bounded hairpin roots of a regular language are considered.

[†] This work was done, while the author was funded as a post-doctoral fellow by the Japanese Society for the Promotion of Science under number P07810.

[‡] Work supported by the Alexander von Humboldt Foundation

[§] Work supported by the Grant-in-Aid No. 19-07810 by Japan Society for the Promotion of Science

1 Introduction

This paper is a continuation of a series of works started with [4] (based on some ideas from [3]), where a new formal operation on words inspired by the DNA manipulation called hairpin completion was introduced. That initial work has been followed up by a several related papers [16–19], where both the hairpin completion as well as its inverse operation, namely the hairpin reduction, were further investigated.

Several problems remained unsolved in these papers. This is the mathematical motivation for the work presented here. By considering a weaker variant of the hairpin completion operation, called here the bounded hairpin completion, we hope to be able to solve some of the problems in this new setting that remained unsolved in the aforementioned papers. Another motivation is a practical one, closely related to the biochemical reality that inspired the definition of this operation. It seems more practical to consider that the prefix/suffix added by the hairpin completion cannot be arbitrarily long. In a laboratory, there will only be a finite amount of resources, especially time, available for every step of a computation; thus the length of the added word would be bounded by both the amount of additional nucleic acids in the test tube and the time given for one step of computation.

We briefly highlight some of the biological background that inspired the definition of the *Watson-Crick superposition* in [3]. The starting point is the structure of the DNA molecule. It consists of a double strand, each DNA single strand being composed by nucleotides which differ from each other in their bases: A (adenine), G (guanine), C (cytosine), and T (thymine). The two strands which form the DNA molecule are kept together by relatively weak hydrogen bonds between the bases: A always bonds with T and C with G. This phenomenon is usually referred to as *Watson-Crick complementarity.* The formation of these hydrogen bonds between complementary single DNA strands is called *annealing.*

A third essential feature from biochemistry is the PCR (Polymerase Chain Reaction). From two complementary, annealed strands, where one is shorter than the other, it produces a complete double stranded DNA molecule as follows: enzymes called polymerases add the missing bases (if they are available in the environment) to the shorter strand called *primer* and thus turn it into a complete complement of the longer one called *template.*

We now informally explain the superposition operation and how it can be related to the aforementioned biochemical concepts. Let us consider the following hypothetical biological situation: two single stranded DNA molecules x and y are given such that a suffix of x is Watson-Crick complementary to a prefix of y or a prefix of x is Watson-Crick complementary to a suffix of y, or x is Watson-Crick complementary to a subword of y. Then x and y get annealed in a DNA molecule with a double stranded part by complementary base pairing and then a complete double stranded molecule is formed by DNA polymerases. The mathematical expression of this hypothetical situation defines the superposition operation. Assume that we have an alphabet and a complementary relation on its letters. For two words x and y over this alphabet, if a suffix of x is complementary to a prefix of y or a prefix of x is complementary to a suffix of y, or x is complementary to a subword of y, then x and y bond together by complementary letter pairing and then a complete double stranded word is formed by the prolongation of x and y. Now the word obtained by the prolongation of x is considered to be the result of the superposition applied to x and y. Clearly, this is just a mathematical operation that resembles a biological reality considered here in an idealized way.

On the other hand, it is known that a single stranded DNA molecule might produce a hairpin structure, a phenomenon based on the first two biological principles mentioned above. Here one part of the strand bonds to another part of the same strand. In many DNA-based algorithms, these DNA molecules often cannot be used in the subsequent steps of the computation. Therefore it has been the subject of a series of studies to find encodings that will avoid the formation of hairpins, see e.g. [5, 8, 9] or [13] and subsequent work for investigations in the context of Formal Language Theory. On the other hand, those molecules which may form a hairpin structure have been used as the basic feature of a new computational model reported in [23], where an instance of the 3-SAT problem has been solved by a DNA-algorithm whose second phase is mainly based on the elimination of hairpin structured molecules.

We now consider again a hypothetical biochemical situation: we are given one single stranded DNA molecule z such that either a prefix or a suffix of z is Watson-Crick complementary to a subword of z. Then the prefix or suffix of z and the corresponding subword of z get annealed by complementary base pairing and then z is lengthened by DNA polymerases up to a complete hairpin structure. The mathematical expression of this hypothetical situation defines the hairpin completion operation. By this formal operation one can generate a set of words, starting from a single word. This operation is considered in [4] as an abstract operation on formal languages. Some algorithmic problems regarding the hairpin completion are investigated in [16]. In [17] the inverse operation to the hairpin completion, namely the hairpin reduction, is introduced and one compares some properties of the two operations. This comparison is continued in [18], where a mildly context-sensitive class of languages is obtained as the homomorphic image of the hairpin completion of linear context-free languages. This is, to our best knowledge, the first class of mildly context-sensitive languages obtained in a way that does not involve grammars or acceptors.

In the aforementioned papers, no restriction is imposed on the length of the prefix or suffix added by the hairpin completion. This fact seems rather unrealistic though this operation is a purely mathematical one and the biological reality is just a source of inspiration. On the other hand, several natural problems regarding the hairpin completion remained unsolved in the papers mentioned above. A usual step towards solving them might be to consider a bit less general setting and try to solve the problems in this new settings. Therefore, we consider here a restricted variant of the hairpin completion, called *bounded hairpin completion*. This variant assumes that the length of the prefix and suffix added by the hairpin completion is bounded by a constant. We investigate the closure properties of some classes of formal languages under the non-iterated and iterated bounded hairpin completion. We then consider the bounded hairpin completion distance between two words and generalize this distance to languages and discuss algorithms for computing them. The inverse operation of the bounded hairpin completion, namely bounded hairpin reduction, as well as the set of all primitive bounded hairpin roots of a regular language are finally considered.

2 Basic definitions

We assume the reader to be familiar with the fundamental concepts of formal language theory and automata theory, particularly the notions of grammar and finite automaton [21] and basics from the theory of abstract families of languages [24].

An alphabet is always a finite set of letters. For a finite set A we denote by $card(A)$ the cardinality of A. The set of all words over an alphabet V is denoted by V^*. The empty word is written λ; moreover, $V^+ = V^* \setminus \{\lambda\}$. Two languages are considered to be equal if they contain the same words with the possible exception of the empty word.

A concept from the theory of abstract families of languages that we will refer to is that of a *trio*. This is is a non-empty family of languages closed under non-erasing morphisms, inverse morphisms and intersection with regular languages. A trio is *full* if it is closed under arbitrary morphisms.

Given a word w over an alphabet V, we denote by $|w|$ its length, while $|w|_a$ denotes the number of occurrences of the letter a in w. If $w = xyz$ for some $x, y, z \in V^*$, then x, y, z are called prefix, subword, suffix, respectively, of w. For a word w, $w[i..j]$ denotes the subword of w starting at position i and ending at position j, $1 \leq i \leq j \leq |w|$. If $i = j$, then $w[i..j]$ is the i-th letter of w which is simply denoted by $w[i]$.

Let Ω be a "superalphabet", that is an infinite set such that any alphabet considered in this paper is a subset of Ω. In other words, Ω is the *universe* of the alphabets in this paper, i.e., all words and languages are over alphabets that are subsets of Ω. An *involution* over a set S is a bijective mapping $\sigma : S \longrightarrow S$ such that $\sigma = \sigma^{-1}$. Any involution σ on Ω such that $\sigma(a) \neq a$ for all $a \in \Omega$ is said to be a *Watson-Crick involution*. Despite the fact that this is nothing more than a fixed point-free involution, we prefer this terminology since the hairpin completion defined later is inspired by the DNA lengthening by polymerases, where the Watson-Crick complementarity plays an important role. Let $\bar{\cdot}$ be a Watson-Crick involution fixed for the rest of the paper. The Watson-Crick involution is extended to a morphism from Ω to Ω^* in the usual way. We say that the letters a and $\overline{a}$ are complementary to each other. For an alphabet V, we set $\overline{V} = \{\overline{a} \mid a \in V\}$. Note that V and $\overline{V}$ can intersect and they can be, but need not be, equal. Recall that the DNA alphabet consists of four letters, $V_{DNA} = \{A, C, G, T\}$, which are abbreviations for the four nucleotides and we may set $\overline{A} = T$, $\overline{C} = G$.

We denote by $(\ldots)^R$ the mapping $V^* \longrightarrow V^*$ defined by, $(a_1a_2\ldots a_n)^R = a_n \ldots a_2a_1$. Note that ^{R} is both an involution and an *anti-morphism*, because

$(xy)^R = y^R x^R$ for all $x, y \in V^*$. Note also that the two mappings $\bar{\cdot}$ and $\cdot^R$ commute, namely, for any word x, $(\overline{x})^R = \overline{x^R}$ holds.

The reader is referred to [4] or any of the subsequent papers [16–19] for the definition of the (unbounded) *k-hairpin completion*; it is essentially the same as for the bounded variant defined below, only without the bound $|\gamma| \leq p$. Thus the prefix or suffix added by hairpin completion can be arbitrarily long. By the mathematical and biological reasons mentioned in the introductory part, in this work we are interested in a restricted variant of this operation that allows only prefixes and suffixes of a length bounded by a constant to be added. Formally, if V is an alphabet, then for any $w \in V^+$ we define the p-bounded k-hairpin completion of w, denoted by $pHC_k(w)$, for some $k, p \geq 1$, as follows:

$$\begin{aligned}
pHC \curvearrowleft_k (w) &= \{\overline{\gamma^R} w \mid w = \alpha\beta\overline{\alpha^R}\gamma, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*, |\gamma| \leq p\}\\
pHC \curvearrowright_k (w) &= \{w\overline{\gamma^R} \mid w = \gamma\alpha\beta\overline{\alpha^R}, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*, |\gamma| \leq p\}\\
pHC_k(w) &= pHC \curvearrowleft_k (w) \cup pHC \curvearrowright_k (w).
\end{aligned}$$

This operation is schematically illustrated in Figure 1.

