Testing Generalised Freeness of Words*

Paweł Gawrychowski[1], Florin Manea[2], and Dirk Nowotka[2]

1 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
gawry@cs.uni.wroc.pl
2 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
{flm,dn}@informatik.uni-kiel.de

Abstract

Pseudo-repetitions are a natural generalisation of the classical notion of repetitions in sequences: they are the repeated concatenation of a word and its encoding under a certain morphism or antimorphism (anti-/morphism, for short). We approach the problem of deciding efficiently, for a word w and a literal anti-/morphism f, whether w contains an instance of a given pattern involving a variable x and its image under f, i.e., $f(x)$. Our results generalise both the problem of finding fixed repetitive structures (e.g., squares, cubes) inside a word and the problem of finding palindromic structures inside a word. For instance, we can detect efficiently a factor of the form xx^Rxxx^R, or any other pattern of such type. We also address the problem of testing efficiently, in the same setting, whether the word w contains an arbitrary pseudo-repetition of a given exponent.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Stringology, Pattern matching, Repetition, Pseudo-repetition

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.337

1 Introduction

A word is a *repetition* if it can be written as a repeated concatenation of one of its prefixes to itself. A word w is a *pseudo-repetition* if it can be written as a repeated concatenation of one of its prefixes t and its image $f(t)$ under some morphic or antimorphic function f (for short, an "anti-/morphism" f), thus $w \in t\{t, f(t)\}^+$.

The concept of pseudo-repetitions (introduced in [4]) draws its original motivations from two important biological concepts: tandem repeat, i.e., the consecutive repetition of the same sequence of nucleotides, and the inverted repeat, i.e., a sequence of nucleotides whose reversed complement (or, Watson-Crick complement) occurred already in the longer DNA sequence we analyse, both occurrences (the original one and the complemented one) encoding, essentially, the same genetic information. Noting that the Watson-Crick complement can be abstracted as an antimorphic involution on the DNA-alphabet, pseudo-repetitions formalise generalised tandem repeats, in which one sequence is followed by several consecutive occurrences of either its copy or of its reversed complement.

Other situation in which one encounters pseudo-repetitions appears in musical theory. The repetition of some fragment, in its initial form but also slightly modified (like the initial fragment on a higher or lower pitch), are used to provide unity to a musical piece. For instance, the *ternary (song) form* appears frequently: three consecutive musical fragments

* P. Gawrychowski is supported by the *NCN* grant 2011/01/D/ST6/07164, F. Manea by the *DFG* grant 596676, D. Nowotka by the *DFG Heisenberg* grant 590179.

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 337–349

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

such that the first and third ones are identical, while the second one is constructed in order to provide a contrast to the other two (and, sometimes, this can be seen as the image of the other two parts under some simple anti-/morphism).

Note that both in biology and music, the function applied to obtain the pseudo-repetition is structurally simple: it usually rewrites each letter (nucleotide or note) into another one (i.e., it is literal), and usually does not rewrite more letters into the same one (i.e., it is bijective). Besides the two examples above, in which pseudo-repetitions occur, the concept seems to be of intrinsic theoretical interest, as it generalises a combinatorics on words concept (that is, repetitions) that is central both in theory and applications. If we consider palindromes as a natural modification of squares, repetitive structures containing both normal occurrences of some factor and mirrored occurrences of the same factor seem to be one of the most natural, hence, interesting, concepts derived from the classical repetitions.

The results obtained so far on pseudo-repetitions were both of combinatorial [4, 12, 13] and of algorithmic [6, 7, 14] nature. We continue here the study of algorithmic problems related to this concept. More precisely, we study how efficiently can one decide, given an input word w, a literal (in some cases, bijective) anti-/morphism f, and a pattern involving both a variable x and its image under f, namely $f(x)$, whether an input word contains as a factor an instance of that pattern. The problem seems natural to us, both in the light of the combinatorial questions on the avoidability of patterns under anti-/morphisms, raised in [13], as well as in the respect that it generalises both the well-studied problems of efficiently deciding whether a word contains k-repetitions or, alternatively, various palindromic structures.

The paper is structured as follows. We begin with a series of basic definitions, then we overview our results and compare them to the existing literature, and conclude with two technical sections, containing the proofs of our results.

Some Basic Concepts. For more detailed definitions we refer to [2].

Let V be a finite alphabet; V^* is the set of all words over V, and the *empty word* is denoted by λ. The *length* of a word $w \in V^*$ is denoted by $|w|$, while $\mathrm{alph}(w)$ denotes the set of all letters that occur in w. In our problems, we assume that the letters of any word w of length n are in fact integers from $\{1, \ldots, n\}$; accordingly, w is a sequence of integers. This is a common assumption in stringology (see, e.g., [8]).

If $w = xuy$, for $w, x, u, y \in V^*$, then u is a *factor* of w, x is a *prefix* of w, and y is a *suffix* of w. For $1 \leq i \leq j \leq |w|$, we denote by $w[i]$ the symbol at position i in w and by $w[i..j]$ the factor $w[i]w[i+1]\cdots w[j]$ of the word w. A word u occurs in w at position i if u is a prefix of $w[i..|w|]$. The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$. A word w that is not a power of some other word is called *primitive*. If w is not primitive, then there exists a unique primitive word u, called the primitive root of w, such that $w = u^n$ for some $n \geq 2$. A *period* of a word w over V is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod{p}$; $per(w)$ denotes the smallest period of w.

A function $f : V^* \to V^*$ is a morphism if $f(xy) = f(x)f(y)$ for all $x, y \in V^*$; f is an antimorphism if $f(xy) = f(y)f(x)$ for all $x, y \in V^*$. To define an anti-/morphism it is enough to define $f(a)$, for all $a \in V$. We call f *literal* if $|f(a)| = 1$ for all $a \in V$. Let $f(V) = \{f(a) \mid a \in V\}$. We assume that any literal anti-/morphism f is given as the image of the letters of V under f (in order, from 1 to $|V|$).

