Detecting Unary Patterns

Dmitry Kosolobov\textsuperscript{1,2}, Florin Manea\textsuperscript{1}, and Dirk Nowotka\textsuperscript{1}

\textbf{1} Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany, \{flm,dn\}@informatik.uni-kiel.de

\textbf{2} Ural Federal University, Institute of Mathematics and Computer Science, Ekaterinburg, Russia, dkosolobov@mail.ru

---

\textbf{Abstract}

Given a pattern \( p = s_1 x_1 s_2 x_2 \cdots s_{r-1} x_{r-1} s_r \) such that \( x_1, x_2, \ldots, x_{r-1} \in \{ x, \bar{x} \} \), where \( x \) is a variable and \( \bar{x} \) its reversal, and \( s_1, s_2, \ldots, s_r \) are strings that contain no variables, we describe an algorithm that constructs in \( O(n) \) time a compact representation of all \( P \) instances of \( p \) in an input string of length \( n \), so that one can report those instances in \( O(P) \) time.

\textbf{1998 ACM Subject Classification} F.2.2 Pattern Matching

\textbf{Keywords and phrases} pattern matching, patterns with variables, repetitions, pseudo-repetitions

---

\textbf{1 Introduction}

A \textit{pattern} is a string consisting of \textit{variables} (e.g., \( x, y, z \)) and \textit{terminal letters} (e.g., \( a, b, c \)). The terminal letters are treated as constants, while the variables are letters to be uniformly replaced by strings over the set of terminals (i.e., all occurrences of the same variable are replaced by the same string); by such a replacement, a pattern is mapped to a terminal string. Patterns appeared in various areas of computer science: stringology and pattern matching \[14, 24\], combinatorics on words \[13, 20, 22\], language theory \[2\], learning theory \[2, 7, 14, 23, 25\], or extended regular expressions \[4, 10\], used in programming languages like Perl, Java, Python.

In such applications, patterns are used to express string searching questions such as testing whether a string contains regularities. Here, we consider simple unary patterns \( p = s_1 x_1 \cdots s_{r-1} x_{r-1} s_r \) such that, for all \( z, x_z \in \{ x, \bar{x} \} \), where \( x \) is a variable and \( \bar{x} \) its reversal, and \( s_z \) is a string over some set \( \Sigma \) of terminals. Such patterns naturally generalize repetitions (patterns from \( \{ x \}^* \)) and pseudo-repetitions (patterns from \( \{ x, \bar{x} \}^* \)). An instance of \( p \) in a string \( t \) is a substring \( \{ i,j \} = s_1 w_1 \cdots s_{r-1} w_{r-1} s_r \), with \( w_z = w \) if \( x_z = x \) and \( w_z = \bar{w} \) if \( x_z = \bar{x} \), for some non-empty \( w \in \Sigma^* \) called \textit{substitution} of \( x \). Efficiently finding instances of such patterns in texts was a central problem in \[12, 8\]; thus, we consider the following problem.

\textbf{Problem 1.} Given a string \( t \in \Sigma^* \) of length \( n \) and a pattern \( p = s_1 x_1 \cdots s_{r-1} x_{r-1} s_r \) such that, for \( 1 \leq z \leq r-1, x_z \in \{ x, \bar{x} \} \) where \( x \notin \Sigma \) is a variable and \( \bar{x} \) its reversal, and \( s_z \in \Sigma^* \) for \( 1 \leq z \leq r \), report all \( P \) instances of \( p \) in \( t \) (in a form allowing their retrieval in \( O(P) \) time).

We assume that \( t \) and all strings \( s_z \), for \( z = 1, \ldots, r \), are over an integer alphabet \( \Sigma = \{ 0, 1, \ldots, n^{O(1)} \} \). Under these assumptions, we propose an algorithm that reports in \( O(n) \) time all instances of \( p \) in \( t \) in a compactly encoded form, which indeed allows us to retrieve them in \( O(P) \) time. Our approach is based on a series of deep combinatorics on words observations, e.g., regarding the repetitive structure of the text and on the usage of efficient string-processing data structures, combining and extending in novel and nontrivial ways the ideas from \[8, 12, 13\]. For simplicity of exposure, we omit some of the technical details of the solution of Problem 1 in the main part of the paper; they are given in Appendix.
2 Detecting Unary Patterns

If the pattern contains only a constant number of variables (e.g., generalized squares or cubes with terminals between the variables), our algorithm is asymptotically as efficient as the algorithms detecting fixed exponent (pseudo-)repetitions. For arbitrary patterns, our solution generalizes and improves the results of [12], where an $O(n^2 \log n)$-time solution to the problem of matching unary patterns with reversals, but without terminals, was given. Also, we improve the results of [8] in several directions: we find all instances of a unary pattern (in [8] only some instances were found), our algorithm is faster by a log $n$ factor (hereafter, log denotes the logarithm with base 2), and it also handles reversed variables. As a direct application, we show how to efficiently find instances of certain patterns with only one repeated variable (but with other variables occurring each only once).

Problem 2. Decide whether a text $t \in \Sigma^*$ of length $n$ contains an instance of a pattern $p = \left( \prod_{j=1,k}^{i,j}(y_{j}x_{j} \cdots x_{r_{j}-1,j}x_{s_{j}r_{j}}) \right) y_{k+1}$, with $r_{j} \geq 3$ for all $1 \leq j \leq k$, $r = \sum_{j=1}^{k} r_{j}$, such that $x$ and $y_{2}, \ldots, y_{k}$ are variables, $y_{1}$ and $y_{k+1}$ are either variables or the empty strings (i.e., they might be missing), and the strings $s_{2,j} \in \Sigma^*$.

We solve this problem in $O(n \log n)$ time by extending our solution of Problem 1. Note that in our patterns the maximal blocks containing only $x$ variables and terminals must have at least two occurrences of $x$. Our algorithm runs, for such restricted patterns and for $r = o(\frac{n}{\log n})$ (which also covers the important case of fixed patterns), faster than the $O(n^2)$-time algorithm matching patterns with one repeated variable from [8], which was the main step in designing an algorithm matching patterns with $k$ repeated variables.

2 Solving Problem 1

Preliminaries. Let $w$ be a string of length $n$. Denote $|w| = n$. The empty string is denoted by $\epsilon$. We write $w[i]$ for the $i$th letter of $w$ and $w[i..j]$ for $w[i]w[i+1] \cdots w[j]$. A string $u$ is a substring of $w$ if $u = w[i..j]$ for some $i$ and $j$. The pair $(i,j)$ is not necessarily unique; we say that $i$ specifies an occurrence of $u$ in $w$. A substring $w[1..j]$ [resp., $w[i..n]$] is a prefix [resp. suffix] of $w$. The reversal of $w$ is the string $\overline{w} = w[n] \cdots w[2]w[1]$; $w$ is a palindrome if $w = \overline{w}$. For any $i,j$, the set \{ $k \in \mathbb{Z}$: $i \leq k \leq j$ \} (possibly empty) is denoted by $[i,j]$; denote $(i..j) = [i,j] \setminus \{i\}$, $[i,j] = [i,j] \setminus \{j\}$, $(i..j) = [i,j] \cap (i..j)$. Our notation for arrays is similar to that for strings: for instance, $a[i..j]$ denotes an array indexed by the numbers $[i,j]$.

Let $t$ be an input string of length $n$ and $p = s_{1}s_{2}x_{2} \cdots s_{r-1,x_{r-1}}s_{r}$ be a pattern such that, for $z \in \{1..r\}$, $x_{z} \in \{x, \overline{x}\}$ and $s_{1}, s_{2}, \ldots, s_{r}$ are strings that contain no $x$ or $\overline{x}$. An instance of $p$ in $t$ is a substring $[i..j] = s_{1}w_{1}s_{2}w_{2} \cdots s_{r-1}w_{r-1}s_{r}$ such that, for $z \in \{1..r\}$, $w_{z} = w_{i}$ if $x_{z} = x$, and $w_{z} = \overline{w}_{z}$ if $x_{z} = \overline{x}$, where $w$ is a string called a substitution of $x$; $\overline{w}$ is called a substitution of $\overline{x}$. We use the word RAM model with $\Theta(\log n)$-bit machine words.

An integer $d > 0$ is a period of a string $w$ if $w[i] = w[i+d]$ for all $i \in \{1,..,|w|-d\}$; $w$ is periodic if it has a period $\leq |w|$. For a string $w$, denote by $pre_{d}(w)$ and $suf_{d}(w)$, resp., the longest prefix and suffix of $w$ with period $d$. A run of a string $w$ is a periodic substring $w[i..j]$ such that both substrings $w[i..j-1]$ and $w[i..j+1]$, if defined, have strictly greater minimal periods than $w[i..j]$. A string $w$ is primitive if $w \neq w^{k}$ for any string $v$ and integer $k > 1$.

Lemma 1 (see [8]). Any primitive string $v$ has exactly two occurrences in the string $vv$.

Lemma 2. Let $R$ be the set of all runs of $t$, whose period is at least three times smaller than the length of the run. Then $\sum_{s \in R} |s| \in O(n \log n)$.

Proof. This follows immediately, as $\sum_{s \in R} |s|$ is upper bounded by the three times the number of primitively rooted squares occurring in $t$. At each position of $t$ occur at most $2[\log n]$ primitively rooted squares (see, e.g., [8]), so the result follows. 

▶
We find all runs in $t$ in $O(n)$ time using the algorithm of [33] and, using the radix sort, construct lists $R_d$, for $d = 1, 2, \ldots, n$, such that $R_d$ contains the starting positions of all runs with the minimal period $d$ in increasing order. For each $R_d$, we create two sublists $R'_d$ and $R''_d$ containing only runs with the lengths $\geq \log n$ and $\geq \log \log n$, respectively. The following lemma provides us the fast access to the lists $R_d, R'_d, R''_d$ from periodic substrings of $t$.

**Lemma 3** (see [15] Lemma 6.6). Assuming $O(n)$ time preprocessing, we can decide for any substring of $t$ whether it is periodic and, if so, compute its minimal period $d$ and find in $R_d$ or $R'_d$ or $R''_d$ the run containing this substring (if any) in $O(1)$ time.

For $i, j \in [1..n]$, denote by $\text{lcp}(i, j)$ and $\overline{\text{lcp}}(i, j)$ the lengths of the longest common prefixes of the strings $t[i..n]$, $t[j..n]$ and $\overline{t[1..i]}$, $\overline{t[1..j]}$, respectively. Our algorithm constructs in $O(n)$ time the so called $\text{lcp-data structure}$ that allows to calculate $\text{lcp}(i, j)$ and $\overline{\text{lcp}}(i, j)$ for any $i, j \in [1..n]$ in constant time (e.g., see [3]). W.l.o.g., assume that $\log n$ is an integer.