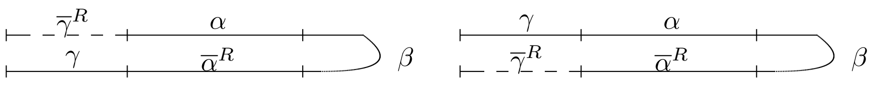

Figure 1: Bounded hairpin completion

The *p-bounded hairpin completion* of w is defined by

$$pHC(w) = \bigcup_{k\geq 1} pHC_k(w).$$

As above, the p-bounded hairpin completion operation is naturally extended to languages by

$$pHC_k(L) = \bigcup_{w\in L} pHC_k(w) \qquad pHC(L) = \bigcup_{w\in L} pHC(w).$$

The iterated version of the p-bounded hairpin completion is defined in a similar way to the unbounded case, namely:

$$\begin{array}{ll}
pHC_k^0(w) = \{w\}, & pHC^0(w) = \{w\},\\
pHC_k^{n+1}(w) = pHC_k(pHC_k^n(w)), & pHC^{n+1}(w) = pHC(pHC^n(w)),\\
pHC_k^*(w) = \bigcup_{n\geq 0} pHC_k^n(w), & pHC^*(w) = \bigcup_{n\geq 0} pHC^n(w),\\
pHC_k^*(L) = \bigcup_{w\in L} pHC_k^*(w), & pHC^*(L) = \bigcup_{w\in L} pHC^*(w).
\end{array}$$

3 The non-iterated case

The case of bounded hairpin completion is rather different in comparison to the unbounded variant considered in [16–19]. As it was expected, the closure problem of any trio under bounded hairpin completion is simple: every (full) trio is closed under this operation.

Proposition 1 *Every (full) trio is closed under p-bounded k-hairpin completion for any $k, p \geq 1$.*

Proof. A generalized sequential machine (*gsm*) A can easily add the finite suffix (prefix) from a hairpin completion of length at most p to its input. Before starting its computation, A guesses both the length of the new factor i which is bounded by p and the length of the complementary factors j which is bounded by k; further it guesses the option of adding either a prefix or a suffix. In the former case, it outputs i random letters and remembers them in its state. Then it reads the first j input letters and stores them, too. Every letter that is read is output directly in the same step. Continuing, A reads and outputs the entire string, always remembering the last $i + j$ letters that have been read.

When the end of the input is reached, the initial $i + j$ output letters can be compared to the last $i+j$ letters. If they comply with the definition of p-bounded k-hairpin completion, then the computation was successful. Otherwise the *gsm* has guessed wrongly and rejects. Adding a suffix follows analogous tactics. As every trio is closed under *gsm* mappings, see [24], we are done. □

We recall that neither the class of regular languages nor that of context-free languages is closed under unbounded hairpin completion, see [4]. By Proposition 1 both classes are closed under bounded hairpin completion.

On the other hand, in [16] it was proved that if the membership problem for a given language L is decidable in $\mathcal{O}(f(n))$, then the membership problem for the hairpin completion of L is decidable in $\mathcal{O}(nf(n))$ for any $k \geq 1$. Further, the factor n is not needed for the class of regular languages, but the problem of whether or not this factor is needed for other classes remained open in [16]. An easy adaption of the algorithm provided there shows that this factor is never needed in the case of bounded hairpin completion and thus membership is always decidable in $\mathcal{O}(f(n))$.

4 The iterated case

As in the non-iterated case, the iterated bounded hairpin completion offers also a rather different picture of closure properties in comparison to the unbounded variant considered in the same papers cited above. We start with a general result.

Theorem 1 *Let $p, k \geq 1$ and $\mathcal{F}$ be a (full) trio closed under substitution. Then $\mathcal{F}$ is closed under iterated p-bounded k-hairpin completion if and only if $pHC_k^*(w) \in \mathcal{F}$ for any word w.*

Proof. The "only if" part is obvious as any trio contains all singleton languages.

For the "if" part, let $L \in \mathcal{F}$ be a language over the alphabet V. We write $L = L_1 \cup L_2$, where

$$L_1 = \{x \in L \mid |x| < 2(k+p)+1\},$$
$$L_2 = \{x \in L \mid |x| \geq 2(k+p)+1\}.$$

Clearly, $pHC_k^*(L) = pHC_k^*(L_1) \cup pHC_k^*(L_2)$. As any trio contains all finite languages, it follows that any trio closed under substitution is closed under union. Therefore, as L_1 is a finite language, we conclude that $pHC_k^*(L_1) \in \mathcal{F}$. Consequently, it remains to show that $pHC_k^*(L_2) \in \mathcal{F}$.

Let $\alpha, \beta \in V^+$ be two arbitrary words with $|\alpha| = |\beta| = k + p$. We define $L_2(\alpha, \beta) = L_2 \cap \{\alpha\}V^+\{\beta\}$. We have that

$$L_2 = \bigcup_{|\alpha|=|\beta|=k+p} L_2(\alpha,\beta) \quad \text{and} \quad pHC_k^*(L_2) = \bigcup_{|\alpha|=|\beta|=k+p} pHC_k^*(L_2(\alpha,\beta)).$$

On the other hand, it is plain that $pHC_k^*(L_2(\alpha,\beta)) = s(pHC_k^*(\alpha X \beta))$, where X is a new symbol not in V and s is a substitution $s : (V \cup \{X\})^* \longrightarrow 2^{V^*}$ defined by $s(a) = \{a\}$ for all $a \in V$ and $s(X) = \{w \in V^+ \mid \alpha w \beta \in L_2(\alpha,\beta)\}$. The language $\{w \in V^+ \mid \alpha w \beta \in L_2(\alpha,\beta)\}$ is in $\mathcal{F}$ (even if $\mathcal{F}$ is not full) as it is the image of a language from $\mathcal{F}$, namely $L_2(\alpha,\beta)$, through a *gsm* mapping that deletes both the prefix and suffix of length $k + p$ of the input word. By the closure properties of $\mathcal{F}$, it follows that $pHC_k^*(L_2(\alpha,\beta))$ is in $\mathcal{F}$ for any α, β as above, which completes the proof. □

We recall that none of the families of regular, linear context-free, and context-free languages is closed under iterated unbounded hairpin completion. Here the bounded hairpin completion is much more tractable.

Corollary 1 *The family of context-free languages is closed under iterated p-bounded k-hairpin completion for any $k, p \geq 1$.*

Proof. By the previous result, it suffices to prove that $pHC_k^*(w)$ is context-free for any word w. To this end, we show how to generate $pHC_k^*(w)$ by a grammar and a few more operations that preserve context-freeness. Given the word $w \in V^+$ and the parameters $k, p \geq 1$, the corresponding grammar is $G = (\{S, X\}, V \cup \{\#\}, S, P)$, where the set of productions P is the following:

$$\begin{aligned} P = {} & \{S \to yXz \mid w = zy\} \cup \{\overline{z}^R yXz \to \overline{z}^R yX\overline{y}^R z \mid 1 \leq |y| \leq p, |z| = k\} \\ & \cup \{\overline{z}^R Xyz \to \overline{z^R y^R} Xyz \mid 1 \leq |y| \leq p, |z| = k\} \cup \{X \to \#\}. \end{aligned}$$

Notice that the rules from the second and third set are not context-free. The left hand sides are of length up to $p + 2k + 1$. However, they are rather simple in the sense that only the non-terminal X occurs on both sides of the rules, and that the left hand sides reappear on the right hand sides; only at one point right next to the non-terminal a new factor is added. By a result of Baker languages generated by this type of grammar are context-free [1]. Therefore also the language generated by G is context-free.

Now let us see in what relation the language of G stands to $pHC_k^*(w)$. The rules from the first set can rewrite the start symbol to any of the cyclic permutations of w with X marking the point where w ends and starts. The advantage of these cyclic permutations is that all the up to $2k + 2p$ decisive letters for a hairpin completion are next to each other with only X added. Thus an operation that acts on the two ends of the original string is turned into a local operation and can be simulated by the finitely many rules of the second and third set in the definition of P. Finally, the rule $X \to \#$ ends this simulations and leaves $\#$ as a mark in the place of X.

To obtain $pHC_k^*(w)$ from the language generated we do the following. First we take all cyclic permutations of the words generated. To filter out the ones that are in the order that we need, we eliminate all words that do not have $\#$ in the last position via the intersection with $V^+ \cdot \{\#\}$. It remains to remove $\#$, which can be done via a morphism h that erases $\#$ and leaves all letters of V unchanged. Summarizing, we have that

$$pHC_k^*(w) = h(cp(L(G)) \cap V^+ \cdot \{\#\}).$$

Here *cp* stands for the mapping that takes every word to the set of all its circular permutations and every language to the set of all circular permutations of its words. As the class of context-free languages is closed under all the oprations involved circular permutation we infer that $pHC_k^*(w)$ is context-free. The only non-standard operation is cyclic permutation, and closure of context-free languages under this operation was shown by Ruohonen [22]. □

The above argument does not work for the class of linear context-free languages as this class is known not to be closed under circular permutation. However, also this family is closed under iterated bounded hairpin completion.