We say that a word w is an *f-repetition* with root t or, alternatively, an f-power with root t, if w is in $\{t, f(t)\}^+$, for a factor t of w. If w is not an f-power, then w is *f-primitive*. For example, the word *abcaab* is primitive from the classical point of view (i.e., **1**-primitive, where **1** is the identical morphism) as well as g-primitive for the morphism g defined by

$g(a) = b$, $g(b) = a$, $g(c) = c$. However, when considering the morphism $f(a) = c$, $f(b) = a$, $f(c) = b$, we get that $abcaab = ab\, f(ab)\, ab$, thus, being an f-repetition with root ab.

For an anti-/morphism $f : V^* \to V^*$, a unary f-pattern p is an element of the set $\{x, f(x)\}^*$ (i.e., a word over the alphabet having the letters x and $f(x)$); here x is called variable and, if $p \in \{x, f(x)\}^k$, we say that k is the length of p. An instance of the f-pattern p is a word obtained by replacing in it the variable x by a word $t \in V^+$ and evaluating the resulting expression in V^*. For instance, if f is the identity antimorphism (i.e., f is $(\cdot)^R$, the mirror image) then $xf(x) = xx^R$ is a pattern whose instances are all palindromes of even length; if f is the identity morphism then $xf(x) = x^2$ is a pattern whose instances are all squares. We will only discuss the case of unary f-patterns, called patterns for brevity.

Finally, the computational model we use is the standard unit-cost RAM with logarithmic word size. Also, all logarithms appearing here are in base 2.

2 The problems

In [6], the problem of identifying, given a word w and an anti-/morphism f, all the f-repetitions (either of arbitrary or of given exponent) contained in w was efficiently solved. The solutions and their time complexities depended on the properties of f, but, in all cases, they were influenced by the fact that we needed to output *all* f-repetitions.

Here we approach two related problems. We are given a word w, an anti-/morphism f, just like before, but also a unary f-pattern p. We want to test whether w has as factor

an instance of the given pattern p (i.e., whether w is p-free). Further, in the same setting, but given a number k instead of the pattern, we are interested in testing whether the word w contains an f-repetition of exponent k, that is, a word of the form $\{t, f(t)\}^k$. This task was called in [1] the problem of testing pseudo-kth-power freeness. Compared to the first problem, this one requires deciding whether w contains an instance of *any* pattern of length k.

The results we present deal with the cases when f is a literal morphism or antimorphism; moreover, in part of the results we restrict f to being bijective. These cases seem to be interesting as they cover exactly the classes of morphisms and antimorphisms that play an important role in the literature: the classical mirror image of words, the antimorphic involutions used in the initial papers on pseudo-repetitions [1, 4, 14] to model the DNA Watson-Crick complementarity, or the anti-/morphic permutations used in [13] in the context of avoiding pseudo-repetitions or in [12] to show generalised periodicity lemmas.

The two problems are defined formally in the following.

▶ Problem 1. Given $w \in V^+$, $|w| = n$, $f : V^* \to V^*$ a literal anti-/morphism, and an f-pattern p of length k, decide whether there exists an instance of p occurring in w.

▶ Problem 2. Given $w \in V^+$, $|w| = n$, $f : V^* \to V^*$ a literal anti-/morphism, and an integer $k > 0$, decide whether there exists a factor v of w with $v \in \{t, f(t)\}^k$ for some $t \in V^+$.

Our results are the following. We first give, in Section 4, solutions to Problem 1 and 2 that run in $\mathcal{O}(nk^2)$ time when f is both a literal morphism or a literal antimorphism. This is especially interesting as it shows that Problem 1 can be solved in linear time for f-patterns of fixed length (or Problem 2 for constant k). For instance, the occurrence of patterns having length 2 (generalised squares) or 3 (cubes under literal anti-/morphisms), inside a word can be tested in linear time. The solutions we present here rely both on several combinatorics on words results and on the possibility of constructing efficient data structures for word processing. It is worth noting that our approach begins similarly to that used in [11] to detect classical repetitions; however, the arguments used further in our algorithms and proofs are more general and required both a deeper analysis and novel techniques. Moreover, the

tools we developed may have a broader range of applications. For instance, Lemmas 3, 4, or 7, provide novel insight in the combinatorics of pseudo-repetitions, while Lemma 2 shows how an extended Lempel-Ziv-like factorisation of a word can be efficiently computed, which might be useful in, e.g., efficiently detecting inverted repeats in DNA sequences.

In Section 5, we consider the problems for f bijective. As already described, this case played an important role in the existing theory and its motivations. In this setting we propose new solutions for Problems 1 and 2, running in $\mathcal{O}(n \log n)$ time. While being based on a different approach than the previous ones, these solution have also the nice feature that they are efficient even for larger values of k. Moreover, the solution of Problem 2 can be used to compute the maximum k such that w contains an instance of a pattern of length k.

Before concluding this section, let us note that our results improve significantly the results reported in [14], [1], and [6]. In [14] it was shown that factors of the form $t^{k-1}f(t)$ or $(tf(t))^k$ (and symmetrical) can be detected in $\mathcal{O}(kn)$ time; in [1] these results were extended to show that that pseudo-cube freeness can be tested in $\mathcal{O}(n^2)$ time. Both these results were developed for f an antimorphic involution and an alphabet of constant size. In [1], within the same setting, an algorithm outputting all the pseudo-k-powers contained in a word in $\mathcal{O}(n^2 \log n)$ time was reported, and used to test pseudo-kth-power freeness; a faster algorithm finding all the pseudo-k-powers occurring in a word in time $\Theta(n^2)$ was given in [6] (for f literal, and working over integer alphabets, as here), but it was still slower than our algorithms when used to test pseudo-kth-power freeness. For patterns without functional dependencies, Problems 1 and 2 coincide, and can be solved in linear time (see, e.g., [11]).