**General strategy.** For each $z \in [1..r]$, using a pattern matching algorithm, we fill in $O(n)$ time a bit array $D_z[1..n]$ such that, for $i \in [1..n]$, $D_z[i] = 1$ iff $s_2$ occurs at position $i$. Assume that neither $p = s_1xs_2$ nor $p = s_1zs_2$; these cases can be easily processed using $D_1$ and $D_2$.

Denote $\alpha = \frac{4}{3}$. For each $k \in [0..\log\log n]$, our algorithm finds all instances of $p$ that are obtained by the substitution of $x$ by $y$ strings having length from the segment $\left(\frac{\alpha}{2}\alpha^k..2\alpha^k\right]$. It is easy to verify that the segments $\left(\frac{\alpha}{2}\alpha^k..2\alpha^k\right]$ do not intersect and their union covers the segment $[2..n]$. Hence, we obtain all nontrivial instances of $p$ in this manner. The remaining trivial instances can be found in $O(rn)$ time using the arrays $\{D_z\}_{z=1}^n$ in an obvious way.

Fix $k \in [0..\log\log n]$. Suppose that, for $i, j \in [1..n]$, $t[i..j] = s_1w_1s_2w_2\cdots s_{r_1}w_{r_1}s_r$ is an instance of $p$ and $\frac{3}{2}\alpha^k < |w_1| = \cdots = |w_{r_1}-1| \leq 2\alpha^k$; then $w_1$ contains a substring $v$ of length $[\alpha^k]$ starting either at position $h[v] + 1$ or at position $h[v] + \lfloor \frac{|v|}{\alpha^k} \rfloor$ for some integer $h \geq 0$. Based on this observation, we consider all substrings $v$ of length $[\alpha^k]$ starting at positions $h[\alpha^k] + 1$ and $h[\alpha^k] + \lfloor \frac{|v|}{\alpha^k} \rfloor$ for $h \geq 0$ and find all corresponding instances of $p$ that contain $v$ in the substitution of $x_1$ in $O(r + \frac{r|v|}{\log n})$ time plus $O(\frac{\log n}{\log \log n})$ time if $\frac{\log n}{\log \log n} \leq |v| \leq \log n$. Since there are $O\left(n \frac{1}{\alpha^k}\right)$ such substrings $v$ and at most $O(n/\log \log n)$ of them (for all $k = 0, 1, \ldots$ in total) satisfy the condition $\frac{\log n}{\log \log n} \leq |v| \leq \log n$, the overall time is $O(\sum_{k=0}^{\log\log n} n \frac{r}{\alpha^k} + \frac{n}{\log \log n}) + n = O(\sum_{k=0}^{\log\log n} n \frac{r}{\alpha^k} + \frac{rn}{\log \log n}) = O(rn)$.

More precisely, the lemmas we show in the following sections prove the next theorem.

**Theorem 4.** Problem [2] can be solved in $O(rn)$ time.

### 2.1 Non-periodic substring $v$

Let $v$ be a substring of $[\alpha^k]$ starting at position $q_1 = h[v] + 1$ for some integer $h \geq 0$ (the case of position $h[v] + \lfloor \frac{|v|}{\alpha^k} \rfloor$ is similar). Using Lemma 3, we check whether $v$ is periodic. Suppose that $v$ is not periodic; the case of periodic $v$ is considered in Section 2.2.

Our aim is to find all instances $t[i..j] = s_1w_1s_2w_2\cdots s_{r_1}w_{r_1}s_r$ of $p$ such that $i + |s_1| \leq q_1 < q_1 + |v| \leq i + |s_1|w_1| = \cdots = |w_{r_1}-1| \leq 2|v|$. Let $t[i..j]$ be such substring. It follows from the inequality $\frac{3}{2}|v| < |w_1| = \cdots = |w_{r_1}-1| \leq 2|v|$ that, if $w_1 = w_2$ (resp., $w_1 = \overline{w_2}$), then the string $v$ (resp., $\overline{v}$) has an occurrence starting in $[q_1 + |s_2|..q_1 + |s_2|v|]$. Since $v$ is not periodic, the length of overlapping of any two distinct occurrences of $v$ is less than $\frac{3}{2}|v|$. Hence, there are at most four occurrences of $v$ (resp., $\overline{v}$) starting in $[q_1 + |s_2|..q_1 + |s_2|v|]$. To find these occurrences, our algorithm applies the following lemma putting $\lambda = |s_2|$.
Detecting Unary Patterns

Lemma 5. Given an integer $\lambda \geq 0$, assuming $O(n)$ time preprocessing, one can find for any given non-periodic substring $v = [q..q']$ all occurrences of $v$ and $\bar{v}$ starting in the segment $t[q' + \lambda..q' + \lambda + 2|v|]$ in $O(\frac{|v|}{\log n})$ time if $|v| > \log n$, $O(\frac{\log n}{\log \log n})$ time if $\frac{\log n}{\log \log n} \leq |v| \leq \log n$, and $O(1)$ time otherwise.

Proof. For $i \in [1..n]$, denote $t_i = t[i..i + \log n]$ and $t_i' = t[i..i + \log \log n]$ (if defined). Denote by $S$ [resp., $S'$] the set of all distinct strings $t_i$ [resp., $t_i'$]. Using the suffix array of $t$, the lcp data structure, and the radix sort, we construct in $O(n)$ time the sets of arrays $\{A_s\}_{s \in S}$ and $\{A'_s\}_{s' \in S'}$ such that, for any $s \in S$ [resp., $s' \in S'$], $A_s$ [resp., $A'_s$] contains the starting positions of all occurrences of $s$ [resp., $s'$] in $t$ in ascending order. Further, using the suffix array of the string $\bar{t}$, the lcp data structure, and the radix sort, we build in $O(n)$ time arrays of pointers $B[1..n], B'[1..n], \bar{B}[1..n], \bar{B}'[1..n]$ such that, for $i \in [1..n]$, $B[i]$ [resp., $B'[i], \bar{B}[i], \bar{B}'[i]$] points to the element of $A_i$ [resp., $A'_i$, $A_i'$, $A_i''$] storing the leftmost position $j$ such that $j \geq i + \lambda$ and $t_i = t_j$ [resp., $t_i' = t_j'$, $\bar{t}_i = \bar{t}_j$, $\bar{t}_i' = \bar{t}_j'$]; $B[i]$ [resp., $B'[i], \bar{B}[i], \bar{B}'[i]$] is undefined if there is no such $j$.

The case $|v| > \log n$. To find all required occurrence of $v$, we first find all occurrences of $t_q$ starting in the segment $[q + \lambda..q' + \lambda + 2|v|]$. The sequence of all such occurrences forms a contiguous subarray in $A_q$ and $\bar{B}[q]$ points to the beginning of this subarray. Suppose that the distance between any two positions in this subarray is greater than $\frac{|v|}{2}$. Then there are at most $O(\frac{|v|}{\log n}) = O(\frac{|v|}{\log \log n})$ such occurrences of $t_q$. Some of these occurrences are candidates for an occurrence of $v$. To check whether $v$ occurs at a given position $i$, we use the lcp data structure. The case of the string $\bar{v}$ is analogous but involves $\bar{t}_q$ and $\bar{B}$ instead of $t_q$ and $B$. Hence, we find all required occurrence of $v$ and $\bar{v}$ in $O(\frac{|v|}{\log \log n})$ time.

Suppose that, during the scanning of $A_q$ [resp., $A_q'$], we found two occurrences of $t_q$ [resp., $\bar{t}_q$] whose starting positions differ by at most $\frac{|v|}{2}$. Then $t_q$ is periodic. Using Lemma 3, we compute the minimal period $d$ of $t_q$ and find in the list $R_{d}'$ a run $t[j..j']$ containing $t_q$ in $O(1)$ time. Since $v$ is not periodic, we have $|v| = \text{pre}_q(v)$ and $|\text{pre}_q(v)| < |v|$. Since $R_{d}'$ contains only runs of length $\geq \log n$ and any two runs with period $d$ cannot overlap on $d$ letters, there are at most $O(\frac{|v|}{\log n})$ runs in $R_{d}'$ that overlap with the segment $[q + \lambda..q' + \lambda + 2|v|]$, we find them all in $O(\frac{|v|}{\log n})$ time. Some of the found runs are candidates for an occurrence of $v$ [resp., $\bar{v}$]: if $t[j'..j''']$ is such run, then there might be an occurrence of $v$ starting at position $j'' - j' + q$ or an occurrence of $\bar{v}$ ending at position $i'' + j' - q$. So, using the lcp data structure, we find all required occurrence of $v$ [resp., $\bar{v}$] in $O(\frac{|v|}{\log \log n})$ time.

The case $\frac{\log n}{\log \log n} \leq |v| \leq \log n$. This case is similar to the case $|v| > \log n$ but now we use the string $t_i'$ instead of $t_q$, the arrays $A_i', B_i', \bar{B}_i$, and the list $R_{d}'$ instead of $R_d$. The processing takes $O(\frac{|v|}{\log \log n}) = O(\frac{\log n}{\log \log n})$ time.

The case $|v| < \frac{\log n}{\log \log n}$. Using the radix sort, our algorithm reduces the alphabet of $t$ to $[0..n]$ in $O(n)$ time. Denote by $S$ the special letter $n$. For $h \in [0..\frac{n}{\log n}]$, define $e_h = t[h \log n + 1..h \log n + 2 \log n]$ and $f_h = t[h \log n + \lambda..h \log n + \lambda + 5 \log n]$ assuming $S = t[n + 1] = t[n + 2] = \ldots$, so that $e_h$ and $f_h$ are well defined. Note that the string $v$ is a substring of $e_h$ for $h = \frac{|v| - 1}{\log n}$ and, if there is an occurrence of $v$ [resp., $\bar{v}$] starting in the segment $[q' + \lambda..q' + \lambda + 2|v|]$, then this occurrence is a substring of $f_h$.

For each $h \in [0..\frac{n}{\log n}]$, our algorithm constructs a string $g_h = e_h S f_h$ and reduces the alphabet of $g_h$ to $[1..|g_h|]$ as follows. Let $E[0..n]$ be an array of integers filled with zeros. While processing $g_h$, we maintain a counter $c$; initially, $c = 0$. For $i = 1, 2, \ldots, |g_h|$, we check whether $E[g_h[i]] = 0$ and, if so, assign $c \leftarrow c + 1$ and $E[g_h[i]] \leftarrow c$. Regardless of the result
Therefore, the following segments must be non-empty (see Fig. 1): 

\[ \text{Proof.} \]

that \( q \mid Z \)

Furthermore, if such instance \( S = q_1 \cdots q_r \) is found in \( Z \) of \( g \), we perform \( t[q .. q + |v|] \) in \( O(1) \) time using a precomputed table of size \( O(2^{\log_2 n}) = O(n) \).