Theorem 2 *The family of linear context-free languages is closed under iterated p-bounded k-hairpin completion for any $k, p \geq 1$.*

Proof. Let L be a language generated by the linear grammar $G = (N, T, S, P)$. We construct a linear grammar that generates $pHC_k^*(L)$. The idea of this construction is the following: let w' be a word in $pHC_k^*(L)$. Then there exists a word $w \in L$ such that $w' \in pHC_k^*(w)$. The new grammar generates w' starting from the parts that are not present in w by simulating a derivation via hairpin completion from w to w' in inverse order. To this end, it guesses what the prefix and suffix of length up to $k+p$ are that were involved in this last step. They are stored in non-terminals of the form $[\alpha, \beta]$. The factor that is produced by this hairpin completion is output on the correct side of the non-terminal, while the latter is rewritten in such a way that this factor is removed and the factors α and β are prolonged to specify the next hairpin reduction.

At some point, the grammar guesses that all hairpin completions have been simulated. It remains to generate w and to check whether it has the final α and β as prefix and suffix, respectively. To this end, the last non-terminal in a derivation of w in G is guessed and placed between α and β in a non-terminal of the form $[\alpha, A, \beta]$.

We now give the technical details of the grammar $G' = (N', T, S', P')$ doing the things described above.

$$\begin{aligned} N' = N \cup \{S'\} &\cup \{[\alpha, \beta] \mid \alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k+p\} \\ &\cup \{[\alpha, A, \beta] \mid \alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k+p, A \in N\}, \end{aligned}$$

and the set of productions P' is defined by as follows, where in the definition of every set $\alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k+p, A \in N$ holds:

$$\begin{aligned}
P' = P &\cup \{S' \to S\} \cup \{S' \to [\alpha,\beta] \mid \alpha,\beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k+p\} \\
&\cup \{[\alpha,\beta] \to [\alpha',\beta']\overline{y}^R \mid \alpha = \alpha' = yvw, \beta = u\overline{v^R y^R}, |v| = k, |y| \leq p, \\
&\quad \beta' = xu\overline{v}^R, x \in T^*, |\beta'| \leq k+p\} \\
&\cup \{[\alpha,\beta] \to y[\alpha',\beta'] \mid \alpha = yvw, \beta = \beta' = u\overline{v^R y^R}, |v| = k, |y| \leq p, \\
&\quad \alpha' = vwx, x \in T^*, |\alpha'| \leq k+p\} \\
&\cup \{[\alpha,\beta] \to [\alpha,S,\beta] \mid \alpha,\beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k+p\} \\
&\cup \{[\alpha,A,\beta] \to x[\alpha',B,\beta']y \mid A \to xBy \in P, \alpha = x\alpha', \beta = \beta' y, \alpha', \beta' \in T^*\} \\
&\cup \{[\alpha,A,\beta] \to \alpha x[\lambda,B,\beta']y \mid A \to \alpha xBy \in P, \beta = \beta' y, \beta' \in T^*\} \\
&\cup \{[\alpha,A,\beta] \to x[\alpha',B,\lambda]y\beta \mid A \to xBy\beta \in P, \alpha = x\alpha', \alpha' \in T^*\} \\
&\cup \{[\lambda,A,\lambda] \to A \mid A \in N\}.
\end{aligned}$$

As described in the informal description above, we have the derivation

$$S' \Longrightarrow^* x[\alpha,\beta]y \Longrightarrow x[\alpha,S,\beta]y \Longrightarrow^* x\alpha w\beta y$$

in G' if and only if $S \Longrightarrow^* \alpha w\beta$ in G and $x\alpha w\beta y \in pHC_k^*(\alpha w\beta)$. This concludes the proof. □

The problem of whether or not the iterated unbounded hairpin completion of a word is context-free is open. By the previous result, it follows that the iterated bounded hairpin completion of a word is always linear context-free. We do not know whether this language is always regular. More generally, the status of the closure under iterated bounded hairpin completion of the class of regular languages remains unsettled.

We finish this section with another general result.

Theorem 3 *Every trio closed under circular permutation and iterated finite substitution is closed under iterated bounded hairpin completion.*

Proof. We take two positive integers $k, p \geq 1$. Let $\mathcal{F}$ be a family of languages with the above properties and $L \subseteq V^*$ be a language in $\mathcal{F}$. Let L_1 be the circular permutation of $L\{\#\}$, where $\#$ is a new symbol not in V. L_1 still lies in $\mathcal{F}$, because the closure under cyclic permutation is a precondition in the theorem's statement. We consider the alphabet $W = \{[x\#y] \mid x, y \in V^*, 0 \leq |x|, |y| \leq p+k\}$ and define the morphism $h : (W \cup V)^* \longrightarrow (V \cup \{\#\})^*$ by $h([x\#y]) = x\#y$, for any $[x\#y] \in W$, and $h(a) = a$, for any $a \in V$. We now consider the language $L_2 \in \mathcal{F}$ given by $L_2 = h^{-1}(L_1)$. By the closure properties of $\mathcal{F}$, the language $L_3 = s^*(L_2)$ is in $\mathcal{F}$, where s is the finite substitution $s : (W \cup V)^* \longrightarrow 2^{(W \cup V)^*}$ defined by $s(a) = \{a\}, a \in V$, and $s([x\#y]) = \{x\#y\} \cup R$ with

$$\begin{aligned}
R = \{&[x\#\overline{u}^R y] \mid x = vzu, y = \overline{z}^R w, u, v, z, w \in V^*, |z| = k, \\
&|\overline{u}^R y| \leq p+k, |u| \leq p\} \cup \\
\{&[x\#\overline{u}^R y']y'' \mid x = vzu, y = \overline{z}^R w = y'y'', u, v, z, w, y', y'' \in V^*, |z| = k,
\end{aligned}$$

$$|\overline{u}^R y'| = p+k, |u| \le p\}\} \cup$$
$$\{[x\overline{u}^R\#y] \mid x = w\overline{z}^R, y = uzv, u, v, z, w \in V^*, |z| = k,$$
$$|x\overline{u}^R| \le p+k, |u| \le p\}\} \cup$$
$$\{x''[x'\overline{u}^R\#y] \mid x = w\overline{z}^R = x''x', y = uzv, u, v, z, w, x', x'' \in V^*, |z| = k,$$
$$|x'\overline{u}^R| = p+k, |u| \le p\}\}.$$

Finally, let L_4 be the circular permutation of $h(L_3)$. Similarly to the proof of Theorem 1 we see that

$$pHC_k^*(L) = g(L_4 \cap V^*\{\#\}),$$

where g is the morphism that deletes # and leaves unchanged all symbols from V. □

5 Bounded Hairpin Completion Distance

In [16] the *hairpin completion distance* between two words x and y is defined as the minimal number of hairpin completions which can be applied either to x in order to obtain y or to y in order to obtain x. If none of them can be obtained from the other by iterated hairpin completion, then the distance is ∞. In the cited work it is shown that the hairpin completion distance between two words x and w can be computed in $\mathcal{O}(n^3)$ time, where $n = \max(|x|, |w|)$.

We first consider here the bounded hairpin completion distance between two words. Formally, the p-bounded k-hairpin completion distance between x and y, denoted by $pHCD_k(x, y)$ is defined by:

$$pHCD_k(x,y) = \begin{cases} \min\{m \mid x \in pHC_k^m(y) \text{ or } y \in pHC_k^m(x)\}, \\ \infty, \text{ if neither } x \in pHC_k^*(y) \text{ nor } y \in pHC_k^*(x) \end{cases}$$

We stress from the very beginning that the function defined above, applied on pairs of words, is not a distance function in the strict mathematical sense, since it does not necessarily verify the triangle inequality. Rather, it can be seen as a similarity measure between strings, or, if we consider our biological motivation, a measure that tells us how many evolution steps are needed to transform a string into the other. However, we prefer to call it distance for sake of uniformity as many similar measures are called distances in the literature.

The algorithm presented in [16] can easily be adapted to compute the p-bounded k-hairpin distance between two words x and w; the time complexity obtained in this way is $\mathcal{O}(n^2 p)$. In the following we show that this complexity can be decreased to $\mathcal{O}(n^2 \log p)$; moreover, we show that the algorithm we present here can be generalized to compute the unbounded hairpin completion distance in $\mathcal{O}(n^2 \log n)$ time.

5.1 Data Structures

First we present several data structures results that are useful in the sequel (see [6, 10]). Note that all the time bounds we give here hold on the unit-cost

RAM model (see [14] for a discussion on how the complexity of algorithms is evaluated).

We recall a variant of the so-called Range Minimum Query Problem:

Problem 1. (p-RMQ)
Let T be an array with n elements from a totally ordered set (with order relation $\leq$) and p be a natural number, with $1 \leq p \leq n$. Preprocess the array T in order to be able to answer queries "find $\min pos_T(i,j)$*", for the indices i and j, with $i \leq j \leq i+p-1$, assuming that* $\min pos_T(i,j) = arg \min_{k\in\{i,\ldots,j\}} T[k]$ *(i.e.* $\min pos_T(i,j)$ *returns the position of the smallest value in the interval of T starting on position i and ending on the position j: $T[i], T[i+1], \ldots, T[j]$). In case of multiple possible answers, we assume that* min*pos returns the rightmost position where the smallest value in the interval is found.*

The usual form of the p-RMQ problem takes $p = n$, and is simply called RMQ.