3 Prerequisites

For a word u, $|u| = n$, over $V \subseteq \{1, \ldots, n\}$ we can build in $\mathcal{O}(n)$ time the suffix tree and suffix array structures, as well as data structures allowing us to retrieve in constant time the length of the longest common prefix of any two suffixes $u[i..n]$ and $u[j..n]$ of u, denoted $LCP_u(i, j)$ (the subscript u is omitted when there is no danger of confusion). Such structures are called LCP data structures in the following. For details, see, e.g., [8], and the references therein. Similarly, we can build structures allowing us to retrieve in constant time the length of the longest common suffix of any two prefixes $u[1..i]$ and $u[1..j]$ of u, denoted $LCS_u(i, j)$.

▶ Remark 1. Given a word w of length n and a divisor ℓ of n we can use one LCP query to check in constant time whether $w = x^k$, where $|x| = \ell$ and $k = \frac{n}{\ell}$. Indeed, $w = x^k$ if and only if $LCP(1, \ell + 1) = n - \ell$. The longest prefix of w that is a power of $u[1..\ell]$ is obtained similarly, as the longest prefix w' of w whose length is divisible by ℓ and $|w'| \leq LCP(1, \ell + 1)$.

When solving Problems 1 and 2 we construct LCP data structures for the words w, w^R (the mirror image of w), and $v = wf(w)$; this takes $\mathcal{O}(|w|)$ time. Note that, when f is a literal morphism, checking whether $f(w[i..j])$ appears at position ℓ in w is equivalent to checking whether $\ell + j - i \leq n$ and $LCP_v(\ell, |w| + i) \geq j - i + 1$. When f is a literal antimorphism, checking whether $f(w[i..j])$ appears at position ℓ in w is, in this case, equivalent to checking whether $\ell + j - i \leq n$ and $LCP_v(\ell, 2|w| - j + 1)) \geq j - i + 1$.

In some of the proofs we will need an efficient solution for the *interval union-find* problem, which asks to maintain a partition of the universe $U = [1, n]$ into a number of disjoint intervals, so that given any element we can locate its current interval, and we can merge two currently adjacent intervals into one. Both operations can be implemented in amortised constant time [5] in our model of computation.

The following classical combinatorial result is used in this paper; for proofs and details see [10], the handbook [2, Chapter 9], and the references therein.

▶ **Lemma 1.** *Let $w \in V^*$, with $|w| = n$, and $PS_w = \{u \text{ primitive} \mid u^2 \text{ prefix of } w\}$. Then $|PS_w| \leq 2 \log n$ and one can compute all the sets $PS_{w[i..n]}$, for $1 \leq i \leq n$, in $\mathcal{O}(n \log n)$ time.*

4 General solutions

The main result we show in this section is that both our problems can be solved in $\mathcal{O}(nk^2)$ time, so in linear time for constant k. That is, testing freeness with respect to a fixed f-pattern or testing pseudo-kth-power freeness for constant k can be done in linear time. These results seem highly interesting to us as they show that the efficiency of testing square, cube, or palindrome freeness is preserved for a class of more general patterns.

We begin this section with a slightly modified version of the s-factorisation defined in [11] (see also [2]), and series of lemmas on the newly defined concept. Let $g : V^* \to V^*$ be a literal anti-/morphism and $w \in V^*$ a word. The g-factorisation of w is defined as follows. We factor $w = u_1 \cdots u_r$ if the following hold for all $i \geq 1$:

- If letter a occurs in w immediately after $u_1 \cdots u_{i-1}$ and neither a nor $g(a)$ appeared in $u_1 \cdots u_{i-1}$, then $u_i = a$.
- Otherwise, u_i is the longest word such that $u_1 \cdots u_{i-1}u_i$ is a prefix of w and u_i or $g(u_i)$ occurs at least once as a factor in $u_1 \cdots u_{i-1}$.

The **1**-factorisation of w (the not-self-referential variant of the s-factorisation from [11]) is obtained by just taking g to be **1**, the identity morphism.

By arguments similar to those used in [3], it follows that g-factorisations of words can be computed in linear time, for g literal anti-/morphism.

▶ **Lemma 2.** *If g is a literal anti-/morphism we can compute the g-factorisation of a word w of length n in time $\mathcal{O}(n)$.*

The following combinatorial lemma shows the relation between possible occurrences of an f-pattern p in a word and its f-factorisation, for a morphism f.

▶ **Lemma 3.** *Let f be a literal morphism, w a word, and p a pattern of length $k \geq 2$, such that $p \neq x^{k-1}f(x)$. Let $w = u_1 \cdots u_r$ be the f-factorisation of w and consider all instances of p. Then for any instance $w[i..j]$ with $|u_1 \cdots u_{h-1}| < j \leq |u_1 \cdots u_h|$ we have two mutually exclusive possibilities:*
1. *$i > |u_1 \cdots u_{h-1}|$, and we call $w[i..j]$ a* secondary *instance, completely contained in u_h,*
2. *$j - i + 1 \leq k(|u_{h-1}| + |u_h|)$, and we call $w[i..j]$ a* crossing *instance.*
Furthermore, the leftmost instance of the pattern is crossing.

A related result can be shown also for f antimorphic, but in a slightly more particular setting: the word w is **1**-factored and the pattern p has at least length 3.

▶ **Lemma 4.** *Let f be a literal anti-/morphism, w a word, and p a pattern of length $k \geq 3$, such that $p \notin \{x^{k-1}f(x), f(x)^{k-1}x\}$. Let $w = u_1 \cdots u_r$ be the **1**-factorisation of w and consider all instances of the pattern p. Then for any such instance $w[i..j]$ with $|u_1 \cdots u_{h-1}| < j \leq |u_1 \cdots u_h|$ we have two mutually exclusive possibilities:*
1. *$i > |u_1 \cdots u_{h-1}|$, and $w[i..j]$ is a secondary instance, completely contained in u_h,*
2. *$j - i + 1 \leq k(|u_{h-1}| + |u_h|)$, and $w[i..j]$ is a crossing instance.*
Furthermore, the leftmost instance of the pattern is crossing.