A Lyndon root \( [\text{resp., reversed Lyndon root}] \) of a run \( t[i .. j] \) with period \( d \) is a lexicographically smallest substring \( t[i+1 .. i+d] \) \( [\text{resp., } t[i+1 .. i+d]] \) such that \( i \in [i-1 .. i-j] \).

\begin{lemma}[see Lemma 1] \end{lemma}

The leftmost \( [\text{reversed}] \) Lyndon root of any run in \( t \) can be found in \( O(1) \) time assuming \( O(n) \) time preprocessing.

Let \( q_2 \in \{ q_1 + [v]s_2, q_1 + [v]s_2 \} \) be the starting position of an occurrence of \( v \) found by Lemma 5. If \( x_1 = x_2 \), then \( \beta = q_2 - q_1 - |s| \) is the length of substitution \( x \) that could produce the occurrence of \( v \) at position \( q_2 \). Once the length \( \beta \) is computed, all corresponding instances of \( p \) can be found by the following lemma (see the case \( x_1 \neq x_2 \) in Appendix).

\begin{lemma} \end{lemma}

Given a substring \( t[h_1 .. h_2] = v \) and an integer \( \beta \geq |v| \), we can compute a bit array \( oc[h_2 - \beta - |s_1| + 1 .. h_1 - |s_1|] \) such that, for any \( i \), we have \( oc[i] = 1 \) iff the string \( t[i+|s_1| .. s_i + (r-1)\beta - 1] \) is an instance of \( p \) such that \( i + |s_1| \leq h_1 < h_2 < i + |s_1| + \beta \). This computation takes \( O(r + \log n) \) time assuming \( O(n) \) time.

Proof. For \( z \in [1 .. r] \), denote \( q_z = h_1 + |s_1| \cdots |s_1| + (z-1)\beta \). Denote by \( Z \) \( [\text{resp., } \tilde{Z}] \) the set of all \( z \) \( [\text{resp., } \tilde{z}] \) such that \( x_z = x \) \( [\text{resp., } x_z = \tilde{x}] \). If there is an instance \( t[i .. j] = s_1w_1s_2w_2 \cdots s_{r-1}w_{r-1}s_r \) of \( p \) such that \( |w_1| = \cdots = |w_{r-1}| = \beta \) and \( i + |s_1| \leq h_1 < h_2 < i + |s_1| \), then, for any \( z, z' \in Z \) \( [\text{resp., } z, z' \in \tilde{Z}] \), \( t[q_z .. q_{z'} + |v| - 1] = t[q_z .. q_{z'} + |v| - 1] \). We check this necessary condition in \( O(r) \) time using the lcp data structure. Suppose that this checking succeeds. Note that there might exist many corresponding instances of \( p \) (see Fig. 1).

**Figure 1** Two instances of the pattern \( p = bxabxx \)

In an obvious way we calculate the numbers \( b_r = \min \{ \lcp(q_{z-1}, q_{z-1}) : (z, z') \in (Z \times Z) \cup (\tilde{Z} \times \tilde{Z}) \} \) and \( b_r = \min \{ \lcp(q_z + |v|, q_{z'} + |v|) : (z, z') \in (Z \times Z) \cup (\tilde{Z} \times \tilde{Z}) \} \) in \( O(r) \) time. By the definition of \( b_r \) and \( b_r \), if \( t[i .. j] = s_1w_1s_2w_2 \cdots s_{r-1}w_{r-1}s_r \) is an instance of \( p \) such that \( |w_1| = \cdots = |w_{r-1}| = \beta \) and \( i + |s_1| \leq h_1 < h_2 < i + |s_1| \), then we necessarily have \( q_z - \delta \geq q_b - b_r \) and \( q_z - \delta - \beta \leq q_z + |v| + b_r \) for all \( z \in [1 .. r] \), where \( \delta = q_1 - (i + |s_1|) \).

Therefore, the following segments must be non-empty (see Fig. 1):

\[ S_z = \{ q_z - |s_z| - b_r .. q_{z-1} + |v| + b_r \} \cap \{ q_{z-1} + |v| .. q_z - |s_z| \} \text{ for } z \in [1 .. r], \]

\[ S_1 = \{ q_1 - |s_1| - b_r .. q_{|s_1| - 1} + |v| \cap \{ q_1 + |v| .. |s_1| - \beta \} \cap \{ q_1 - |s_1| - b_r \} \}

\[ S_z = \{ q_{z-1} + |v| .. q_{z-1} + |v| + b_r \cap \{ q_{z-1} + |v| .. q_{z-1} + |v| + b_r + \beta \} \}

Further, if such instance \( i[i .. j] \) exists, then there is a sequence of positions \( \{ i_z \}_{z=1} \) such that \( i_z \in S_z \), \( D_z[i_z] = 1 \) for \( z \in [1 .. r] \) and \( i_{z+1} - i_z = |s_z| + \beta \) for \( z \in [1 .. r] \) (namely,
Detecting Unary Patterns

If \( i_1 = i \). If \( x_1 = \cdots = x_{r-1} \), then the converse is also true: if some sequence \( \{i_1^r\}_{i=1}^r \) satisfies all these conditions, then \( t[i_1..i_r] + |s_r| - 1 = s_1 w s_2 w \cdots \cdot s_r w s_r \), where \( |w| = \beta \) and \( i + |s_r| \leq h_1 < h_2 < i + |s_1| + \beta \). The bit arrays \( \{D_i\}_{i=1}^r \) help us to find all such sequences.

Denote \( D'_z = D_z[q_z-1+|v|+|v|-|s_z|] \) for \( z \in \{1..r\} \) and \( D'_z = D_z[q_z-|s_z| - |s_z| - |q_z|] \). For each \( z \in \{1..r\} \), we clear in the array \( D'_z \) all bits corresponding to the regions that are not covered by the segment \( S_z \) and then perform the bitwise “and” of \( D'_1, \ldots, D'_r \); thus, we obtain a bit array \( D[0..\beta|-|v|] \) (see Fig. 1). If \( x_1 = \cdots = x_{r-1} \), then, for any \( i \in [0..\beta|-|v|] \), we have \( D[i] = 1 \) iff there is a string \( s_1 w s_2 w \cdots \cdot s_r w s_r \) starting at \( i = h_2 - \beta - |s_1| + i + 1 \) such that \( |w| = \beta \) and \( i + |s_1| \leq h_1 < h_2 < i' + |s_1| |w| \). Obviously, one can put \( \text{occ} [h_2 - \beta - |s_1| + 1...h_1 - |s_1|] = D[0..\beta|-|v|] \). Since the length of each of the arrays \( D'_1, \ldots, D'_r \) does not exceed \( \beta \), all these calculations can be performed in \( O(r + \frac{r^2}{\log n}) \) time with the aid of the standard bitwise operations on the \( \Theta (\log n) \)-bit machine words.

The case when \( p \) contains both \( x \) and \( \tilde{x} \) is discussed in Appendix.

### 2.2 Periodic substring \( v \)

Suppose that \( v \) is periodic. Using Lemma \[ \ref{lem:periodic} \] we find in \( O(1) \) time the minimal period \( \rho \) of \( v \) and a run \([i'..j']\) with period \( \rho \) such that \( i' \leq q_1 < q_1 + |v| - 1 \neq j' \). We are searching for instances \( t[i..j] = s_1 w s_2 w \cdots \cdot s_r w s_r \) of \( p \) such that \( \frac{1}{2} |v| < |w| = \cdots = |w_{r-1}| \leq 2 |v| \) and \( i + |s_1| \leq q_1 < q_1 + |v| \leq i + |s_1| |w| \). Let \( t[i..j] \) be such instance. Then, either \( w_1 \) has period \( d \) or one of the strings \( v' = t[q_j..j'+1] \) or \( v'' = t[j'-1..q_1+1] \) is a substring of \( w_1 \).

Suppose that \( w \) contains \( v' \) as a substring (the case of \( v'' \) is similar). It is well known that, since the minimal period of \([q_1..j']\) is \( d \), \( 2d \leq j' - q_1 + 1 \), and \( t[j'+1] \neq t[j'+1-d] \), the string \( v' \) is not periodic. Hence, \( v' \) can be processed in the same way as \( v \) in Section \[ \ref{sec:binary} \].

Suppose that \( w \) has period \( d \). Periodic substitutions (such as \( w \)) can produce a lot of instances of \( p \): e.g., \( a^n \) contains \( \Theta (n^2) \) instances of \( xx \). However, it turns out that such multiple instances have a uniform structure that can be compactly encoded and appear only when all substitutions of \( x \) and \( \tilde{x} \) lie either in one or two runs. Before the discussion of this case, let us first consider the case when more than two runs contain the substitutions.

#### Three and more runs

Let \( t[i..j] \) be an instance of \( p \) with a substitution of \( x_1 \) denoted by \( w \) and such that \( w \) has period \( d \), \( \frac{1}{2} |v| < |w| \leq 2 |v| \), and \( i + |s_1| \leq q_1 < q_1 + |v| \leq i + |s_1| |w| \). Since \( |v| \geq 2d \), we have \( |w| \geq \frac{3}{4} |v| \geq 3d \). Clearly, each substitution of \( x \) or \( \tilde{x} \) in \( t[i..j] \) is contained in some run with period \( d \) (some of these runs can coincide). It turns out that if all substitutions of \( x \) and \( \tilde{x} \) in \( t[i..j] \) are contained in at least three distinct runs with period \( d \), then the length \( |w| \) is equal to one of a constant number of possible lengths each of which can be efficiently found and then processed by Lemma \[ \ref{lem:three} \]. To begin with, let us introduce several technical lemmas (see Appendix for lemmas concerning reversals).

#### Lemma 8

Suppose that \( ws \) is a substring of \( t \) such that \( w \) has period \( d \), \( |w| \geq 3d \), and \( ws \) has no period \( d \). Let \( t[i..j] \) be a run with period \( d \) containing \( w \). Denote by \( h \) the starting position of \( s \). Then, either \( h = j - [\text{pre}_d(s)] + 1 \) or \( h \in (j+1-d..j'+1] \).

**Proof.** Suppose that \( h \leq j + 1 - d \). Then, \( |\text{pre}_d(s)| \geq j - h + 1 \geq d \). Thus, since \( t[j+1] \neq t[j+1-d] \), \( |\text{pre}_d(s)| \) must be equal to \( j - h + 1 \). Hence \( h = j - |\text{pre}_d(s)| + 1 \).