Several solutions of the RMQ problem are presented in [6]. The most efficient of them (proposed in [10]) requires $\mathcal{O}(n)$ preprocessing time, $\mathcal{O}(n)$ space to store the constructed data structures, and $\mathcal{O}(1)$ time to answer each query; it can also be applied to solve the p-RMQ problem. However, for the purposes of this paper, we will use a different solution of the p-RMQ problem, which requires $\mathcal{O}(n \log p)$ preprocessing time, $\mathcal{O}(n \log p)$ space to store the data structures constructed, and $\mathcal{O}(1)$ time to answer each query. This solution consists in the following strategy:

• We define the matrix M with n rows and $\log_2 p$ columns, where:

$$M[i][k] = \text{minpos}_T(i - 2^k + 1, i), \text{ for } 1 \leq i \leq n \text{ and } 0 \leq k \leq \min(\log_2 i, \log_2 p).$$

Moreover, we assume that the elements $M[i][k]$ are undefined for other pairs i and k than the ones mentioned in the above formula.

• The elements of this matrix can be computed by dynamic programming, using the following recurrence formula, for all i and $k \leq \min(\log_2 i, \log_2 p)$:

$$M[i][k] = \begin{cases} i, \text{ if } k = 0, \\ M[i][k-1], \text{ if } T[M[i][k-1]] \leq T[M[i-2^{k-1}][k-1]] \text{ and } k \geq 1, \\ M[i-2^{k-1}][k-1], \text{ if } T[M[i][k-1]] > T[M[i-2^{k-1}][k-1]] \text{ and } k \geq 1. \end{cases}$$

• After M is computed we can answer minpos queries for all i and j, such that $i \leq j \leq i+p-1$ and $k = \lfloor \log_2(j-i+1) \rfloor$ (here $\lfloor z \rfloor$ denotes the integer part of the positive real number z):

$$\text{minpos}_T(i,j) = \begin{cases} M[i+2^k-1][k], \text{ if } T[M[i+2^k-1][k]] < T[M[j][k]], \\ M[j][k], \text{ if } T[M[i+2^k-1][k]] \geq T[M[j][k]]. \end{cases}$$

It is clear that all the elements of M can be computed in time $\mathcal{O}(n \log p)$, using the above relation, and that the answer to each query can be obtained in time $\mathcal{O}(1)$.

Now, consider the following update operation of the array T: we add an element x to T, on the position $n+1$; in the following we will denote this operation by $Add(T, x)$. For simplicity, we consider that this operation can be performed in constant time, no memory allocation being needed; basically, this assumption is justified by the fact that in the algorithms we present here the number of updates we perform on an array is bounded by a value depending on the input size. Thus, we can allocate enough memory for all the arrays that we use and update from the beginning of the algorithm execution.

We are interested in updating the matrix M in order to be able to answer minpos queries, defined in the p-RMQ problem, for the newly obtained array. To this aim we make the update as follows:

• A new row is added to the matrix M, namely the row $M[n+1][k]$, with $k \leq \min(\log_2(n+1), \log_2 p)$.

• The elements of this row are computed using the following formula, for $k \leq \min(\log_2(n+1), \log_2 p)$:

$$M[n+1][k] = \begin{cases} \text{minpos}_T(n-2^k+2, n), \text{ if } T[\text{minpos}_T(n-2^k+2,n)] < T[n+1], \\ n+1, \text{ otherwise.} \end{cases}$$

Clearly, after M is updated we can still answer minpos queries for the updated array T in the same manner as described above.

Also, as in the case of updating the array T, we may assume that adding a new row to the matrix M takes the same time as that needed to compute the row (or, in other words, we do not need to care about how to reallocate the matrix M or how to address its elements, after it is updated). Therefore, the total time needed to update the matrix M is $\mathcal{O}(\log p)$.

For a more general discussion on the $p-$RMQ problem see [7].

5.2 Computing the p-Bounded k-Hairpin Distance

In the following we propose an algorithm that computes the hairpin completion distance between the two words x and w. Assume that $n = |w| \geq |x|$; it is clear that x cannot be obtained by p-bounded k-hairpin completion from w.

First, similarly to the algorithm described in [16], we compute the $n \times n$ matrix P defined by:

$$P[i][j] = \begin{cases} \max(\{t \mid w[i..i+t-1] = \overline{w[j-t+1..j]^R}\} \cup \{0\}), i < j \\ 0, \text{ otherwise} \end{cases}$$

This matrix can easily be computed in time $\mathcal{O}(n^2)$, by dynamic programming, using the following recurrence relation:

$$P[i][j] = \begin{cases} P[i+1][j-1]+1, \text{ if } i < j \text{ and } w[i] = \overline{w[j]} \\ 0, \text{ otherwise} \end{cases}$$

Let $Compute\_matrix(P, w)$ be a procedure implementing the above recurrence relation (defined in [16]):

Algorithm 1

```
procedure  $Compute\_matrix\ (P, w)$;
begin
1.for  $i = 1$ to $n$
2.    for  $j = 1$ to $n$
```

```
3.        $P[i][j] := 0$
4.    endfor
5.endfor
6.for  $l = 2$ to $n$
7.    for  $i = 1$ to $n - l + 1$
8.        $j := i+l-1;$
9.        if $w[i] = \overline{w[j]}$ then $P[i][j] := P[i+1][j-1] + 1$
10.   endfor
11.endfor
end.
```

Further, we follow the approach from [16] and define a $n \times n$ matrix M, with $H[i][j] = pHCD_k(x, w[i..j])$. The main idea in computing this matrix is to have the following conditions fulfilled for all $1 \leq i, j \leq n$:

- Initially $H[i][j] = +\infty$.
- If, at some point of the execution of the algorithm computing the matrix H, we set $H[i][j] = t$, for some natural number $t \neq +\infty$, then $pHCD_k(x, w[i..j]) = t$. Also, once we set $H[i][j]$ to a value $t \neq +\infty$ we will never change it.

The algorithm computing the matrix H is based on a dynamic programming strategy, and works as follows: we analyze all the subwords of w, in increasing order of their length; during this process, we identify all the subwords $w[i..j]$ of w that can be obtained by p-bounded k-hairpin completion from x, we compute the p-bounded k-hairpin completion distance between x and such a subword, and save it as $H[i][j]$.

In order to decide if $w[i..j]$ can be obtained by iterated p-bounded k-hairpin completion from x we simply have to check if $w[i..j] = x$, or if one of the following two conditions hold: there exists a number s such that $w[i..s] \in pHC_k^*(x)$ and $w[i..j] \in pHC_k(w[i..s])$, or there exists a number t such that $w[t..j] \in pHC_K^*(x)$ and $w[i..j] \in pHC_k(w[t..j])$. Further, if $w[i..j] \in pHC_k^*(x)$, we need to compute the distance between x and $w[i..j]$, namely $H[i][j]$. If $x = w[i..j]$ then, clearly, this distance is equal to 0, i.e., we set $H[i][j] = 0$. Otherwise, we search for an index s such that $w[i..s] \in pHC_k^*(x)$, $w[i..j] \in pHC_k(w[i..s])$, and the distance between x and $w[i..s]$ is less than the distance between x and any other proper prefix of $w[i..j]$ that can be transformed by (one-step) p-bounded k-hairpin completion into $w[i..j]$; also, we search for an index t such that $w[t..j] \in pHC_k^*(x)$, $w[i..j] \in pHC_k(w[t..j])$, and the distance between x and $w[t..j]$ is less than the distance between x and any other proper suffix of $w[i..j]$ that can be transformed by (one-step) p-bounded k-hairpin completion into $w[i..j]$. Once the indices s and t are computed we set $H[i][j] = \min(H[i][s], H[t][j]) + 1$. Note that the dynamic programming strategy ensures the fact that when we compute $H[i][j]$, the distances between x and any proper factor of $w[i..j]$, including here the values $H[i][s]$ and $H[t][j]$ needed in the above formula, were already correctly computed.

To implement efficiently the above strategy, differently from [16], we use four arrays of size n, $right$, $Cright$, $left$ and $Cleft$, such that the following conditions are satisfied, during the execution of the algorithm:

- Initially, we assume that all these arrays are void (or, alternatively, that the n values stored in them are set to a special value $+\infty$).
- For all i, with $1 \leq i \leq n$, $right[i]$ contains, at any moment of the execution, all the numbers s, $1 \leq s \leq n$, identified by the algorithm until that moment, such that $w[i..s] \in pHC_k^*(x)$; moreover, these numbers are ordered increasingly. Whenever a new number s with $w[i..s] \in pHC_k^*(x)$ is identified, it is added to the array $right[i]$, in a manner that preserves the order of the elements in the array. We simply denote by $|right[i]|$ the number of elements contained at a given moment in the array $right[i]$.
- At any moment of the execution, for all i with $1 \leq i \leq n$, $Cright[i]$ contains $|right[i]|$ entries. Moreover, $Cright[i][j]$ (i.e., the element on the j^{th} position of $Cright[i]$) equals d if and only if $right[i][j]$ (i.e., the number on the j^{th} position in the array $right[i]$) equals s, and d is the p-bounded k-hairpin completion distance between x and $w[i..s]$.
- For all j, with $1 \leq j \leq n$, $left[j]$ contains, at any moment of the execution, all the numbers t, $1 \leq t \leq n$, identified by the algorithm until that moment, such that $w[t..j] \in pHC_k^*(x)$; moreover, these numbers are ordered decreasingly. Whenever a new number t with $w[t..j] \in pHC_k^*(x)$ is identified, it is added in the array $left[j]$, in a manner that preserves the order of the elements in the array. We simply denote by $|left[j]|$ the number of elements contained at a given moment in the array $left[j]$
- At any moment of the execution, for all i with $1 \leq j \leq n$, $Cleft[j]$ contains $|left[j]|$ entries. Also, $Cleft[j][i] = d$ if and only if $left[j][i] = t$ and d is the m-bounded k-hairpin completion distance between x and $w[t..j]$.