We only present here the solutions of Problems 1 and 2 for f antimorphic. The morphic case can be solved in a similar manner with less technicalities. For Problem 1, for instance, we compute the f-factorisation of the input word w, and then we try to identify (like in

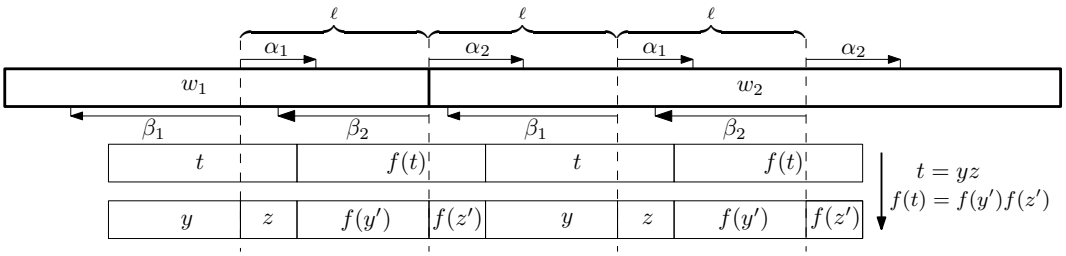

Figure 1 Finding $tf(t)tf(t)$ in the catenation of two words.

the antimorphic case, described below) the leftmost occurrence, if any, of an instance of the pattern p, which is crossing by Lemma 3. The usage of f-factorisations is needed here so that the case of patterns of length 2 (or f-squares) is also covered. The time complexity of this approach is $\mathcal{O}(nk^2)$.

The case of Problem 1, for antimorphisms. When f is an antimorphism we use **1**-factorisations of the input words, instead of f-factorisations. The major points of the algorithm detecting instances of the pattern f are given in the following.

In the following, a pseudopalindrome of length ℓ is a word of the form $uf(u)$ with $|u| = \ell$. Its occurrence is centred at position i if the first character of $f(u)$ is aligned there. The pseudopalindromic radius at i is simply the length of the longest pseudopalindrome centred at i, which can be computed in constant time using LCS queries on $wf(w)$.

To begin with, patterns of the form $xf(x)$ or $f(x)x$ are detected in $\mathcal{O}(n)$ time by checking the existence of positions with strictly positive pseudopalindromic radius in w and, respectively, w^R. Moreover, if the pattern is x^k, the problem reduces to detecting the usual repetitions, hence can be done in $\mathcal{O}(n)$ time. If the pattern is $f(x)^k$, we just need to check which letters can be an image under f, so which factors of w can contain an instance of the pattern, and again we can reduce the problem to the standard case of detecting repetitions.

▶ **Lemma 5.** *Let f be a literal antimorphism, w be a* **1***-factorised word, and p a pattern of length $k \geq 3$, $p \notin \{x^{k-1}f(x), f(x)x^{k-1}, f(x)^{k-1}x, xf(x)^{k-1}, x^k, f(x)^k\}$. All M crossing instances of p (if any) can be detected (and output as pairs of indices) in $\mathcal{O}(nk^2 + M)$ time.*

Proof. The proof is based on Lemma 4. Assume that the **1**-factorisation of w is $u_1 \cdots u_r$ and for each $h \leq r$ we look for an instance of p ending inside u_h, shorter than $k(|u_{h-1}| + |u_h|)$.

We describe how to find an instance v of a given pattern p of length k, in the concatenation of two words w_1 and w_2, such that this occurrence crosses the boundary; then we can apply the strategy for $w_2 = u_h$ and w_1 a suffix of $u_1 \cdots u_{h-1}$, $|w_1| = \min\{|u_1 \cdots u_{h-1}|, k(|u_{h-1}u_h|) - 1\}$. For each such w_1 and w_2 we construct a list of simple conditions which guarantee the existence of an occurrence. Then we consider all such conditions together, and verify them in a specific order. For simplicity, we only explain the case of $v = tf(t)tf(t)$ (an instance of $p = xf(x)xf(x)$), with $k = 4$, and then show how to generalise. Let us also assume that the first letter of w_2 belongs to the first $f(t)$, see Figure 1; the other cases are analogous. Let ℓ denote $|t|$ and assume that we built LCP and LCS data structures for $w' = w_1w_2f(w_1)f(w_2)$.

Now, let α_1 be the length of the longest factor starting at positions $|w_1| - \ell + 1$ in w_1 and ℓ in w_2; let α_2 be the length of the longest factor starting at positions 1 and $2\ell + 1$ in w_2. Similarly, let β_1 be the length of the longest factor ending at positions $|w_1| - \ell$ in w_1 and $\ell - 1$ in w_2, and β_2 be the length of the longest factor ending at positions $|w_1|$ in w_1 and 2ℓ in w_2. All $\alpha_1, \alpha_2, \beta_1, \beta_2$ can be computed with $LCP_{w'}$ and $LCS_{w'}$ queries. Now,

if $\min\{\alpha_1, \alpha_2\} + \min\{\beta_1, \beta_2\} < \ell$, clearly no instance exists. Otherwise, the only possible instances have $|z| \leq \min\{\alpha_1, \alpha_2\}$ and $|z| \geq \ell - \min\{\beta_1, \beta_2\}$ (where z is defined as in Figure 1). Moreover, those two conditions guarantee that all fragments corresponding to t are the same, and all fragments corresponding to $f(t)$ are equal, too. They do not guarantee, though, that the fragment which is supposed to be $f(t)$ is indeed an image of the fragment corresponding to t. Hence we also need to check if the pseudopalindromic radius at $|w_1| - \ell + |z|$ in w_1w_2 is at least ℓ. That is, an instance of the pattern corresponds exactly to a pseudopalindrome of length ℓ centred at $i \in [|w_1| - \min\{\beta_1, \beta_2\}, |w_1| - \ell + \min\{\alpha_1, \alpha_2\}]$ in w_1w_2, if any.