#### Lemma 9

Suppose that \( wsw \) resp., \( \tilde{wsw} \) is a substring of \( t \) such that \( w \) has period \( d \), \( |w| \geq 2d \), and \( \tilde{wsw} \) resp., \( wsw \) has no period \( d \). Let \( t[i..j] \) be a run with period \( d \) containing the first occurrence of \( w \) resp., \( \tilde{w} \) in \( wsw \) resp., \( \tilde{wsw} \). Denote by \( h \) the starting position of \( s \). Then, we have \( h = j - |\text{pre}_d(s)| + 1 \) or \( h \in (j+1-d..j'+1] \) or \( h \in (j-|s|\cdots..j-|s|] \).
Proof. If \( h + |s| > j \), then, by Lemma~\[8\] either \( h = j - |\text{pred}(s)| + 1 \) or \( h \in (j+1-d..j+1) \). Suppose that \( h + |s| \leq j \). Let \( t'[i..j'] \) be a run with period \( d \) containing the last occurrence of \( w \) in \( \text{esw} \) [resp., \( \text{esw}^r \)]. Clearly, \( i' \leq h + |s| \). Hence, since \( t[i..j] \) and \( t'[i..j'] \) cannot overlap on \( d \) letters, we obtain \( j - d + 1 < h + |s| \). Therefore, \( h \in (j-|s| - d..j-|s|) \).

Lemma 10. Let \( v = t[h_1..h_2] \) be a string with the minimal period \( d \leq \frac{|v|}{2} \). Given \( z, z' \) such that \( 1 < z < z' < r \), we can find all instances \( t[i..j] = s_1 w_{s_2} w_2 \cdots s_{r-1} w_{r-1} s_r \) of \( p \) such that \( \frac{3}{2}|v| < |w_1| = \cdots = |w_{r-1}| \leq 2|v| \), \( i + |s_1| \leq h_1 \leq h_2 \leq i + |s_1|w_1|_j \), \( w_{s_2} w_2 \cdots s_{r-1} w_{r-1} \) and \( w_{s_2} w_2 \cdots s_{r-1} w_{r-1} \) both have period \( d \), and \( w_{s_2} w_2 \cdots s_{r-1} w_{r-1} \) both have no period \( d \) in \( O(r + \frac{r|v|}{\log \log n}) \) time plus \( O(\frac{\log n}{\log \log n}) \) time, if \( \frac{\log \log n}{\log \log n} \leq |v| \leq \log n \).

Proof. We can find a run \( t[i'..j'] \) with period \( d \) containing \( v = t[h_1..h_2] \) in \( O(1) \) time using Lemma~\[3\]. Let \( t[i..j] = s_1 w_{s_2} w_2 \cdots s_{r-1} w_{r-1} s_r \) be an instance of \( p \) satisfying the conditions in the statement of the lemma. Let us find all possible runs with period \( d \) that can contain \( w_z \). Denote by \( h \) the start position of \( s_z \). Since \( |w_1| \geq \frac{3}{2}|v| \geq 3d \), Lemma~\[3\] implies that \( h = j' - |\text{pred}(s_z)| + 1 \) or \( h \in (j' + 1 - d..j' + 1) \) or \( h \in (j' - |s_z| - d..j' - |s_z|) \). Suppose that \( h \in (j' + 1 - d..j' + 1) \) [resp., \( h \in (j' - |s_z| - d..j' - |s_z|) \)], \( h = j' - |\text{st}(s_z)| + 1 \). Then, any run containing \( w_z \) contains the substring \( t[j' + |s_z| + 1..j' + |s_z| + 2d] \) [resp., \( t[j' + 2d - 1] \), \( h + |s_z| - h + |s_z| + 2d] \]. We find a run with period \( d \) containing this substring in \( O(1) \) time by Lemma~\[3\]. Thus, we have three possible locations for a run with period \( d \) containing \( w_z \) (note that the found runs can coincide). We process each of the found runs separately: let \( t[i'..j'] \) be one of these three runs. So, suppose that \( t[i'..j'] \) and \( t[i''..j''] \) are runs with period \( d \) containing the substrings \( w_z w_2 \cdots s_{r-1} w_{r-1} \) and \( w_z s_{r-1} w_{r-1} \), respectively.

Denote by \( h' \) the starting position of \( s_{r-1} \). Note that \( t[h..h' - 1] = s_1 w_{s_2} w_2 \cdots s_{r-1} w_{r-1} \). Hence, if the number \( h' - h \) is known, we can calculate \( |w_1| = h' - h - |s_{r-1}| - 1 \), apply Lemma~\[7\] for \( \beta = |w_1| \), and thus find all corresponding instances of \( p \) in \( O(r + \frac{r|v|}{\log \log n}) \) time plus \( O(\frac{\log n}{\log \log n}) \) time, if \( \frac{\log \log n}{\log \log n} \leq |v| \leq \log n \). So, our aim is to find a constant number of possible values for \( h' - h \) and process each of them with the aid of Lemma~\[7\].

Suppose that \( x_{z-1} = x_{z-1} \). By Lemma~\[4\] since \( t[h-d..h - 1] = t[h'-d..h'-1] \), we have \( h - \ell \equiv h' - \ell' \) (mod \( d \)), where \( \ell \) and \( \ell' \) are the starting positions of Lyndon roots of \( t[i'..j'] \) and \( t[i''..j''] \), respectively; \( \ell \) and \( \ell' \) can be found in \( O(1) \) time by Lemma~\[6\]. So, we obtain \( h' - h \equiv \ell - \ell' \) (mod \( d \)). By Lemma~\[6\] \( h \) either equals \( j' - |\text{pred}(s_z)| + 1 \) or lies in one of the segments \( S_1 = (j' + 1 - d..j' + 1) \) or \( S_2 = (j' - |s_z| - d..j' - |s_z|) \) of length \( d \); similarly, \( h' \) either equals \( j'' - |\text{pred}(s_{r-1})| + 1 \) or lies in \( S_1' = (j'' + 1 - d..j'' + 1) \) or \( S_2' = (j'' - |s_{r-1}| - d..j'' - |s_{r-1}|) \).

For each \( h \in S_1 \), there exists exactly one \( h' \in S_1' \) such that \( h' - h \equiv \ell - \ell' \) (mod \( d \)); moreover, each can easily prove that in this case \( h' - h \) is equal to either \( j'' - j' - \theta \) or \( j'' - j' - \theta + d \), where \( \theta = ((j'' - \ell') - (j' - \ell)) \mod d \) (see Fig.~\[2\]). For each of these values of \( h' - h \), we apply Lemma~\[7\] putting \( \beta = \frac{h' - h - |s_{r-1}| - 1}{d} \). Other combinations \( h \in S_1, h \in S_2 \) and \( h' \in S_1', h \in S_2 \) and \( h' \in S_2' \) are analogous; the cases when either \( h = j' - |\text{pred}(s_z)| + 1 \) or \( h' = j'' - |\text{pred}(s_{r-1})| + 1 \) are even simpler. See the case \( x_{z-1} = x_{z-1} \) in Appendix.

Figure 2 The case \( h' - h = j'' - j' - \theta \) in the proof of Lemma~\[10\]

Denote by \( Z \) [resp., \( Z', Z'' \)] the set of all numbers \( z \in (1..r) \) such that \( x_{z-1} = x_z \) [resp., \( \overline{x}_{z-1} = \overline{x}_z, x_{z-1} = \overline{x}_z \)]. Using Lemma~\[1\] one can easily prove the following lemma:
Detecting Unary Patterns

 Lemma 11. Let w be a string of length \( \geq d \). For \( z \in [1..r] \), denote \( w_z = w \) if \( x_z = x \), and \( w_z = \overline{w} \) if \( x_z = \overline{x} \). For any numbers \( z_1, z_2 \in \mathbb{Z} \) [resp., \( z_1, z_2 \in \mathbb{Z}' \); \( z_1, z_2 \in \mathbb{Z}'' \)], if the strings \( w_{z_1}w_{z_2} \) and \( w_{z_2}w_{z_1} \) both have period \( d \), then the following properties hold:

\[
x_i = x_{i + d} \quad \mod d, \quad s_{z_1} \text{ and } s_{z_2} \text{ both have period } d.
\]

(1)

In a sense, the converse is also true: if \( |z_1| \geq d \), \( w_{z_1}w_{z_2}w_{z_3} \) has period \( d \), and \( z_1 \) and \( z_2 \) satisfy (1), then the string \( w_{z_1}w_{z_2}w_{z_3} \) necessarily has period \( d \).

We call a pair of numbers \( (z, z') \) such that \( z \leq z' \) and \( z, z' \in \mathbb{Z} \) a separation in \( \mathbb{Z} \) if any numbers \( z_1, z_2 \in \{ (1..z) \cup (z..z') \} \cap \mathbb{Z} \) satisfy (1) and any numbers \( z_1 \in \{ (1..z) \cup (z..z') \} \cap \mathbb{Z} \) and \( z_2 \) do not satisfy (1). Obviously, if \( (z, z') \) is a separation, then \( (z, z') \) is also a separation. We find all separations in \( \mathbb{Z} \) of the form \( (z, z) \) applying the following lemma with \( Z_0 = \mathbb{Z} \).

 Lemma 12. For any subset \( Z_0 \subset \mathbb{Z} \) [resp., \( Z_0 \subset \mathbb{Z}' \); \( Z_0 \subset \mathbb{Z}'' \)], there are at most three numbers \( z \in Z_0 \) satisfying the following properties (2): all such \( z \) can be found in \( \mathcal{O}(r) \) time.

\[
\begin{align*}
\text{any } z_1, z_2 \in (1..z) \cap Z_0 \text{ satisfy (1),} \\
\text{any } z_1 \in (1..z) \cap Z_0, z_2 = z \text{ either do not satisfy (1) or satisfy } |s_{z_1}| < d \leq |s_{z_2}|.
\end{align*}
\]

(2)

Proof. Let \( z' = \min \{ z \in Z_0 \} \). Clearly, \( z = z' \) satisfies (2). Using the lcp data structure on the string \( \overline{p} \), we find in \( \mathcal{O}(r) \) time the minimal number \( z'' \in Z_0 \) such that \( z_1, z_2 \in [z'..z''] \cap Z_0 \) satisfy (1) and some \( z_1, z_2 \in [z'..z''] \cap Z_0 \) do not satisfy (1); assume \( z'' = +\infty \) if there is no such \( z'' \). Obviously, if \( z' = +\infty \), then \( z = z'' \) satisfies (2). Any \( z \in (z'..+\infty) \cap Z_0 \) does not satisfy (2) because in this case \( z'' = +\infty \) and some \( z_1, z_2 \in [z'..z''] \cap Z_0 \) do not satisfy (1). In \( \mathcal{O}(r) \) time we find the minimal \( z'' \in [z'..z''] \cap Z_0 \) such that \( |s_{z''}| \geq d \); assume \( z'' = z' \) if there is no such \( z'' \). By the definition, we have \( s_{z_1} = s_{z_2} \) and \( |s_{z_1}| = |s_{z_2}| < d \) for any \( z_1, z_2 \in [z'..z''] \cap Z_0 \). Therefore, any \( z \in (z'..z'') \cap Z_0 \) does not satisfy (2). Further, any \( z \in (z'..z'') \cap Z_0 \) does not satisfy (2) since in this case \( z_1 = z'' \) and \( z_2 = z \) satisfy (1) and \( |s_{z_1}| \geq d \), which contradicts to (2). Finally, if \( z'' = +\infty \), then \( z = z'' \) obviously satisfies (2). So, \( z' \), \( z'' \), \( z''' \) are the only possible numbers in \( Z_0 \) that can satisfy (2).