The dynamic programming strategy ensures that every time a new subword $w[i..j] \in pHC_k^*(x)$ is identified, we simply have to add j at the end of $right[i]$ and i at the end of $left[j]$, since these arrays contain only elements smaller than j, respectively greater than i. The arrays $Cright$ and $Cleft$ are also updated accordingly: we add at their ends the numbers representing the hairpin completion distance between x and $w[i..j]$.

The computation of the distance between x and a subword $w[i..j]$ goes on as follows:

- If $w[i..j] = x$ then $d_1 = d_2 = 0$.
- If $j - i + 1 > |x|$ let s be a natural number s such that $j - p \leq s < j$ and $s \geq j - P[i][j] + k$ (i.e., $w[i..s]$ is a subword from which we can obtain $w[i..j]$ by p-bounded k-hairpin completion), $w[i..s] \in pHC_k^*(x)$ and $pHCD_k(x, w[i..s]) \leq pHCD_k(x, w[i..s'])$ for all the number s' such that $j - p \leq s' < j$ and $s' \geq j - P[i][j] + k$(i.e., the distance between x and $w[i..s]$ is less or equal to the distance between x and any other prefix of $w[i..j]$ that can be completed to obtain this subword). If such a number s exists, we set $d_1 = pHCD_k(x, w[i..s])$, otherwise we set $d_1 = +\infty$.
- If $j - i + 1 > |x|$ let t be a natural number s such that $i + p \geq t > i$ and $t \leq i + P[i][j] - k$ (i.e., $w[t..j]$ is a subword from which we can obtain $w[i..j]$ by p-bounded k-hairpin completion), $w[t..j] \in pHC_k^*(x)$ and $pHCD_k(x, w[t..j]) \leq pHCD_k(x, w[t'..j])$ for all the number t' such that

$i+p \geq t' > i$ and $t' \leq i+P[i][j]-k$(i.e., the distance between x and $w[t..j]$ is less or equal to the distance between x and any other suffix of $w[i..j]$ that can be completed to obtain this subword). If such a number t exists, we set $d_2 = pHCD_k(x, w[i..s])$, otherwise we set $d_2 = +\infty$.
– Take $pHCD_k(x, w[i..j]) = \min(d_1, d_2)$.

It remains to clarify only the way we find the numbers s and t used above. In this respect, note that the existance of a number s as above is equivalent to the existance of a a number l such that $0 \leq l \leq |right[i]|$, $s = right[i][l] \geq \max(j - P[i][j] + 1, j - p)$ and $Cright[i][l] \leq Cright[i][l']$ for all l' such that $right[i][l'] \geq \max(j-P[i][j]+1, j-p)$; in this case, $d_1 = Cright[i][l]$. Equivalently, the existance of a number t as above is equivalent to the existance of a a number l such that $0 \leq l \leq |left[j]|$, $t = left[j][l] \leq \min(i + P[i][j] - 1, i + p)$ and $Cleft[j][l] \leq Cleft[j][l']$ for all l' such that $left[i][l'] \leq \min(i+P[i][j]-1, i+p)$; in this case, $d_2 = Cleft[j][l]$.

These conditions suggest the exact way we should proceed when computing the minimum number of iterated p-bounded k-hairpin completion steps needed to be applied to x for obtaining $w[i..j]$:

– Assume that $Cright$ and $Cleft$ are processed such that we can answer minpos queries (as in p-RMQ problem) for them; also, recall that $right$ and $left$ are ordered increasingly and, respectively, decreasingly.
– Search (using binary search) the minimum index l which verifies $0 \leq l \leq |right[i]|$ and $right[i][l] \geq \max(j - P[i][j] + k, j - p)$. When a value l as above is found, set $d_1 = \text{minpos}_{Cright[i]}(l, |right[i]|)$.
– Search (using binary search) the minimum index l which verifies $\max(0, |left[j]| - p) \leq l \leq |left[j]|$ and $left[j][l] \leq \min(i + P[i][j] - k, i + p)$. When a value l as above is found, set $d_2 = \text{minpos}_{Cleft[j]}(l, |left[j]|)$.
– If none of the above searches returns a valid index l, then we conclude that $w[i..j] \notin pHC_k^*(x)$. Otherwise, it follows that $w[i..j] \in pHC_k^*(x)$ and $pHCD_k(x, w[i..j]) = d = \min(d_1, d_2) + 1$. In this case, we add j to $right[i]$ and i to $left[j]$; then, we add t to the arrays $Cright[i]$ and $Cleft[j]$ (i.e., $Cright[i][|right[i]|] = Cleft[j][|left[j]|] = t$). Once these arrays are updated, we also update, as discussed in the previous section, the data structures constructed to answer minpos queries for them.

It is clear that the computation described above takes $\mathcal{O}(\log_2 p)$ time, the most time consuming steps being the two binary searches and the update of the data structures needed to answer minpos queries for the arrays $Cright$ and $Cleft$.

Now we can state the algorithm computing the p-bounded k-hairpin distance between two words x and w.

Algorithm 2

```
function $Hairpin\_completion\_distance(p, k, x, w)$;
begin
1. if $x = w$ then return $0$;
2. if $(\min(|x|, |w|) < 2k + 1))$ or $(|x| = |w|$ and $x \neq w)$ then return $+\infty$;
```

```
3. if $|x| > |w|$ then interchange $x$ with $w$; set $n = |w|$; (now we have $|x| < |w| = n$);
4. for $1 \leq i \leq n$ allocate memory for the arrays $left[i], right[i], Cleft[i], Cright[i]$ with
   at most $n$ positions each; initially we assume that all these arrays have $0$ elements;
   also, allocate the $n \times n$ matrices $H$ and $P$;
5. for $1 \leq i \leq n$ allocate memory for (and initialize) the structures needed to answer
   minpos queries for the arrays $Cright[i]$ and $Cleft[i]$;
6. set $H[i][j] = +\infty$ for all $i, j \in \{1, \dots, n\}$;
7. locate all the occurences of $x$ in $w$; if $w[i..j] = x$ then set $H[i][j] = 0$;
8. $Compute\_matrix(P, y)$;
9. for $l = |x|$ to $n$
10.    for $i = 1$ to $n - l + 1$
11.      set $j := i + l - 1$;
12.      if $H[i][j] = 0$ then $Add(left[i], i), Add(right[i], j)$, $Add(Cright[i], 0)$,
           $Add(Cleft[j], 0)$, process the updated $Cright$ and $Cleft$ in order to answer
           minpos queries for them, set $t = 0$ and go to 14;
13.      compute the minimum number $d$ of $p$-bounded $k$-hairpin completion steps
           needed to obtain $w[i..j]$ from $x$ and update the data structures used to
           answer minpos queries for $Cright$ and $Cleft$ (as described above);
14.      set $H[i][j] = d$;
15.    endfor;
16. endfor;
17. return $H[1][n]$;
end.
```

From the above explanations it is clear that the algorithm just presented computes correctly the minimum number of p-bounded k-hairpin completion steps that can be applied to x for obtaining w. To compute the complexity of the algorithm note that steps $1 - 4$ and 6 require $\mathcal{O}(n^2)$ time to be completed; also step 7 can be completed in $\mathcal{O}(n)$ using the Knuth-Morris-Pratt algorithm (see [15]). Step 5 takes $\mathcal{O}(n^2 \log p)$ time, as we have explained in the Data Structures section; step 8 can be completed in time $\mathcal{O}(n^2)$. Finally, steps $11 - 14$ are executed for n^2 times, and they consume at most $\mathcal{O}(\log p)$ time. Thus, steps $9 - 16$ take $\mathcal{O}(n^2 \log p)$ time. In conclusion, the whole algorithm is executed in time $\mathcal{O}(n^2 \log p)$. Since the data structures computed to answer minpos queries need the largest storage space, we can easily deduce that the space used in the above algorithm is $\mathcal{O}(n^2 \log p)$.

In conclusion, we have proved:

Theorem 4 *The p-bounded k-hairpin completion distance between two words x and w can be computed in $\mathcal{O}(n^2 \log p)$ time and $\mathcal{O}(n^2 \log p)$ space, where $n = \max(|x|, |w|)$.*

It is worth mentioning that this algorithm can easily be generalized to compute the unbounded k-hairpin completion distance between two words x and w. Assume, as in the previous considerations that $|x| \leq |w|$. Then, if x can be transformed by k-hairpin completion into w, then the length of the words added in the hairpin completion operations applied at each step is bounded by $|w|$. Thus computing the unbounded k-hairpin completion distance between x and w is the same as computing the n-bounded k-hairpin completion distance between x and w (where $n = \max(|x|, |w|)$), and takes $\mathcal{O}(n^2 \log n)$ time. This improves the cubic time bound shown in [16].