We build a list of all such conditions corresponding to different values of $\ell \leq \frac{|w_1|+|w_2|}{k}$ (and different pairs of concatenated words w_1 and $w_2 = u_h$, as described in the beginning) and then process them all at once in the order of increasing ℓ. This is done efficiently by maintaining in a structure S all positions i such that the pseudopalindromic radius at i is at least the current ℓ. Then, reporting all instances corresponding to a single condition requires iterating through all $i \in S$ such that i is in the corresponding range. Note now that S can be implemented as the interval union-find structure. Then, if the range contains H numbers, we output them in $\mathcal{O}(1 + H)$ time. We check similarly all the other possibilities for a factor t or $f(t)$ to fall on the border, so the claimed complexity (for $p = xf(x)xf(x)$) follows.

Other patterns of length 3 and 4 are analysed similarly. If the pattern is longer (of length $k > 4$), we check k possibilities for the factor falling on the border. For each of them we need to execute $\mathcal{O}(k)$ constant time queries to generate the conditions on $|y|$ or detect that no such instance is possible. Hence the total number of conditions is $\sum_{2 \leq h \leq r} \sum_{\ell \leq |u_{h-1}|+|u_h|} k = \mathcal{O}(nk)$, and the total time to generate all of them is $\mathcal{O}(nk^2)$, plus additional $\mathcal{O}(n)$ to maintain the interval union-find structure. Then each instance is reported in constant time. ◀

Assume first that $p \notin \{x^{k-1}f(x), f(x)x^{k-1}, f(x)^{k-1}x, xf(x)^{k-1}, x^k, f^k(x)\}$. Lemma 4 shows that if p occurs in w, then there exists a crossing instance of p in w. The previous lemma shows that we can locate in $\mathcal{O}(nk^2)$ time such a crossing instance of p, if any (just output the first instance we meet and stop searching for others). If we found one, we conclude that w contains an instance of the pattern; otherwise, w does not contain any instance of p.

Further, we only have to consider now the cases when p is $x^{k-1}f(x)$ or $xf(x)^{k-1}$; the other cases can be reduced to these two by looking for the occurrences of p^R in w^R. In the first (respectively, second) remaining case, we only need to check if there is a position i such that the pseudopalindromic radius at i is at least as long as the length of the shortest word whose k-th power is a suffix (respectively, prefix) of $w[1..i-1]$ (respectively, $w[i..n]$). Thus, to conclude, we apply the following lemma for w (respectively, w^R).

▶ **Lemma 6.** *Given a word w of length n and $k \geq 2$, we can compute for each position i the smallest $\ell \geq 1$ such that $w[i - k\ell + 1..i]$ is a power of $w[i - \ell + 1..i]$, in $\mathcal{O}(n)$ total time.*

This technical lemma and its main consequence, that we can detect instances of the patterns $x^{k-1}f(x)$ and $f(x)^{k-1}x$ in $\mathcal{O}(n)$ time, improves significantly the results reported in [14]: we decreased the complexity from $\mathcal{O}(nk)$, and our algorithm works for integer alphabets.

The case of Problem 2, for antimorphisms. Let us first give the following lemma:

▶ **Lemma 7.** *If w contains $\max(k, 3)$ pseudopalindromes of length ℓ starting at positions $s, s + \delta_1, s + \delta_2, \ldots$ with all $\delta_i \leq \frac{\ell}{4}$, then w has a factor r^k with $r = f(r)$. Accordingly, w contains an instance of any pattern of length k.*

To solve Problem 2, we begin with checking if there is an instance of x^k, $f^k(x)$, $f(x)^{k-1}x$, $xf(x)^{k-1}$, $x^{k-1}f(x)$, or $f(x)x^{k-1}$ using the method from the previous section. If there is

no such instance, we apply the reasoning described in Lemma 5. Of course, now we do not know the exact structure of the pattern. Nevertheless, we can look at the **1**-factorisation $w = u_1 \cdots u_r$, and for each two adjacent factors u_{h-1} and u_h we consider all possibilities for the length ℓ of t, the image of the variable x. Assume that for each such possibility we get that an instance of the pattern corresponds to the existence of a pseudopalindrome of length ℓ centred in a range of at most $2\ell k$ positions, these positions being the suffix of w_1 of length ℓk and the prefix of w_2 of length ℓk, with w_1 and $w_2 = u_h$ defined as in the proof of Lemma 5. We generate all pseudopalindromes in such ranges using the same approach as in that lemma, i.e., by considering the entire set of conditions at once and maintaining an interval union-find structure. If we can generate sufficiently many results, we terminate, as Lemma 7 shows that w contains instances of all patterns of length k. More concretely, for each range of length $2\ell k$ there can be at most $8k^2$ pseudopalindromes of length ℓ centred there, provided that no word r^k with $r = f(r)$ exists in w. Also, $\ell \leq |u_{h-1}| + |u_h|$ (by Lemma 4). So, summing up over all h and ℓ we get that the total number of generated pseudopalindromes should not exceed $\sum_{2 \leq h \leq r} \sum_{\ell \leq |u_{h-1}|+|u_h|} 8k^2 = 16nk^2$. Thus, assume that we generated at most $16nk^2$ pseudopalindromes. Each of them corresponds to certain values of ℓ and i such that the instance of $xf(x)$ is centred at i and the image of x has length ℓ. So, we can check in $\mathcal{O}(k)$ time whether the image of $xf(x)$ can be extended to an instance of a pattern of length k. For this we do not need to know the exact structure of this pattern, we just check that how many fragments of length ℓ are the same as either the left or the right half of the found pseudopalindrome with LCP queries. This gives an $\mathcal{O}(nk^3)$ time solution.

We can shave off one factor of k by using dynamic programming. We consider all generated pseudopalindromes with the same value of ℓ at once. For each such pseudopalindrome $vf(v)$, we compute the largest k' such that there is a k-repetition with $x = v$ starting with this $vf(v)$. The idea is that after any $vf(v)$ we must have a power of v or $f(v)$ followed by another $vf(v)$ (or by the end of the instance of the pattern). Hence with a constant number of LCP queries we can compute the highest power of both v and $f(v)$ following this $vf(v)$, and then check if the next fragment of length 2ℓ is $vf(v)$ as well. Then the total running time becomes proportional to the number of generated pseudopalindromes, which is $\mathcal{O}(nk^2)$.