Finally, for each found separation \( (z_0, z_0) \) in \( Z \), we apply Lemma 12 with \( Z_0 = Z \setminus \{ z_0 \} \) and thus obtain all separations in \( Z \) of the form \( (z_0, z'_0) \) for some \( z'_0 > z_0 \). By Lemma 12 we obtain at most 12 separations in \( Z \) in total and this set of separations is exhaustive.

In-a-run instances of \( p \). Now we find not only related to \( v \) instances of \( p \) whose substitutions all lie in the run \( t[i..j] \) but we process the whole run \( t[i..j] \) (we process each such run in this way only once) and find all instances \( t[i..j] \) of \( p \) satisfying the following properties:

\[
t[i + |s_1|..j] \text{ is a substring of } t[i'..j']
\]

(3)
Let \( t[i..j] \) satisfy (3) and \( w \) be a substitution of \( x \) in \( t[i..j] \). W.l.o.g., assume that \( x_{r-1} = x \) (the case \( x_{r-1} = \overline{x} \) is symmetrical). Suppose that \( p \neq s_1 x s_2 \) and \( p \neq s_1 \overline{x} s_2 s_3 \) (\( p = s_1 x s_2 \) was considered in preliminaries; \( p = s_1 x s_2 x s_3 \) is considered in Appendix). Then, either there is \( z \in (1..r) \) such that \( x_{z-1} = x_z \) or there are \( z', z'' \in (1..r) \) such that \( x_{z'-1} s x_{z'} = \overline{x} s x_{z''} \) and \( x_{z''-1} s x_{z''} = x s x_{z''} \). Therefore, the number \(|w| \mod d\) can be calculated as follows.

\[ \]

**Lemma 13.** Let \( t[i..j] \) satisfy (3) and \( w \) be a substitution of \( x \) in \( t[i..j] \). If, for \( z \in (1..r) \), \( x_{z-1} = x_z \), then \(|w| \equiv -|s_z| \mod d\); if, for \( z' \), \( z'' \in (1..r) \), \( x_{z'-1} s x_{z'} = \overline{x} s x_{z''} \) and \( x_{z''-1} s x_{z''} = x s x_{z''} \), then either \(|w| \equiv \frac{d-|s_{z''}| - |s_z|}{2} \mod d\) or \(|w| \equiv \frac{-|s_{z''}| - |s_z|}{2} \mod d\).

**Proof.** Suppose that \( x_{z-1} = x_z \). Since, by Lemma 1, the distance between any two occurrences of \( w \) or \( w' \) in \( t[i'..j'] \) is a multiple of \( d \), we have \(|w| \equiv -|s_z| \mod d\). See Appendix for the case of \( z', z'' \in (1..r) \) such that \( x_{z'-1} s x_{z'} = \overline{x} s x_{z''} \) and \( x_{z''-1} s x_{z''} = x s x_{z''} \).

Consider the segments \( \{ (j'+1-bd..j'+1-(b-1)d) \}_{b=1}^f \), where \( f \) is the maximal integer such that \( j'+1-fd \geq i' \). For each \( b \in [1..f] \), our algorithm finds in \( O(r \log \frac{1}{d} \cdot \frac{r}{d}) \) time using Lemma 16 below all strings \( t[s..j] \) satisfying (3) such that \( j-s_r+1 \in (j'+1-bd..j'+1-(b-1)d) \). Clearly, in this case we will find all instances of \( p \) satisfying (3) in \( O(r \log \frac{1}{d} \cdot \frac{r}{d}) \) time.

Since \( j'-i' + 1 \geq 3d \), it follows from Lemma 16 and from Lemma 2 that the sum of the values \( \frac{j'-i' + 1}{d} \frac{d}{2} \frac{d}{2} \) over all such runs \( t[i'..j'] \) is \( O(n) \) and hence the overall time is \( O(rn) \).

Since \( t[i'..j'] \) has period \( d \), the following lemma is straightforward.

**Lemma 14.** Let \( t[i..j] \) be a string satisfying (3) such that \( i' \leq i \). Then \( t[i+(r-1)d..j] \) is an instance of \( p \) and, if \( i = - (r-1)d \leq i' \), \( t[i-(r-1)d..j] \) is also an instance of \( p \).

**Lemma 15.** Suppose that \( x_1 = \cdots = x_{r-1} \). Given a run \( t[i'..j'] \) with period \( d \), a segment \( [b..b] \subset [i'..j'] \) of length \( d \), and an integer \( \eta \geq d \), we can compute in \( O(r \log \frac{1}{d} \cdot \frac{r}{d}) \) time a bit array \( D'[b..b] \) such that, for \( h \in [b_1..b_2] \), \( D'[h] = 1 \) if the string \( t[h-|s_{1} s_{2} \cdots s_{r-1}|-(r-1)\eta..h+|s_r-1|] \) is an instance of \( p \) and \( h-|s_{2} s_{3} \cdots s_{r-1}|-(r-1)\eta \geq i' \).

**Proof.** We obtain a bit array \( D'[b_1..b_2] \) performing in \( O(r \log \frac{1}{d} \cdot \frac{r}{d}) \) time the bitwise “and” of the subarrays \( \{ D_2[b_i - \gamma_2..b_i - \gamma_2 - \gamma_z] \}_{z=1}^r \), where \( \gamma_z = (r-z)\eta-|s_{2} s_{3} \cdots s_{r-1}| \). In \( O(d \log \frac{1}{d} \cdot \frac{r}{d}) \) time we fill with zeros a subarray \( D'[b_1..b] \) for the maximal \( b \in [b_1..b_2] \) such that \( b-(r-1)\eta-|s_{2} s_{3} \cdots s_{r-1}| < i' \) (if any). Since \( x_1 = \cdots = x_{r-1} \) and \( \eta \geq d \), it follows from Lemma 3 that, for \( h \in [b_1..b_2] \) such that \( D'[h] = 1 \), \( t[h-|s_{1} s_{2} \cdots s_{r-1}|-(r-1)\eta..h+|s_r-1|] \) is an instance of \( p \) if \( \eta \equiv -|s_z| \mod d \) for all \( z \in (1..r) \) (i.e., if substitutions of \( x \) are aligned properly). So, if \( \eta \equiv -|s_z| \mod d \) for some \( z \in (1..r) \), then we fill \( D'[b_1..b_2] \) with zeros.

For each \( \delta = 3d + \delta' \), where \( \delta' \in [0..d] \) is one of the possible values of \(|w| \mod d \) described in Lemma 13, we process each segment \( [b_1..b_2] = (j'+1-bd..j'+1-(b-1)d) \), for \( b \in [1..f] \), by the following lemma (the cases \( p = s_1 x s_2 x s_3 \) and \( p = s_1 \overline{x} s_2 x s_3 \) are considered in Appendix).

**Lemma 16.** Let \( p \neq s_1 x s_2 x s_3 \), \( p \neq s_1 \overline{x} s_2 s_3 \), \( r \geq 3 \). Given a run \( t[i'..j'] \) with period \( d \), a number \( \delta \geq d \), and a segment \( [b..b] \subset [i'..j'+1] \) of length \( d \), we can compute in \( O(r \log \frac{1}{d} \cdot \frac{r}{d}) \) time numbers \( d', d', h', h', a', a' \) and bit arrays \( E[b_1..b_2] \), \( F[b_1..b_2] \) such that:

1. for any \( h \in [b_1..h'] \) [resp., \( h \in (h'..b_2'] \), we have \( E[h] = 1 \) iff the strings \( t[h-|s_{1} s_{2} \cdots s_{r-1}|-(r-1)\delta+cd..h+|s_r-1|] \) for all \( c \in [0..d'] \) [resp., for all \( c \in [0..d'] \)] are instances of \( p \) and \( h-|s_{1} s_{2} \cdots s_{r-1}|-(r-1)\delta+cd \geq i' \);
2. for any \( h \in [b_1..h'] \) [resp., \( h \in (h'..b_2'] \), we have \( F[h] = 1 \) iff the string \( t[h-|s_{1} s_{2} \cdots s_{r-1}|-(r-1)\delta+ad..h+|s_r-1|] \) where \( a = a' \) [resp., \( a = a'' \)] is an instance of \( p \) and \( h-|s_{2} s_{3} \cdots s_{r-1}|-(r-1)((\delta + ad) \geq i' \).
in addition, we find at most one instance \( t[i_0..j_0] = s_1w_1s_2w_2 \cdots w_{r-1}s_r \) of \( p \) satisfying \((3)\) and such that \( j_0 - s_r + 1 \in [b_1..b_2] \), \( |w_1| \equiv \delta \mod d \), and it is guaranteed that if a string \( t[i..j] = s_1w_1s_2w_2 \cdots w_{r-1}s_r \) satisfies \((3)\), \( j - |s_r| + 1 \in [b_1..b_2] \), and \( |w_1| \equiv \delta \mod d \), then either \( t[i..j] \) is encoded in one of the arrays \( E \), \( F \) or \( i = i_0 \) and \( j = j_0 \).

**Proof.** Choose \( h \in [b_1..b_2] \). Denote \( i_h = h - |s_1s_2 \cdots s_{r-1}| - (r - 1)\delta \), \( j_h = h + |s_r| - 1 \), and \( c_h = \frac{|s_{r+1} \cdots s_p|}{|s_1|} \) . By Lemma 14, if a string \( t[i..j] = s_1w_1s_2w_2 \cdots s_{r-1}w_{r-1}s_r \) satisfies \((3)\), \( i \leq i_h \), \( j \geq j_h \), \( |w_1| \equiv \delta \mod d \), then \( t[i_h..j_h] \) is an instance of \( p \) and \( t[i..j] = t[i_h-(r-1)c_d..j_h] \) for some \( c \in [0..c_h] \); conversely, if \( t[i_h..j_h] \) is an instance of \( p \), then all substrings \( t[i_h-(r-1)c_d..j_h] \), for \( c \in [0..c_h] \), are instances of \( p \). By the definition of \( c_h \), there is a threshold \( h' \in [b_1..b_2] \) such that, for any \( h_1, h_2 \in [b_1..b_2] \), \( c_h = c_{h'} \) if \( h_1, h_2 \in [b_1..h'] \) or \( h_1, h_2 \in (h',b_2] \), and \( c_{h_1} - c_{h_2} = 1 \), otherwise; \( h' \) can be found in \( O(1) \) time by simple calculations. Put \( d' = a_h' \) and \( d'' = a_{h'+1} \). Suppose that \( x_1 = \cdots = x_{r-1} \). Then, applying Lemma 15 for \( \eta = \delta \), we compute the required bit array \( E[i_0..b_2] \). Now it remains to find all strings \( t[i..j] \) satisfying \((3)\) and such that \( i < i' \) and \( j - |s_r| + 1 \in [b_1..b_2] \).