6 A Generalization of the Bounded Hairpin Completion Distance

In the following we consider a generalization of the problem presented in the previous section. Namely, given two regular languages L_1 and L_2 we define the p-bounded k-hairpin completion distance between them: $pHCD_k(L_1, L_2)$ equals the minimum number t such that $pHC_k^t(L_1) \cap L_2 \neq \emptyset$ or $L_1 \cap pHC_k^t(L_2) \neq \emptyset$. More precisely:

$$pHCD_k(L_1, L_2) = \begin{cases} \min\{d \mid pHC_k^d(L_1) \cap L_2 \neq \emptyset \text{ or } L_1 \cap pHC_k^d(L_2) \neq \emptyset\}, \\ \infty, \text{ if } pHC_k^*(L_1) \cap L_2 = L_1 \cap pHC_k^*(L_2) = \emptyset \\ 0, \text{ if } L_1 = L_2 = \emptyset \end{cases}$$

As in the case of the function $pHCD_k$ defined for words, the above function, applied on pairs of languages, is not is not a distance function in the strict mathematical sense. In this case, it does not necessarily verify the triangle inequality; also $pHCD_k(L_1, L_2) = 0$ does not imply that $L_1 = L_2$. Once again, we prefer to call it distance by the same reason mentioned when we introduced the bounded hairpin distance between two words.

In the sequel we show how this distance can be computed, assuming that both L_1 and L_2 are given by two deterministic finite automata M_1 and M_2, respectively. We stress out that a similar generalization was not done in the case of unbounded k-hairpin completion.

Note that it is enough to show how we can compute the minimum number d_1 such that $pHC_k^{d_1}(L_1) \cap L_2 \neq \emptyset$. Once this value was computed, we can proceed in the same manner to compute the minimum number d_2 such that $pHC_k^{d_2}(L_2) \cap L_1 \neq \emptyset$, and return $pHCD_k(L_1, L_2) = \min(d_1, d_2)$.

Let us first set some notations. Assume that $L_1 \cup L_2 \subseteq V^*$. Let $M_1 = (Q, V, q_0, F, \delta)$ be a deterministic finite automaton accepting L_1; let $M_2 = (P, V, p_0, E, \tau)$ be a deterministic finite automaton accepting L_2. We may assume without loss of generality that $Q \cap P = \emptyset$, $Q = \{q_0, q_1, \ldots, q_n\}$ and $P = \{p_0, p_1, \ldots, p_m\}$.

Our aproach is based on the idea to explore all the possible words that can be obtained by hairpin completion from L_1 and see if they are contained in L_2. However, these words can be infinitely many. Therefore, we will look for all the possible pairs of states from M_2 than can be connected by words obtained by iterated hairpin completion from the words in L_1; in order to find these pairs we need to store (and explore) only the parts of words obtained by hairpin completion that are important for the p-bounded k-hairpin completion, i.e., the first $p+k$ and the last $p+k$ symbols of each word. Thus, we will try to generate all the possible tuples fromed a prefix and a suffix of length $p+k$ of the words obtained by iterated hairpin completion from L_1, and the pairs of states connected by such a word in M_2. To do this, we will use as data structures three queues: D, D_f and D_w. The first one, D, will store all the tuples $[p_1, p_2, w, i]$ such that $p_1, p_2 \in P$, $\tau(p_1, w) = p_2$, $w \in V^*$ with $|w| < 2(p+k)$, and w can be obtained from a word in L_1 by iterated p-bounded k-hairpin completion, in i steps. Further, D_f stores all the tuples $[p_1, p_2, w, i]$ such that $p_1, p_2 \in P$, $\tau(p_1, w) = p_2$, and w is a word that can be obtained from a word contained in a tuple from D, by applying once the p-bounded k-hairpin completion, but $|w| \geq 2(p+k)$; i has the same meaning as above. Intuitively, D stores information about all the words that can be derived by hairpin completion from L, can connect a pair of states from M_2 that is not contained in $\{p_0\} \times E$, and have length shorter than $2(p+k)$ (i.e., the words from which we cannot extract a prefix and a suffix of length $p+k$, that do not overlap), while D_f stores information about all the words that can be derived by hairpin completion from L, have length greater or equal to $2(p+k)$, but they can be obtained from words shorter than $2(p+k)$, or have length shorter than $2(p+k)$, but are part of L_2 already. Of course, both D and D_f can contain a finite number of elements, that depend on m, k and p. Finally, D_w contains all the tuples $[p_1, p_2, w_1, w_2, i]$ such that $p_1, p_2 \in P$, and there exists a word $w = w_1 x w_2$, $w_1, x, w_2 \in V^*$, $|w_1| = |w_2| = p+k$, $\tau(p_1, w) = p_2$, and w is a word that can be obtained by iterated p-bounded k-hairpin completion from the words in L_1 in i steps. D_w contains information on all the possible tuples fromed by pairs of non-overlapping prefix and a suffix of length $p+k$ of the words obtained by iterated hairpin completion from L_1, and the pairs of states connected by such a word in M_2; clearly, D_w will contain a finite number of elements, depending on m, p and k. These queues will be used as follows: we search for the minimum i such that a tuple containing i, the state p_0 and a final state of M_2, appears in one of the queues, and we set $d_1 = i$. It is convenient to implement these queues as priority queues, ordered according to the value of i.

Our strategy can be described more formally as follows.

In order to compute d_1 we first check if $L_1 \cap L_2 \neq \emptyset$ or if $L_1 = L_2 = \emptyset$. If the answer is positive, we have $d_1 = 0$. Otherwise, we generate the set $L_1' = \{w \in L_1 \mid |w| \leq 2(p+k)\}$, and initialize two empty queues D and D_f. Then we add into the queue D all the elements of the set $\{[p_1, p_2, w, 0] \mid p_1, p_2 \in P, w \in L_1', \tau(p_1, w) = p_2\}$. After this, we repeat the following procedure:

– Extract the first element $[p_1', p_2', w', i]$ of the queue D and generate all the elements $[p_1'', p_2'', w'', i+1]$ such that one of the following hold:
  – $w'' = \alpha\beta x \overline{\beta^R \alpha^R}$, $w' = \beta x \overline{\beta^R \alpha^R}$, $0 < |\alpha| \leq p$, $|\beta| = k$, $\tau(p_1'', \alpha) = p_1'$ and $p_2'' = p_2'$,
  or
  – $w'' = \alpha\beta x \overline{\beta^R \alpha^R}$, $w' = \alpha\beta x \overline{\beta^R}$, $0 < |\alpha| \leq p$, $|\beta| = k$, $\tau(p_2', \overline{\alpha^R}) = p_2''$ and $p_1'' = p_1'$.

– From the elements generated in the above step, insert into D the elements $[p_1'', p_2'', w'', i+1]$, with $|w| < 2(p+k)$ and $(p_1'', p_2'') \notin \{p_0\} \times E$, that were not previously present in D; all the other elements, that are not contained already in D_f, are inserted into D_f. Note that both D and D_f have their elements ordered increasingly with respect to their last component, which equals the number of hairpin completion steps in which the word present on the third component of each element, respectively, was obtained from a word in L_1.

It is clear that the queue D will become empty in a finite number of steps. Indeed, the tuples that may appear in D are finitely many. But, at each step, we delete one tuple from the queue, and a tuple cannot appear more than once in D. It follows that the process is finite. The number of steps performed until the queue D becomes empty is $\mathcal{O}(n^2)$, but the constant hidden by the $\mathcal{O}$-denotation is proportional with $|V|^{2(p+k)}$.

Then, we generate all the elements $[p_1, p_2, w_1, w_2, 0]$ such that $p_1, p_2 \in P$, $|w_1| = |w_2| = p+k$, and there exists $x \in V^*$ such that $w_1 x w_2 \in L_1$ and $\tau(p_1, w_1 x w_2) = p_2$. These elements can easily be computed as follows:

– Find all the tuples (q_1', q_2', p_1', p_2') such that there exists $x \in V^*$ and $\delta(q_1', x) = q_2'$ and $\tau(p_1', x) = p_2'$. This step takes $\mathcal{O}(n^2 m^2)$ time, and the constant hidden by the $\mathcal{O}$-denotation is proportionally with $|V|$.
– Store, from the tuples computed above, all tuples (q_1', q_2', p_1', p_2') for which there exists $w_1, w_2 \in V^{p+k}$ such that $\delta(q_0, w_1) = q_1'$ and $\delta(q_2', w_2) \in F$. This step takes $\mathcal{O}(n^2)$ time, but the constant hidden by the $\mathcal{O}$-denotation is proportionally with $|V|^{2(p+k)}$. Then, construct the elements $(p_1, p_2, w_1, w_2, 0)$ with p_1 such that $\tau(p_1, w_1) = p_1'$ and $p_2 = \tau(p_2', w_2)$. This step takes $\mathcal{O}(m^2)$ time, but the constant hidden by the $\mathcal{O}$-denotation is, once again, proportionally with $|V|^{2(p+k)}$.

It is not hard to see that this procedure generates the tuples $[p_1, p_2, w_1, w_2, 0]$ defined above. Further, we add these elements to the queue D_w. Again, it is clear that this process is finite, and the number of computational steps that are needed to compute the queue D_w is bounded by $\mathcal{O}(n^2 m^2)$, where the constant hidden by the $\mathcal{O}$-denotation is proportional to $|V|^{2(p+k)}$.