5 The bijective case

An $\mathcal{O}(n \log n)$ solution for Problem 1. The second solution we propose for Problem 1 is based on a careful analysis of the length 3 factors that may occur in the pattern, and is valid, in this case, both for f literal bijective morphism and antimorphism.

We first check whether the pattern p contains any of the factors $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$. This takes $\mathcal{O}(k) \subset \mathcal{O}(n)$ time. If not, then the pattern is a prefix of x^∞, $f(x)^\infty$, $(xf(x))^\infty$, or of $(f(x)x)^\infty$. We treat separately each of these cases.

First, let us note that if p is a prefix of x^∞ then Problem 1 is equivalent to testing whether w contains k-repetitions. This can be done in $\mathcal{O}(n)$ time, using the algorithm given by Main [11]. When p is a prefix of $f(x)^\infty$ the Problem 1 can be solved similarly, in linear time, by detecting repetitions in the factors of w containing only letters from $f(V)$.

▶ **Lemma 8.** *Testing whether a word w, $|w| = n$, has as factor an instance of a pattern p, $|p| = k$, that contains $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, can be done in $\mathcal{O}(kn \log n)$ time.*

Proof. We analyse now the cases when p contains $xxf(x)$ or $f(x)f(x)x$ (the cases when p contains $f(x)xx$ or $xf(x)f(x)$ are solved by considering the reversed word and reversed pattern). Hence, any instance of p contains a square. We analyse the first occurrence of a

factor $xxf(x)$ or $f(x)f(x)x$ in p and check whether it is possible that such a factor occurs at position i in the word w, for all $i \in \{1, \ldots, n\}$. To this end, we consider such a number i; by Lemma 1, there are at most $2 \log n$ primitive squares occurring at position i in w. Clearly, when $vvf(v)$ is a word that occurs at position i then v is a power of some $y \in PS_{w[i..n]}$. Thus, we iterate through all the elements of $y \in PS_{w[i..n]}$ and see whether we can construct a word v such that $vvf(v)$ or $f(v)f(v)v$ occurs at position i in w and v is a power of y.

Let us assume that $xxf(x)$ occurs before any occurrence of $f(x)f(x)x$ in p (the other case can be treated similarly). We have two subcases to analyse. The first subcase is when $y = f(y)$ (and this can be checked in constant time). Then we compute the maximum q such that y^q is a prefix of $w[i..n]$; this takes $\mathcal{O}(1)$ time, as $q = \frac{LCP_w(i,i+|y|)}{|y|} + 1$. Similarly, using an LCP_{w^R} query, we compute the maximum ℓ such that y^ℓ is a suffix of $w[1..i]$. If $q \geq 3$ we take $v = y$ and, clearly, an instance of the pattern p occurs in w whenever $\ell + q \geq k$, where $k = |p|$. The second subcase is when $y \neq f(y)$. Just like before, we compute the maximum q such that y^q is a prefix of $w[i..n]$. Now, if q is even and $f(y)$ occurs at position $1 + q|y|$, we have $v = y^{\frac{q}{2}}$; if q is odd then v cannot be a power of y. Once we know v we check whether an instance of p occurs in w such that its first factor $xxf(x)$ is mapped to the factor $vvf(v)$ occurring at position i in w. This check is done in $\mathcal{O}(k)$ time, using $LCP_{wf(w)}$ queries.

If $f(x)f(x)x$ occurs first in p (before $xxf(x)$) then one should also analyse, for each y, two subcases just like before. If $f(v)f(v)v$ occurs at position i then $f(v)$ is a power of some $y \in PS_{w[i..n]}$. The first subcase is when $y = f(y)$, and can be treated just as the case $y = f(y)$ above. The second subcase is when we assume that $y = f(z)$ for some $z \neq y$. As f is injective, we have $y \neq f(y)$. Thus, z is the first factor of length $|y|$ that occurs after the maximal prefix y^p of $w[i..n]$. We then check by an $LCP_{wf(w)}$ query whether $f(z) = y$ and p is even, and follow the same procedure as before, for v starting with z and of length $\frac{p|y|}{2}$. ◀

In conclusion, the case when p contains an occurrence of the factor $xxf(x)$ or of $f(x)f(x)x$ can be solved in $\mathcal{O}(kn \log n)$ time. By similar arguments, the case when p is a prefix of $(xf(x))^\infty$ is analysed faster, in $\mathcal{O}(n \log n)$ time.

Altogether, the analysis of the above cases takes $\mathcal{O}(kn \log n)$, where the most time-consuming step is the case when p contains at least one factor $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, and the word v, to which x is mapped, is different from $f(v)$. In the following, we show how we can reduce the time needed to analyse this case to $\mathcal{O}(n \log n)$ (that is, shave off the k factor), and, consequently, obtain that Problem 1 can be solved in $\mathcal{O}(n \log n)$ time.

▶ **Lemma 9.** *Testing whether a word w, $|w| = n$, has as factor an instance of a pattern p, $|p| = k$, that contains $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, can be done in $\mathcal{O}(n \log n)$ time.*

Proof. The approach in Lemma 8 can be optimised as follows. Instead of iterating through all positions where a factor of the form $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ or $vf(v)f(v)$ occurs, and then verifying in $\mathcal{O}(k)$ time if a given occurrence can be extended to form an occurrence of the whole pattern, we look at entire groups of such factors at once, and adapt the Knuth-Morris-Pratt algorithm (see [9]) to verify all of them at once. To make this verification efficient, we will run the pattern matching algorithm not on the original word, but on its suitably constructed compressed representation.

Using the same strategy as in the initial analysis of this case, we identify all the factors $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ with $v \neq f(v)$ in w. We saw above that there are $\mathcal{O}(n \log n)$ occurrences of such factors. Note that a factor $vvf(v)$ (or a factor $vf(v)f(v)$) can be also identified as a factor $f(v')f(v')v'$ (respectively, as a factor $f(v')v'v'$), but only when $f(f(a)) = a$ for all the letters a occurring in v.