Let \( t[i..j] = s_1w_1 \cdots s_{r-1}w_{r-1}s_r \) satisfy \((3)\), \( j - |s_r| + 1 \in [b_1..b_2] \), \( |w_1| \equiv \delta \mod d \), and \( i < i' \). Denote \( h = j - |s_r| + 1 \). By a symmetric version of Lemma 8 we have either \( i + |s_1| \in [i'..i'+d) \) or \( i + |s_1| = i' + |s_{\text{uf}}(s_1)| \). Suppose that \( i + |s_1| \in [i'..i'+d) \). By Lemma 13 we have \( i + |s_1| = a_h - |s_{\text{uf}}(s_1)| - (r - 1)\delta + a_0d \), where \( a_0 \) is the maximal integer such that \( h - |s_{\text{uf}}(s_1)| - (r - 1)\delta + a_0d \geq i' \). The definition of \( a_0 \) implies that there is \( h' \in [b_1..b_2] \) such that, for any \( h_1, h_2 \in [b_1..b_2] \), \( a_h = a_{h'} \) if \( h_1, h_2 \in [h',b_2] \), and \( a_{h_1} - a_{h_2} = 1 \), otherwise; \( h'' \) can be simply found in \( O(1) \) time. Put \( d' = a_{h''} \) and \( d'' = a_{h'+1} \). Suppose that \( x_1 = \cdots = x_{r-1} \). Then, we compute two bit arrays \( F[b_1..b_2] \) and \( F[b_1..b_2] \) applying Lemma 15 for \( \eta = \delta + d' \) and \( \eta = \delta + d'' \), respectively. Finally, we concatenate the arrays \( F_1[b_1..b'''] \) and \( F_2[b'''+1..b_2] \) to obtain \( F[b_1..b_2] \).

Suppose that \( i + |s_1| = i' + |s_{\text{uf}}(s_1)| \). Since \( t[i+h+1..h-1] = s_1w_1s_2w_2 \cdots s_{r-1}w_{r-1} \), we have \( h - (i' + |s_{\text{uf}}(s_1)|) \equiv |s_{\text{uf}}(s_1)| - (r - 1)\delta \mod d \). Since \( b_2 - b_1 + 1 \) is a divisor, there is exactly one position \( h \in [b_1..b_2] \) such that \( h \equiv \delta \mod d \) can be found in \( O(1) \) time. We check whether \( t[i''] + |s_{\text{uf}}(s_1)| - |s_1| \geq |s_{\text{uf}}(s_1)| \) is an instance of \( p \) in \( O(r) \) time using the lcp data structure and the arrays \( \{D_x\}_{x=1}^z \); thus, we may find an additional instance of \( p \) that is not encoded in \( E \) and \( F \). See the case when \( p \) contains both \( x \) and \( \overline{x} \) in Appendix. 

**In-two-runs instances of \( p \)**. Like in the case of one run, our algorithm for two runs processes each run \( \{i'..\} \) with period \( d \) (only once) and finds all instances of \( p \) whose substitutions have length \( \geq 3d \) and lie in exactly two runs: \( \{i'..\} \) and another run with period \( d \).

Choose \( z \in [1..r] \). Let \( t[i..j] = s_1w_1s_2w_2 \cdots s_{r-1}s_r \) be an instance of \( p \) such that \( |w_1| = \cdots = |w_{r-1}| \geq 3d \), \( t[i+h+1..h-1] = \) a substring of \( t[i'..] \), and \( t[h+1..h] = |s_{z+1}| - |s_r| \) is a substring of another run with period \( d \), where \( h = i + |s_1w_1 \cdots s_{z-1}w_z| \). We call \( z \) a separator in \( t[i..j] \). Obviously, the string \( w_1s_{z+1}w_{z+1} \) has no period \( d \). Hence, by Lemma 9 we have \( h \in (j'+1-d'-j'+1) \) or \( h \in (j'-|s_{z+1}| - d'-|s_{z+1}|) \) or \( h = j' - |s_{z+1}| + 1 \). Suppose that \( h \in (j'+1-d'-j'+1) \) (two other cases are similar). Denote \( b_1 = j' - d - 2 \) and \( b_2 = j' + 1 \).

Since \( |w_1| \geq 3d \), the string \( t[h+1..h] + |s_{z+1}w_{z+1}| - 1 = w_{z+1} \) contains the substring \( t[b_2+1..b_2] = |s_{z+1}s_{z+2}| + 2d - 1 \). So, using Lemma 3 for the latter substring, we find in \( O(1) \) time a run \( \{i'..\} \) with period \( d \) containing \( w_{z+1} \). Clearly, the strings \( t[i+h..h] \) and \( t[h+1..h] \) are instances of the patterns \( s_1x_1 \cdots x_zs_{z+2} \) and \( x_zs_{z+2}w_1s_2w_2 \cdots s_{r-1}s_r \), respectively, and \( t[i+|s_1..h-1|] \) and \( t[h+|s_{z+1}|..h] \) are substrings of the runs \( t[i'..] \) and \( t[i'..] \), respectively. Hence, if either there is \( z' \in (1..r) \backslash \{z+1\} \) such that \( x_{z'} = x \) or there are \( z', z'' \in (1..r) \backslash \{z+1\} \) such that \( x_{z'} = x_0 \) and \( x_{z''} = \overline{x}_0 \), then the number \( |w_1| \mod d \) is equal to one of the values described in Lemma 13; let \( \delta' \in [0..d] \) be
one of these values (we process each such \( \delta' \)). Otherwise (if we could not find such \( z' \) and \( z'' \)), we have \( r \leq 5 \) and we can compute a similar value \( \delta'' \) as follows. If \( p \neq s_1 x s_2 x s_3 \) and \( p \neq s_1 x s_2 x s_4 \), then there are \( z' \in \{1, z\} \) and \( z'' \in \{z + 1, r\} \) such that \( x_{z'} = x_{z''} \). Denote by \( \ell \) and \( \ell' \) the starting positions of Lyndon roots of \( t[i..j'] \) and \( t[i''..j''] \), respectively; \( \ell \) and \( \ell' \) can be computed in \( O(1) \) time by Lemma 11. It follows from Lemma 11 that \( |j' + 1| = \ell' \) and apply the following lemma for each of them.

Therefore, \( |s_{z' + 1} w_{z' + 1} \cdots s_{z''} w_{z''}| = \ell' - \ell \) (mod \( d \)) and hence \( (z'' - z')|s_{z' + 1} w_{z' + 1} \cdots s_{z''} w_{z''}| \equiv \ell' - \ell \) (mod \( d \)) and \( \ell' \). This equation has at most \( z'' - z' \) solutions: \( |w| \equiv \frac{c d + \ell - \ell'}{z'' - z'} \) (mod \( d \)) for \( c \in \{0, z'' - z'\} \); let \( \delta'' \) be one of these (we process all such \( \delta'' \); since \( z'' - z' \leq r - 3 \leq 2 \), there are at most two such \( \delta'' \)). See the cases \( p = s_1 x s_2 x s_3 \) and \( p = s_1 x s_2 x s_4 \) in Appendix. Denote \( \delta = 3d + \delta'' \). It follows from Lemma 11 that any separator \( z \) such that \( z \in Z \) [resp., \( z \in Z' \), \( z \in Z'' \)] satisfies \( \delta \) for \( Z_0 = Z \) [resp., \( Z_0 = Z' \), \( Z_0 = Z'' \)]. So, we find a constant number of “suspected” separators using Lemma 12 and apply the following lemma for each of them.

\textbf{Lemma 17.} Given \( z \in \{1, r\} \), two runs \( t[i..j'] \) and \( t[i''..j''] \) with period \( d \), a number \( \delta \geq d \), and a segment \( [b_1, b_2] \subseteq [i''..j''] \) of length \( d \), we can compute in \( O(d \log n) \) time numbers \( \delta', a', a'' \) and bit arrays \( B[b_1, b_2], B'[b_1, b_2] \) such that:

1. for any \( h \in [b_1, b_2] \) [resp., \( h \in (h', b_1] \) or \( h \in (b_2, b_2] \) or \( h \in [b_1, b_2] \)], we have \( |H[h]| = 1 \) iff the strings \( t[h - |s_{z_1} \cdots s_{z_2} - z\delta + \delta]h + |s_{z_1} \cdots s_{z_2} + s_{r_1} + (r - 1 - z)\delta + \delta - 1| \), for all \( c \in [0, d'] \) [resp., \( c \in \{0, d''\} \), \( c \in \{0, d''\} \), \( c \in \{0, d''\} \}], are instances of \( p \) and \( \ell' \leq h - |s_{z_1} \cdots s_{z_2} - z\delta + \delta| \leq h + |s_{z_1} \cdots s_{z_2} + s_{r_1} + (r - 1 - z)\delta + \delta - 1 \) \leq \( j'' \);

2. for any \( h \in [b_1, b_2] \) [resp., \( h \in (h', b_1] \) or \( h \in (b_2, b_2] \) or \( h \in [b_1, b_2] \)], we have \( |H[h]| = 1 \) iff the string \( t[h - |s_{z_1} \cdots s_{z_2} - z\delta + \delta]h + |s_{z_1} \cdots s_{z_2} + s_{r_1} + (r - 1 - z)\delta + \delta - 1| \), for all \( c \in [0, d'] \) [resp., \( c \in \{0, d''\} \), \( c \in \{0, d''\} \), \( c \in \{0, d''\} \)], is an instance of \( p \) and \( \ell' \leq h - |s_{z_1} \cdots s_{z_2} - z\delta + \delta| \leq h + |s_{z_1} \cdots s_{z_2} + s_{r_1} + (r - 1 - z)\delta + \delta - 1 \) \leq \( j'' \);

in addition, we find at most two instances \( t[i_0..j_0] = s_1 w_1 \cdots w_{r-1}s_r \) of \( p \) such that \( |w_1| = \cdots = |w_{r-1}| \geq 3d \), \( |w_1| \equiv \delta \) (mod \( d \), \( \ell' \leq i_0 + |s_1| \leq j_0 - |s_r| \leq j'' \), \( i_0 + |s_1|, i_0 + |s_2| \in [b_1, b_2] \), and it is guaranteed that any instance \( t[i..j] = s_1 w_1 \cdots w_{r-1}s_r \) of \( p \) such that \( \ell' \leq i + |s_1| \leq j - |s_r| \leq j'' \), \( |w_1| = \cdots = |w_{r-1}| \geq 3d \), \( |w_1| \equiv \delta \) (mod \( d \), \( i_0 + |s_1|, i_0 + |s_2| \in [b_1, b_2] \) either is encoded in the arrays \( E, F \) or is represented by one of the additional instances.