Once we have computed D_w and D_f we repeat the following steps, until a value for d_1 is set:

– If $[p_1, p_2, w_1, w_2, t]$ is the first element of D_w, $[p_1', p_2', w', t']$ is the first element of D_f, $(p_1, p_2) \notin \{p_0\} \times E$, and $t < t'$, or if D_f is empty, then extract $[p_1, p_2, w_1, w_2, t]$ from D_w and construct the elements $[p_3, p_4, w_3, w_4, t+1]$, where:
– $w_1 = \alpha x \beta$, $w_2 = \gamma \overline{x^R}$, $|x| = k$, $w_3 = w_1$, w_4 is formed by the last $p+k$ symbols of $w_2 \overline{\alpha^R}$, $p_3 = p_1$ and $p_4 = \tau(p_2, \overline{\alpha^R})$.
– $w_1 = x\beta$, $w_2 = \gamma \overline{x^R} \alpha$, $|x| = k$, $w_4 = w_2$, w_3 is formed by the first $p+k$ symbols of $\overline{\alpha^R} w_1$, $p_4 = p_2$ and $p_1 = \tau(p_3, \overline{\alpha^R})$.
From these elements we insert in D_w the elements $[p_3, p_4, w_3, w_4, t+1]$ for which no other element of the form $[p_3, p_4, w_3, w_4, l]$ has ever been in D_w.

– If $[p_1, p_2, w_1, w_2, t]$ is the first element of D_w, $[p'_1, p'_2, w', t']$ is the first element of D_f, $(p_1, p_2) \in \{p_0\} \times E$, and $t < t'$, or D_f is empty, then t_1, the minimum number such that $pHC_k^{d_1}(L_1) \cap L_2 \neq \emptyset$, equals t.
– If $[p_1, p_2, w_1, w_2, t]$ is the first element of D_w, $[p'_1, p'_2, w', t']$ is the first element of D_f, $(p'_1, p'_2) \in \{p_0\} \times E$, and $t \geq t'$, or D_w is empty, then t_1, the minimum number such that $pHC_k^{d_1}(L_1) \cap L_2 \neq \emptyset$, equals t'.
– If $[p_1, p_2, w_1, w_2, t]$ is the first element of D_w, $[p'_1, p'_2, w', t']$ is the first element of D_f, $(p'_1, p'_2) \notin \{p_0\} \times E$, and $t \geq t'$, or D_w is empty, then we have $|w'| \geq 2(p + k)$. It follows that $w' = w'_1 x w'_2$ with $|w'_1| = |w'_2| = p + k$. We construct the elements $[p_3, p_4, w_3, w_4, t + 1]$, where:
  - $w'_1 = \alpha x \beta$, $w'_2 = \gamma \overline{x^R}$, $|x| = k$, $w_3 = w'_1$, w_4 is formed by the last $p + k$ symbols of $w'_2 \overline{\alpha^R}$, $p_3 = p_1$ and $p_4 = \tau(p_2, \overline{\alpha^R})$.
  - $w'_1 = x\beta$, $w'_2 = \gamma \overline{x^R} \alpha$, $|x| = k$, $w_4 = w'_2$, w_3 is formed by the first $p + k$ symbols of $\overline{\alpha^R} w'_1$, $p_4 = p_2$ and $p_1 = \tau(p_3, \overline{\alpha^R})$.

  From these elements we insert in D_w the elements $[p_3, p_4, w_3, w_4, t + 1]$ for which no other element of the form $[p_3, p_4, w_3, w_4, l]$ has ever been in D_w.
– If the both queues are empty then $d_1 = +\infty$.

The procedure just described can only be repeated for a finite number of times, bounded by $\mathcal{O}(m^2)$ (the constant hidden by the $\mathcal{O}$ is proportional to $|V|^{2(p+k)}$), since an element having p_1, p_2, w_1, and w_2 on its first four positions, respectively, can only appear once in D_w. Also, the procedure computes correctly the minimum number d_1 such that $pHC_k^{d_1}(L_1) \cap L_2 \neq \emptyset$, following the idea that we eplained in the beginning. Indeed, it basically goes through all the possible pairs of states $(p_1, p_2) \in P^2$ that can be connected by a word $w \in pHC_k^t(L_1)$, in increasing order with respect to t. Once we find a pair (p_0, e), with $e \in E$, such that there exists a word $w \in pHC_k^{d_1}(L_1)$ for which $\tau(p_0, w) = e$, we can return d_1 as the minimum number such that $pHC_k^{d_1}(L_1) \cap L_2 \neq \emptyset$, since all the pairs of states that can be connected by words from $pHC_k^t(L_1)$, for $t < t_1$, were already analyzed. If the algorithm returns $+\infty$, then, clearly, $pHC_k^*(L_1) \cap L_2 = \emptyset$, because no pair of states (p_0, e), with $e \in E$, that can be connected by a word $w \in pHC_k^*(L_1)$, was identified.

In conclusion, given the regular languages L_1, accepted by the deterministic finite automaton M_1 with n states, and L_2, accepted by the deterministic finite automaton M_2 with m states, one can compute the minimum number d_1 such that $mHC_k^{d_1}(L_1) \cap L_2 \neq \emptyset$ (if $t_1 = +\infty$ then $mHC_k^*(L_1) \cap L_2 = \emptyset$), in time $\mathcal{O}(n^2m^2)$. We conclude that one can compute $pHCD_k(L_1, L_2)$ in time $\mathcal{O}(n^2m^2)$. The algorithm presented here can be implemented in $\mathcal{O}(n^2m^2)$ space as well. In both cases, the constants hidden by the $\mathcal{O}$-denotation depend on $|V|^{2(p+k)}$.

Theorem 5 *The p-bounded k-hairpin completion distance between two words regular languages L_1 and L_2 can be computed in $\mathcal{O}(n^2m^2)$ time and space, given that L_1 is accepted by a finite automaton with n states and L_2 is accepted by a finite automaton with m states. The constants hidden by the $\mathcal{O}$-denotation depend on $|V|^{2(p+k)}$.*

7 An inverse operation: the bounded hairpin reduction

We now define the inverse operation of the bounded hairpin completion, namely the bounded hairpin reduction in a similar way to [18], where the unbounded hairpin reduction is introduced. Let V be an alphabet, for any $w \in V^+$ we define the *p-bounded k-hairpin reduction* of w, denoted by $pHR_k(w)$, for some $k, p \geq 1$, as follows:

$$pHR\circlearrowleft_k(w) = \{\alpha\beta\overline{\alpha^R\gamma^R} \mid w = \gamma\alpha\beta\overline{\alpha^R\gamma^R}, |\alpha| = k,\ \alpha, \beta, \gamma \in V^+, 1 \leq |\gamma| \leq p\},$$
$$pHR\circlearrowright_k(w) = \{\gamma\alpha\beta\overline{\alpha^R} \mid w = \gamma\alpha\beta\overline{\alpha^R\gamma^R}, |\alpha| = k,\ \alpha, \beta, \gamma \in V^+, 1 \leq |\gamma| \leq p\}.$$
$$pHR_k(w) = pHR\circlearrowleft_k(w) \cup pHR\circlearrowright(w).$$

The *p-bounded hairpin reduction* of w is defined by

$$pHR(w) = \bigcup_{k\geq 1} pHR_k(w).$$

The bounded hairpin reduction is naturally extended to languages by

$$pHR_k(L) = \bigcup_{w\in L} pHR_k(w) \qquad pHR(L) = \bigcup_{w\in L} pHR(w).$$

The iterated bounded hairpin reduction is defined analogously to the iterated bounded hairpin completion.

We recall that the problem of whether or not the iterated unbounded hairpin reduction of a regular language is recursive is left open in [18]. The same problem for the iterated bounded hairpin reduction is now completely solved by the next more general result. Before stating the result, we need to recall a few notions about string-rewriting systems. To this aim, we follow the standard notations for string rewriting as in [2]. A string-rewriting system (SRS) over an alphabet V is a finite relation $R \subset V^* \times V^*$, and the rewrite relation induced by R is denoted by $\longrightarrow_R$. That is, we write $x \longrightarrow_R y$ if $x = uvw, y = uzw$, for some $u, v, z, w \in V^*$, and $(v, z) \in R$. As usual every pair $(v, z) \in R$ is referred to as a rule $v \to z$. The reflexive and transitive closure of $\longrightarrow_R$ is denoted by $\longrightarrow_R^*$. We use $R^*(L)$ for the closure of the language L under the string-rewriting system R. Formally, $R^*(L) = \{w \mid x \longrightarrow_R^* w,$ for some $x \in L\}$. A rule $v \to z$ is said to be *monadic* if it is length-reducing ($|v| > |z|$) and $|z| \leq 1$. A SRS is called monadic if all its rules are monadic. A class of languages $\mathcal{F}$ is closed under monadic SRS if for any language $L \in \mathcal{F}$ over some alphabet V and any monadic SRS R over V, $R^*(L) \in \mathcal{F}$ holds.

Theorem 6 *Every trio closed under circular permutation and monadic string-rewriting systems is closed under iterated bounded hairpin reduction.*

Proof. Let $\mathcal{F}$ be a trio and k, p be two positive integers. The central idea of the proof is as follows. We permute every word of a language in $\mathcal{F}$ in a circular way. Then the last and first letters are next to each other. Thus the hairpin reduction becomes a local operation and can be simulated by monadic string-rewriting rules. By our hypothesis, these are known to preserve the membership in $\mathcal{F}$.

To start with, we attach a new symbol X to the end of every word of a given $L \in \mathcal{F}$, $L \subseteq V^*$. Then we obtain the language L' by doing a circular permutation to all words in $L\{X\}$. Note that X marks the end and beginning of the original words. On this language we apply a *gsm*-mapping g that introduces redundancy by adding to every letter information about its neighboring letters in the following way:

1. The letter containing the X contains also the $k+p$ letters to the left and to the right of X in order.
2. Every letter left of X contains the letter originally at that position and the $k+p$ letters left of it in order.
3. Every letter right of X contains the letter originally at that position and the $k+p$ letters right of it in order.