Then we group together the occurrences of $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ factors having the same length. This can be done in $\mathcal{O}(n \log n)$ time by describing each such factor as a pair containing the position where it starts and the length of $|v|$, and then putting together in a set (ordered with respect to the starting position) all the factors with v of a certain length. As $|v| \leq n$ we will have $\mathcal{O}(n)$ such sets. Furthermore, we additionally partition each set into smaller sets (again, ordered with respect to the starting position) according to the remainder of the starting position modulo the length of v, which takes linear time in the size of the set. Finally, each such set is split into even smaller groups: we put together all the consecutive elements in the (ordered) set, that have the form $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ for a certain (the same for all factors) word v. As we have already mentioned, a factor $vvf(v)$ (or a factor $vf(v)f(v)$) can be also identified as a factor $f(v')f(v')v'$ (respectively, as a factor $f(v')v'v'$), whenever $v' = f(v) \neq v$; however, the group corresponding to v is exactly the same one as the one constructed for v', so we do not need to consider them separately. Note that, in the end, there may be more than one group corresponding to the same v; however, one occurrence of $vvf(v)$ (as well as one occurrence of $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$) belongs to exactly one group.

The idea of this splitting into groups is that an instance of the pattern p, with x mapped to v, should have all the occurrences of factors $xxf(x)$, $f(x)f(x)x$, $xf(x)f(x)$, or $f(x)xx$ mapped to elements of the same group, that corresponds to v. Hence, in what follows we fix a single group (and, consequently, a word v) and show how to detect a corresponding instance of the pattern, with x mapped to v, in linear time in the size of the group. In particular, if f acts as an involution on the letters of v, we should also try to detect a corresponding instance of p, with x mapped to $f(v)$; but this is done analogously, and takes the same time.

We additionally partition the fixed group into subgroups. Each subgroup is a maximal set of consecutive elements of the group such that between any two of them we either have a power of either v or $f(v)$ (note that two consecutive elements, say $vvf(v)$ and $vf(v)f(v)$, might overlap, and in such case there is nothing between them). The partitioning requires just a single left-to-right scan of the elements of the group, with a constant number of LCP queries when moving from one element to the other. Now a single subgroup corresponds to a factor of w of the form $\{v, f(v)\}^+$, and we have an ordered list of all occurrences of $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ in this factor. Furthermore, the factor starts and ends with factors of this form, so we call it v-delimited. We can represent an v-delimited word in a unique compressed form, called v-representation, as follows.

Sweep through all occurrences of $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$. For each of them append $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, respectively, to the current v-representation. Then, for each such two adjacent occurrences additionally insert one of the following between the corresponding elements of the current representation: -2 if the occurrences overlap by $2|v|$; -1 if the occurrences overlap by $|v|$; 0 if the occurrences are one after another; (x, h) if the fragment between the occurrences is v^h with $h > 0$; $(f(x), h)$ if the fragment between the occurrences is $f(v)^h$ with $h > 0$.

▶ Example 1. If $v = a$, $f(v) = b$, and the word is $aabbaaaaabbbba$, its v-representation is $[xxf(x), -1, f(x)f(x)x, (x, 2), xxf(x), (f(x), 1), f(x)f(x)x]$.

Finally, treat each element of the representation as a single character (over a new alphabet, whose size is $\mathcal{O}(n^c)$ for some constant c, so the characters can be compared in constant time), and the whole representation as a single word. It is clear that above definition guarantees that the v-representation is unique. Furthermore, given a subgroup we can construct its v-representation in linear time (in the size of the subgroup). Similarly, we can locate the first and the last occurrence of $xxf(x)$, $f(x)f(x)x$, $f(x)xx$ or $xf(x)f(x)$ in the pattern and

construct in linear time (in the length of the pattern) the representation of its x-delimited factor starting at the first and ending at the last of such occurrences. Finally, observe that an occurrence of the whole pattern with x mapped to v corresponds to an occurrence of such maximal v-delimited factor, and furthermore given the latter we can verify in constant time if it corresponds to the former using a few LCP queries (i.e., the maximal v-delimited factor can be padded with powers of v or $f(v)$, to get an actual instance of the pattern p).

Hence, we can focus on generating all instances of the maximal x-delimited factor of the pattern, and this is what we designed the x-representation for. We simply apply the usual Knuth-Morris-Pratt algorithm to locate all occurrences of the x-representation of the maximal x-delimited factor of the pattern in the v-representation of the factor corresponding to the subgroup. This takes linear time in the size of the subgroup, excluding the preprocessing, and the number of occurrences is linear, too. Finally, note that the preprocessing of the pattern is done for all subgroups just once. ◀

This approach clearly identifies an instance, if any, of a pattern p that contains one of the factors $xxf(x)$, $f(x)xx$, $xf(x)f(x)$, or $f(x)f(x)x$, in time $\mathcal{O}(n \log n)$. In conclusion, by this last argument and the previous remarks, we obtained an $\mathcal{O}(n \log n)$ solution for Problem 1.

An $\mathcal{O}(n \log n)$ solution for Problem 2. In this case we want to test whether w contains the image of any pattern p from $\{x, f(x)\}^k$, for a given k. However, we will determine the maximum ℓ such that w contains an instance of some pattern $p \in \{x, f(x)\}^\ell$. Let us note that for $\ell \leq 3$ we have a linear time solution for testing whether w contains an f-repetition of exponent ℓ, as described in Section 4. Hence we focus on the remaining case $\ell \geq 4$.

The key remark in our approach is that if w contains an instance of a pattern $p \in \{x, f(x)\}^{k_1}$ then it also contains an instance of a pattern $p' \in \{x, f(x)\}^{k_2}$ where x is replaced by a primitive word v and $k_2 \geq k_1$. Hence, we identify first the maximal factors of w that have the form $\{t, f(t)\}^*$, for some primitive word t (thus, with $f(t)$ primitive, as well), and contain either tt or $f(t)f(t)$, and, then, the ones that do not contain such squares.