\textbf{Proof.} Denote \( p_1 = s_1 x s_2 \cdots x s_2 \) and \( p_2 = x_{z_1} s_{z_2} \cdots x_{r-1}s_r \). We apply Lemma 11 putting \( p := p_1 \) to compute numbers \( d_1', d_1'', h_1', a_1', a_2' \), bit arrays \( E_1[b_1, b_2], F_1[b_1, b_2] \), and, probably, one additional instance \( t[i_0, j_0] \) of \( p_1 \) that all together represent all instances \( t[i..j] = s_1 w_1 \cdots w_{r-1}s_r \) of \( p_1 \) such that \( |w_1| = \cdots = |w_{r-1}| \geq 3d \), \( |w_1| \equiv \delta \) (mod \( d \)), \( \ell' \leq i + |s_1| \leq j', \) and, if and only if \( j' = j, j + 1 \in [b_1, b_2] \). Similarly, putting \( p := p_2 \), we apply a symmetrical version of Lemma 11 to obtain numbers \( d_2', d_2'', h_2', a_2', a_2'' \), bit arrays \( E_2[b_1 + |s_{z_1}|, b_2 + |s_{z_1}|], F_2[b_1 + |s_{z_1}|, b_2 + |s_{z_1}|] \), and, probably, one additional instance \( t[i_0, j_0] \) of \( p_2 \) that together represent all instances \( t[i..j] = w_{z_1} s_{z_2} \cdots w_{r-1}s_r \) of \( p_2 \) such that \( |w_{z_1}| = \cdots = |w_{r-1}| \geq 3d \), \( |w_{z_1}| \equiv \delta \) (mod \( d \)), \( \ell' \leq i + |s_{z_1}| \leq j'' \), \( i \in [b_1 + |s_{z_1}|, b_2 + |s_{z_1}|] \). We combine found instances of \( p_1 \) and \( p_2 \) to obtain all required instances of \( p \) as follows.

To combine instances of \( p_1 \) and \( p_2 \) encoded in the arrays \( E_1[b_1..b_2] \) and \( E_2[b_1 + |s_{z_1}|, b_2 + |s_{z_1}|] \), we perform in \( O(\frac{d}{\log n}) \) time the bitwise “and” of these arrays and the bit array \( D_{z_1}[b_1, b_2] \) and thus obtain a bit array \( E[b_1..b_2] \). Further, we find in \( O(\frac{d}{\log n}) \) time one arbitrary position \( h \in [b_1, b_2] \) such that \( |E[h]| = 1 \). In order to “synthesize” instances of \( p_1 \) and \( p_2 \), we check in \( O(1) \) time using the lcp data structure whether \( |l[h-d..h-1]| = t[h + |s_{z_1}|..h + |s_{z_1}| + d - 1] \), if \( x_z = x_{z_1} \), or \( |\ell[h-d..h-1]| = t[h + |s_{z_1}|..h + |s_{z_1}| + d - 1] \), if \( x_z \neq x_{z_1} \), and if not, then we fill \( E \) with zeros. One can show that \( E \) satisfies the conditions in the statement of the lemma provided \( h' = \min(h_1', h_2'), h_0' = \max(h_1', h_2'), d' = \min(d_1', d_2') \),
Detecting Unary Patterns

\[ d'' = \min\{d''_1, d''_2\}, \quad d' = \min\{d''_1, d'_2\} \text{ if } b'_1 \leq b'_2, \quad \text{and } d'' = \min\{d'_1, d'_2\} \text{ if } b'_1 > b'_2. \]

We apply a similar analysis for all remained combinations: \(E_1\) and \(F_2\), \(F_1\) and \(E_2\), \(F_1\) and \(F_2\); but due to the definitions of the arrays \(F_1\), \(F_2\) and the numbers \(a'_1, a''_1, a'_2, a''_2, h''_1, h''_2\), we can combine the results into one bit array \(F[b_1..b_2]\) putting \(h'' = \min\{h''_1, h''_2\}, h'_0 = \max\{h''_1, h''_2\}, a' = \min\{a'_1, a'_2\}, a'' = \min\{a''_1, a''_2\}\) if \(b''_1 \leq b''_2\), and \(a'' = \min\{a'_1, a'_2\}\) if \(b''_1 > b''_2\). Finally, we try to “extend” in an obvious way the instance \(t[i_0..j_0]\) of \(p_1\) [similarly, \([i_0''..j_0'']\) of \(p_2\)] to a full instance of \(p\) in \(O(r)\) time using the lcp data structure and the arrays \(\{D_z\}_{z=1}^2\). Thus, we obtain at most two additional instances of \(p\).

3 Solving Problem 2

The following theorem sums up our algorithms for the case when no reversed variables \(\bar{x}\) occur in \(p\) and the number of occurrences of \(x\) is at least 2 (see the proof in Appendix).

**Theorem 18.** Suppose that \(x_1 = \cdots = x_{r-1}\) and \(r \geq 3\); then one can report in \(O(rn)\) time all instances of \(p\) in \(t\) encoded in the sets of tuples \(\{(\delta_h, F_h[a_h..b_h])\}_{h \in \mathcal{H}}\) and \(\{(\delta_h, d_h, t[i_h..j_h]), E_h[i_h..i_h + d_h - 1]\}_{h \in \mathcal{H}'}\), where \(F_h\) and \(E_h\) are bit arrays, such that:

1. for any \(h \in \mathcal{H}\) and \(h' \in [a_h..b_h]\), \(F_h[h'] = 1\) iff the string \(t[h'.\cdot h'] + s_1s_2\cdot s_r\cdot + (r - 1)\beta_h - 1\) is an instance of \(p\);
2. for any \(h \in \mathcal{H}'\), \(t[i_h..j_h]\) is a run with the minimal period \(d_h\) such that \(d_h = \frac{h - j_h}{3}\) and, for any \(h' \in [i_h..j_h + d_h]\), \(E_h[h'] = 1\) iff the strings \(t[h' + c d_h..h' + c d_h + s_1s_2.. s_r\cdot + (r - 1)(\delta_h + c d_h) - 1]\) are instances of \(p\) for all integers \(c \geq 0\) and \(r \geq c \geq 0\) satisfying \(h' + c d_h + s_1s_2.. s_r\cdot + (r - 1)(\delta_h + c d_h) - 1 \leq j_h\);
3. \(|\mathcal{H}| + |\mathcal{H}'| = O(n)\) and \(\sum_{h \in \mathcal{H}} (b_h - a_h + 1) + \sum_{h \in \mathcal{H}'} (j_h - i_h + 1) = O(n \log n)\).

One can slightly modify the proof of Theorem 18 to obtain a similar (but more complicated) compact representation for all instances of the patterns containing both \(x\) and \(\bar{x}\), and thus prove Theorem 4. Theorem 18 can be used to get an \(O(rn \log n)\) time solution to Problem 2.

**Theorem 19.** Problem 2 can be solved in \(O(rn \log n)\) time.

**Proof.** For the pattern \(p = \left(\prod_{j=1..k}(y_j s_{1,j} x s_{2,j} x \cdots s_{r-1,j} x s_{r,j})\right) y_{k+1}\), let \(p_j = s_{1,j} x s_{2,j} x \cdots s_{r-1,j} x s_{r,j}\) be its \(j\)th sub-pattern, for \(j \in [1..k]\). Our solution is based on the following approach. We obtain in \(O(rn)\) time for each \(j \in [1..k]\) a representation of all instances of \(p_j\) in the string \(t\) like in Theorem 18. The representation of these instances has \(O(n \log n)\) elements, and we can restructure it easily to contain separately, for all substitutions \(w\) of \(x\) that lead to an instance of \(p_j\), a similar succinct representation of the instances of \(p_j\) obtained via that precise substitution. For each \(j\) and \(w\) we sort the instances of \(p_j\), where \(x\) is replaced by \(w\), by their starting position, in linear time with respect to their number. Further, for a fixed \(w\), we detect whether there exists an instance of \(p\) in \(t\) by a greedy approach: if, for some \(\ell \geq 0\), \(t[1..i] \) ends with an instance of \(\left(\prod_{j=1..k}(y_j s_{1,j} x s_{2,j} x \cdots s_{r-1,j} x s_{r,j})\right)\) where \(x\) is replaced by \(w\), then we use the representation of the instances of \(p_{\ell+1}\) to get the first instance of this sub-pattern where \(x\) is replaced by \(w\), which starts on a position to the right of \(i + 1\) and ends on the leftmost position \(i_{\ell+1}\) among all the instances of \(p_{\ell+1}\) where \(x\) is replaced by \(w\). This strategy clearly works: we separate the instances of the sub-patterns \(p_j\) by mapping the in-between variables to strings as short as possible (this strategy is similar to the one used to match regular patterns in, e.g., [3]). Then, the last variable \(y_{k+1}\) must contain at least one symbol, namely the first symbol of the part of \(t\) that we were not able to cover with instances of the sub-patterns. When \(y_1\) and \(y_{k+1}\) are present in the pattern, we should ensure that \(p_1\) and \(p_k\) are not a prefix and, respectively, a suffix of \(t\) (so that the images of \(y_1\) and \(y_{k+1}\) are not empty). The detailed proof is given in Appendix. ▶
References

1. Amihood Amir and Igor Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.
2. Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
3. H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. The “runs” theorem. *arXiv preprint arXiv:1406.0263v4*, 2014.
4. Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.
5. M. Crochemore, C. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. Extracting powers and periods in a string from its runs structure. In *SPIRE*, volume 6393 of *LNCS*, pages 258–269. Springer, 2010.
6. M. Crochemore and W. Rytter. *Jewels of stringology*. World Scientific Publishing Co. Pte. Ltd., 2002.
7. Thomas Erlebach, Peter Rossmanith, Hans Stadtherr, Angelika Steger, and Thomas Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261:119–156, 2001.
8. H. Fernau, F. Manea, R. Mercas, and M. L Schmid. Pattern matching with variables: fast algorithms and new hardness results. In *STACS 2015*, volume 30 of *LIPIcs*, pages 302–315. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.
9. M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. of Computer and System Sciences*, 47(3):424–436, 1993.
10. Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O’Reilly, Sebastopol, CA, third edition, 2006.
11. Z. Galil and J. Seiferas. A linear-time on-line recognition algorithm for “palstar”. *J. of ACM (JACM)*, 25(1):102–111, 1978.
12. P. Gawrychowski, F. Manea, and D. Nowotka. Testing generalised freeness of words. In *STACS 2014*, volume 25 of *LIPIcs*, pages 337–349. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.
13. Juhani Karhumäki, Wojciech Plandowski, and Filippo Mignosi. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.
14. Michael Kearns and Leonard Pitt. A polynomial-time algorithm for learning $k$-variable pattern languages from examples. In *Proceedings of the 2nd Annual Conference on Learning Theory, COLT*, pages 57–71, 1989.
15. T. Kociumaka, J. Radoszewski, W. Rytter, and T. Waleń. Internal pattern matching queries in a text and applications. In *SODA 2015*, pages 532–551. SIAM, 2015.
16. R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *FOCS 1999*, pages 596–604. IEEE, 1999.
17. T. Kopelowitz and M. Lewenstein. Dynamic weighted ancestors. In *SODA 2007*, pages 565–574. SIAM, 2007.
18. D. Kosolobov. Online detection of repetitions with backtracking. In *CPM 2015*, volume 9133 of *LNCS*, pages 295–306. Springer International Publishing, 2015.
19. D. Kosolobov, M. Rubinchik, and A. M. Shur. Pal$^k$ is linear recognizable online. In *SOFSEM 2015*, volume 8939 of *LNCS*, pages 289–301. Springer, 2015.
20. M. Lothaire. *Algebraic Combinatorics on Words*, chapter 3. Cambridge University Press, Cambridge, New York, 2002.
21. G. Manacher. A new linear-time on-line algorithm finding the smallest initial palindrome of a string. *J. of ACM (JACM)*, 22(3):346–351, 1975.
22. Alexandru Mateescu and Arto Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique Theoretique et Applications*, 28:233–253, 1994.
14 Detecting Unary Patterns

23 Yen K. Ng and Takeshi Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397:150–165, 2008.

24 Sebastian Ordyniak and Alexandru Popa. A parameterized study of maximum generalized pattern matching problems. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation, IPEC*, 2014.