At the word's end and its beginning, where there are not enough letters to fill the symbols, some special symbol signifying a space is placed inside the compound symbols.

Now we can simulate a step of p-bounded k-hairpin reduction by a string-rewriting rule with a right-hand side of length one, i.e. a monadic one. A straightforward approach would be to use rules of the form $u\overline{v}^R X v\overline{u}^R \to uXv\overline{u}^R$. But we see that u and $Xv\overline{u}^R$ are basically not changed, they only form a context whose presence is necessary. Through our redundant representation of the word, their presence can be checked by looking only at the corresponding image of X under g. Further, since the symbols of the image of u under g contain only information about symbols to their left, they do not need to be updated after the deletion of $\overline{v}^R$ to preserve the properties 1 to 3. The same is true for $v\overline{u}^R$. Only in the symbol corresponding to X some updating needs to be done and thus it is the one that is actually rewritten. So the string rewriting rules are

$$g_{\mathsf{left}}(z_0 z_1 u\overline{v}^R)[1\ldots|v|][z_1 u\overline{v}^R X v\overline{u}^R z_2] \to [z_0 z_1 u X v\overline{u}^R z_2],$$

where g_{left} does the part of g described by property 2, and where $z_0, z_1 \in V^*$, $u, v \in V^+$, $|u| = k$, $|v| \leq p$, $|z_0 z_1 u| = p+k$. Analogously, rules that delete symbols to the right of X are defined. Let R be the string-rewriting system consisting of all such rules. It is immediate that $w' \in pHR_k(w) \;\Leftrightarrow\; g(cp(wX)) \to_R g(cp(w'X))$ and by induction $w' \in pHR_k^*(w) \;\Leftrightarrow\; g(cp(wX)) \to_R^* g(cp(w'X))$.

Therefore, at this point we have all circular permutations of words that can be reached by p-bounded k-hairpin reduction from words in L coded under g. To obtain our target language we first undo the coding of g by the gsm-mapping g' that projects all letters to the left of X to their last component, all letters to the right of X to their first component, and the symbol containing X to just X. This mapping is letter-to-letter, the gsm only needs to remember in its state whether is has already passes over the symbol containing X. Of the result of this we take again the circular permutation.

Now we filter out the words that have X at the last position and therefore are back in the original order of L and delete X. By the closure properties of $\mathcal{F}$, the result of this process lies in $\mathcal{F}$, which completes the proof. $\square$

As monadic SRSs are known to preserve regularity (see [11]) we immediately infer that

Theorem 7 *The class of regular languages is closed under iterated bounded hairpin reduction.*

In [17] one considers another concept that seems attractive to us, namely the *primitive hairpin root* of a word and of a language. Given a word $x \in V^*$ and a positive integer k, the word y is said to be the primitive k-hairpin root of x if the following two conditions are satisfied:

$$\begin{array}{rl} (i) & y \in HR_k^*(x)(\text{or, equivalently, } x \in HC_k^*(y)), \\ (ii) & HR_k(y) = \emptyset. \end{array}$$

Here $HR_k^*(z)$ delivers the iterated unbounded hairpin reduction of the word z. In other words, y can be obtained from x by iterated k-hairpin reduction (maybe in zero steps) and y cannot be further reduced by hairpin reduction. The primitive bounded hairpin root is defined analogously. Clearly, a word may have more than one primitive bounded hairpin root; the set of all primitive p-bounded k-hairpin roots of a word x is denoted by $pH_k root(x)$. Naturally, the primitive p-bounded k-hairpin root of a language L is defined by $pH_k root(L) = \bigcup\limits_{x \in L} pH_k root(x)$.

Clearly, to see whether a word is reducible, one has to look only at the first and last $k + p$ symbols. By Theorem 7 we have:

Theorem 8 *$pH_k root(L)$ is regular for any regular language L and any $p, k \geq 1$.*

Proof. We compose the root of two sets. First, there are all the words in L that are too short to be reduced and form the set

$$\{w \in L \mid |w| \leq 2k + 2\}.$$

To these words we have to add the ones that can be reached from words in L via hairpin reduction and cannot be reduced any further. By Theorem 7 the former condition can be checked by regular means, and the latter can be checked by looking at the first and last $k + p$ symbols of the words in $pHR_k^*(L)$. More precisely, we take the set

$$(pHR_k^*(L) \cap \{\alpha x \beta \mid |\alpha| = |\beta| = k + 1, \alpha \neq \overline{\beta}^R, x \in V^+\}).$$

Both of these sets are regular and their union equals $pH_k root(L)$. $\square$

8 Final remarks

We have considered a restricted version of the hairpin completion operation by imposing that the prefix or suffix added by the hairpin completion are bounded by a constant. In some sense, this is the lower extreme case the upper extreme being the unbounded case that might be viewed as a linearly bounded variant. We consider that bounded variants by other sublinear mappings would be of theoretical interest.

Last but not least we would like to mention that hairpin completion and reduction resemble some language generating mechanisms considered in the literature like external contextual grammars with choice [20] or dipolar contextual deletion [12], respectively.

References

1. Baker, B.S: Context-sensitive grammars generating context-free languages. In Automata, Languages and Programming ICALP 1972. North-Holland, Amsterdam (1972) 501–506.
2. Book, R., Otto, F.: String-Rewriting Systems. Springer-Verlag (1993).
3. Bottoni, P., Labella, A., Manca, V., Mitrana, V.: Superposition based on Watson-Crick-like complementarity. Theory of Computing Systems (39) (2006) 503–524.
4. Cheptea, D., Martín-Vide, C., Mitrana, V.: A new operation on words suggested by DNA biochemistry: hairpin completion. In Transgressive Computing (2006) 216–228.
5. Deaton, R., Murphy, R., Garzon, M., Franceschetti, D.R., Stevens, S.E.: Good encodings for DNA-based solutions to combinatorial problems. In Proc. of DNA-based computers II. DIMACS Series (44) (1998) 247–258.
6. Demaine, E., Weimann, O.: Advanced Data Structures. Lecture Notes from MIT, Lecture 15, http://courses.csail.mit.edu/6.851/spring07/lec.html, 2007.
7. Diaconu, A., Manea, F., Tiseanu, C.: Combinatorial Queries and Updates on Partial Words. In Proc. FCT 2009. LNCS 5699 (2009) 96 - 108.
8. Garzon, M., Deaton, R., Neathery, P., Murphy, R.C., Franceschetti, D.R., Stevens, S.E.: On the encoding problem for DNA computing. In The Third DIMACS Workshop on DNA-Based Computing. Univ. of Pennsylvania (1997) 230–237.
9. Garzon, M., Deaton, R., Nino, L.F., Stevens, S.E., Wittner, M.: Genome encoding for DNA computing. In Proc. Third Genetic Programming Conference. Madison, MI (1998) 684–690.
10. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. (13) (1984) 338–355.
11. Hofbauer, D., Waldmann, J.: Deleting string-rewriting systems preserve regularity. Theoretical Computer Science (327) (2004) 301–317.
12. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Information and Computation (131) (1996) 47–61.
13. Kari, L., Konstantinidis, S., Sosík, P., Thierrin, G.: On hairpin-free words and languages. In Proc. Developments in Language Theory 2005. LNCS 3572 (2005) 296–307.
14. Knuth, D.E.: The Art of Computer Programming, Volume I: Fundamental Algorithms. Addison-Wesley (1968).

15. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal of Computing (6) (1977) 323–350.
16. Manea, F., Martín-Vide, C., Mitrana, V.: On some algorithmic problems regarding the hairpin completion. Discrete Applied Mathematics (157) (2009) 2143–2152.
17. Manea, F., Mitrana, V.: Hairpin completion versus hairpin reduction. In Computation in Europe CiE 2007. Vol. 4497 of LNCS. Springer-Verlag (2007) 532–541.
18. Manea, F., Mitrana, V., Yokomori, T.: Two complementary operations inspired by the DNA hairpin formation: Completion and reduction. Theoretical Computer Science (410) (2009) 417–425.
19. Manea, F., Mitrana, V., Yokomori, T.: Some remarks on the hairpin completion. In Proc. 12th International Conference on Automata and Formal Languages (2008) 302–313.
20. Marcus, S.: Contextual grammars. Rev. Roum. Math. Pures Appl. (14) (1969) 1525–1534.
21. Rozenberg, G., Salomaa, A. (Eds.): Handbook of Formal Languages. Springer-Verlag (1997).
22. Ruohonen, K.: On circular words and $(\omega^* + \omega)$-powers of words. Elektr. Inform. und Kybern. E.I.K. (13) (1977) 3–12.
23. Sakamoto, K., Gouzu, H., Komiya, K., Kiga, D., Yokoyama, S., Yokomori, T., Hagiya, M.: Molecular computation by DNA hairpin formation. Science (288) (2000) 1223–1226.
24. Salomaa, A.: Formal Languages. Academic Press (1973).

Report of Changes

Besides correcting and improving numerous small details found by the referees and ourselves, the substantial changes and additions that have been introduced are the following:

Proofs: More explanatory text and detail have been added to the proofs of Proposition 1, Corollary 1, Theorem 2, and Theorem 7 as asked for by referee number two.

Sections 5 and 6: We have explained in more details the algorithms presented in these Sections. All the suggestions of the referees were implemented.