This is done similarly to the subgroup splitting of the previous section. We first locate the occurrences of such factors, in $\mathcal{O}(n \log n)$ time, and then refine this set to obtain the subgroups containing the occurrences of factors defined by the same t which have between them only elements from $\{t, f(t)\}^*$. Like before, this takes $\mathcal{O}(n \log n)$ time.

So far, we obtained the maximal factors of w that have the form $\{t, f(t)\}^*$ and start either with tt or $f(t)f(t)$. For such a factor y that starts with tt we compute in constant time the maximal factor z that ends just before y and has the form $(t\{f(t)t\}^* \cup \{f(t)t\}^*)f(t)$, following Remark 1. Then zy is a maximal factor that contains tt. Similarly, for a factor $y \in f(t)f(t)\{t, f(t)\}^*$ we compute the maximal factor z that ends just before y and has the form $(f(t)\{tf(t)\}^* \cup \{tf(t)\}^*)t$. Then zy is a maximal factor that contains $f(t)f(t)$. Consequently, we computed the maximal factors of w that have the form $\{t, f(t)\}^*$ and contain either tt or $f(t)f(t)$. Clearly, the time complexity of this procedure is $\mathcal{O}(n \log n)$.

Now, if at least one of the maximal factors of w of the form $\{t, f(t)\}^*$ and containing tt or $f(t)f(t)$ is an f-power of t having the exponent at least k, then we can answer Problem 2 positively. Otherwise, we need to check whether w contains a repetition of the form $\{tf(t)\}^\ell$, $\{tf(t)\}^\ell t$, $\{f(t)t\}^\ell$, or $\{f(t)t\}^\ell f(t)$, for a large enough exponent $\ell \geq 2$. But, when looking at maximal f-repetitions of this type, we get that their prefix of length $2|t|$ (that is, $tf(t)$ or $f(t)t$) should be primitive; otherwise, longer repetitions were detected in the previous step. Hence, such repetitions start with a primitively rooted square. Now, analysing all the primitively rooted squares occurring in w, and mapping them to $t(f)tf(t)$ or $f(t)tf(t)t$, depending on

the form pattern we look for (from the ones above), we can detect all these repetitions in $\mathcal{O}(n \log n)$ time, too. In conclusion, we solve completely Problem 2 in $\mathcal{O}(n \log n)$ time.

Using this strategy, we also identify, in $\mathcal{O}(n \log n)$ time, all the factors t and v of w such that t is primitive and $v \in \{t, f(t)\}^*$, and v cannot be extended, in any direction, by neither t nor $f(t)$. It is not hard to see that one of these f-repetitions is the one that has the maximum exponent among all the f-repetitions contained in w. Thus, we can also find the maximum power of an f-repetition contained in w, within $\mathcal{O}(n \log n)$ time.

6 Conclusions

In this paper we showed the following theorems.

▶ **Theorem 10.** *Given a word $w \in V^*$, with $|w| = n$, a literal anti-/morphism $f : V^* \to V^*$, and an f-pattern p of length k, we can decide whether w contains an instance of p in $\mathcal{O}(nk^2)$ time; for a fixed pattern p, the problem can be solved in linear time. If f is bijective, then the problem can be solved in $\mathcal{O}(n \log n)$ time.*

▶ **Theorem 11.** *Given a word $w \in V^*$, with $|w| = n$, a literal anti-/morphism $f : V^* \to V^*$, and a positive integer k, we can decide whether w contains a factor of the form $\{t, f(t)\}^k$, for some word t, in $\mathcal{O}(nk^2)$ time; for a constant k, the problem can be solved in linear time. If f is bijective, then we can compute the maximum k such that w contains a factor of the form $\{t, f(t)\}^k$, for some word t, in $\mathcal{O}(n \log n)$ time.*

We conjecture that results in the line of the second parts of our theorems (and similar proofs) hold also for general literal morphisms. In this case, however, one has to overcome the difficulty that when f is not bijective, then $f(t)$ is no longer primitive for all primitive t; accordingly, the technicalities are expected to be far more involved. Consequently, the time bounds are expected to be larger (although still close to linear time).

The main question left open is whether the results reported here can be improved to find (if there exist) algorithmic solutions for the approached problems running in $\mathcal{O}(n)$ time.

References

1. E. Chiniforooshan, L. Kari, and Z. Xu. Pseudopower avoidance. *Fundamenta Informaticae*, 114(1):55–72, 2012.
2. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
3. Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Walen. Efficient algorithms for two extensions of lpf table: The power of suffix arrays. In *Proc. SOFSEM*, volume 5901 of *LNCS*, pages 296–307, 2010.
4. E. Czeizler, L. Kari, and S. Seki. On a special class of primitive words. *Theoretical Computer Science*, 411:617–630, 2010.
5. H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proc. STOC*, pages 246–251. ACM, 1983.
6. P. Gawrychowski, F. Manea, R. Mercas, D. Nowotka, and C. Tiseanu. Finding Pseudo-repetitions. In *Proc. STACS*, volume 20 of *LIPIcs vol. 20*, pages 257–268, 2013.
7. P. Gawrychowski, F. Manea, and D. Nowotka. Discovering hidden repetitions in words. In *Proc. CiE*, volume 7921 of *LNCS*, pages 210–219. Springer, 2013.
8. J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, 2006.

9 D. E. Knuth, J. H. Morris Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

10 R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Proc. FOCS*, pages 596–604. IEEE Computer Society, 1999.

11 M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989.

12 F. Manea, R. Mercas, and D. Nowotka. Fine and Wilf's theorem and pseudo-repetitions. In *Proc. MFCS*, volume 7464 of *LNCS*, pages 668–680. Springer, 2012.

13 F. Manea, M. Müller, and D. Nowotka. The avoidability of cubes under permutations. In *Proc. DLT*, volume 7410 of *LNCS*, pages 416–427. Springer, 2012.

14 Zhi Xu. A minimal periods algorithm with applications. In *Proc. CPM*, volume 6129 of *LNCS*, pages 51–62. Springer, 2010.