25 Daniel Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.

26 M. Rubinchik and A. M. Shur. Eertree: An efficient data structure for processing palindromes in strings. *arXiv preprint arXiv:1506.04862*, 2015.
Appendix

To Section 2.1

The continuation of the discussion before Lemma 7. We have \( v = t[q_1..q_1 + |v| - 1] \). Let \( q_2 \in [q_1 + |v_s2..q_1 + |v_{s22v}|| \) be the starting position of an occurrence of \( \overline{v} \). If \( p = s_1x\overline{s2x}x_3 \) or \( p = \overline{s1x\overline{s2x}x_3} \), then we apply the following lemma to find all instances \( t[i..j] = s_1w_2\overline{w}_3 \) of \( p \) such that \( \frac{3}{2}|v| < |w| \leq 2|v| \), \( i + |s_1| \leq q_1 < q_1 + |v| \leq i + |s_1w_2| \), and \( q_1 + |v| - (i + |s_1|) = (i + |s_1w_2w|) - q_2 \); the latter equality guarantees that the string \( t[q_2..q_2 + |v| - 1] \) in such instance is a reversal of \( t[q_1..q_1 + |v| - 1] \) produced by the substitution \( \overline{w} \) (see Fig. 3).

Lemma 20. Let \( p = s_1x\overline{s2x}x_3 \) or \( p = \overline{s1x\overline{s2x}x_3} \). Given a substring \( t[h_1..h_2] = v \) and a position \( q > h_2 \) such that \( t[q..q + |v| - 1] = \overline{v} \), we can compute in \( O(1 + \frac{|v|}{\log n}) \) time a bit array \( \text{occ}[h_1 - |s_1v|..h_1 - |s_1|] \) such that, for any \( i \), \( \text{occ}[i] = 1 \) iff \( t[i..n] \) has a prefix \( s_1w_2\overline{w}_3 \) such that \( h_2 - (i + |s_1|) = (i + |s_1w_2w|) - q \) and \( i + |s_1| < h_2 < i + |s_1w| \) (see Fig. 3).

Proof. We first test whether \( (h_2 + 1 + q - |s_2|)/2 \) is integer and \( D_2[(h_2 + 1 + q - |s_2|)/2] = 1 \) to check that \( s_2 \) occurs precisely between the substitutions of \( x \) and \( \overline{x} \) (see Fig. 3). Using the lcp data structure for the string \( t \), we check in \( O(1) \) time that \( t[h_2+1] = t[q-1], t[h_2+2] = t[q-2] \), ... and find the length \( b \) of the longest common prefix of \( t[h_1-1..h_1-1] \) and \( t[q+1..n] \).

Then, for each \( i \in [h_1 - |s_1| - \min\{|b,|v|\}..h_1 - |s_1|] \), we have \( \text{occ}[i] = 1 \) iff \( D_1[i] = 1 \) and \( D_2[q + h_2 - (i + |s_1|) + 1] = 1 \); for all other \( i \in [h_1 - |s_1v|..h_1 - |s_1|] \), we have \( \text{occ}[i] = 0 \). Thus, we obtain \( \text{occ} \) performing the bitwise “and” of the corresponding subarray of \( D_1 \) and the corresponding reversed subarray of \( D_3 \). The length of both these subarrays is bounded by \( |v| \). To obtain the reversed subarray efficiently, we utilize a precomputed table of size \( O(\sqrt{n}) \) that allows us to reverse the order of bits in one \( \Theta(\log n) \)-bit machine word in \( O(1) \) time. Thus, the running time of this algorithm is \( O(\frac{|v|}{\log n}) \).

![Figure 3](https://example.com/figure3.png)

**Figure 3** An instance of \( p = s_1x\overline{s2x}x_3 \) in the proof of Lemma 20.

Suppose that \( p \) starts with \( s_1x\overline{s2x} \) but is not equal to \( s_1x\overline{s2x}x_3 \) (the case when \( p \) starts with \( s_1\overline{x}2x \) is analogous). We are to compute all possible lengths of the substitutions of \( x \) from the range \( \left( \frac{3}{2}|v|[..2|v|] \right) \) that could produce the found occurrences of \( v \) and \( \overline{v} \) starting at positions \( q_1 \) and \( q_2 \). Since \( v \) is not periodic, there are at most four occurrences of \( v \) [or \( \overline{v} \)] starting in the segment \( [q_2 + |v_s3..q_2 + |v_{s3v}|] \). We find all these occurrences with the aid of Lemma 3 putting \( \lambda = |s_3| \). If \( x_3 = x \) [resp., \( x_3 = \overline{x} \)], then, given the starting position \( q_3 \) of an occurrence of \( v \) [resp., \( \overline{v} \)] such that \( q_3 \in [q_2 + |v_s3..q_2 + |v_{s3v}|] \), the number \( \beta = (q_3 - q_1 - |s_23|)/2 \) [resp., \( \beta = q_3 - q_2 - |s_3| \)] is equal to the length of the corresponding substitution of \( x \) that could produce the found occurrences of \( v \) and \( \overline{v} \) starting at \( q_1, q_2 \), and \( q_3 \). Thus, we obtain a constant number of possible lengths for substitutions of \( x \) and, for each of the found lengths, we apply Lemma 7.

Proof of Lemma 7 (continuation). For brevity, denote \( s(i) = h_2 - \beta - |s_1| + i + 1 \). Our aim is to “filter” the bit array \( D[0..|v|] \) so that, for any \( i \), it will be guaranteed that \( D[i] = 1 \) iff there is an instance \( s_1w_1w_2w_3 \cdot \cdot \cdot w_{r-1}w_r..s_r \) of \( p \) starting at \( s(i) \) and such that \( |w_1| = \cdot \cdot \cdot = |w_{r-1}| = \beta \) and \( s(i) + |s_1| \leq h_1 < h_2 < s(i) + |s_1| + \beta \). Without loss of generality,
suppose that \( p \) has a prefix \( s_1x \) (the case of the prefix \( s_1 \bar{x} \) is symmetrical). For each \( h \in (0..8] \), we “filter” the subarray \( D[(y - 1)\lceil \frac{q}{8} \rceil..y\lceil \frac{q}{8} \rceil] - 1 \) in \( O(r + \frac{\beta}{\log n}) \) time (assuming \( D[i] = 0 \) for \( i > \beta - |x| \) so that these subarrays are well defined); hence, the overall running time of our filtration algorithm is \( O(r + \frac{\beta}{\log n}) \).

Fix \( y \in (0..8] \). Suppose that there are two positions \( i, i' \in [(y - 1)\lceil \frac{q}{8} \rceil..y\lceil \frac{q}{8} \rceil] \) such that \( i < i' \) and there are two instances of \( p \) starting at positions \( s(i) \) and \( s(i') \), respectively, such that the lengths of the substitutions of \( x \) in both these instances are equal to \( \beta \); denote by \( w \) and \( w' \) the corresponding substitutions of \( x \) in these instances. Clearly \( D[i] = D[i'] = 1 \). It turns out that in this case \( w \) and \( w' \) both are periodic and, relying on this fact, we will deduce some regularities in the distribution of positions \( i'' \in [(y - 1)\lceil \frac{q}{8} \rceil..y\lceil \frac{q}{8} \rceil] \) such that \( t[s(i'')..s(i')] + |s_1s_2 \cdots s_r| + (r - 1)\beta - 1 \) is an instance of \( p \); so, it will suffice to assign \( D[i''] = 0 \) for all “non-regular” positions \( i'' \). Let us describe precisely the nature of these regularities.

Denote \( \gamma = i' - i \). Fix the minimal number \( z \in (1..r) \) such that \( x_z = \bar{x} \). Since \( \gamma \leq \frac{\beta}{8} \), it follows from Fig. 4 that \( w \) and \( w' \) both have period \( 2\gamma \leq \frac{\beta}{4} \).

Then, for any \( z' \in [1..r] \), the string \( t[s(y\lceil \frac{q}{8} \rceil) + |s_1s_2 \cdots s_r| + (z' - 1)\beta - s(y\lceil \frac{q}{8} \rceil) + |s_1s_2 \cdots s_r| + (z' - 1)\beta + \lceil \frac{q}{8} \rceil] \) is a substring of the substitutions of \( x_{z'} \) in the instances of \( p \) starting at \( s(i) \) and \( s(i') \). Therefore, this string has period \( 2\gamma \leq \frac{\beta}{2} \). Applying Lemma 3 for this string, we find in \( O(1) \) time a run \( t[i_z..j_{z'}] \) with the minimal period \( \gamma_z \leq 2\gamma \) that contains the substitutions of \( x_{z'} \) occurring at positions \( s(i) + |s_1s_2 \cdots s_r| + (z' - 1)\beta \) and \( s(i') + |s_1s_2 \cdots s_r| + (z' - 1)\beta \), respectively (note that \( d_{z'} \) may not equal \( 2\gamma \); the run \( t[i_z..j_{z'}] \) must exist because otherwise there cannot exist such positions \( i \) and \( i' \) corresponding to two instances of \( p \). The found runs \( t[i_z..j_{z'}] \), \( t[i_{z'}..j_{z'}] \), \( t[i_{z'}1..j_{z'}] \) are uniquely determined by \( y \in (0..8] \) and their choice does not depend on the choice of \( i \) or \( i' \). Moreover, since the choice of \( i \) and \( i' \) was arbitrary, it follows that if, for some \( i'' \in [(y - 1)\lceil \frac{q}{8} \rceil..y\lceil \frac{q}{8} \rceil] \), there is an instance of \( p \) starting at position \( s(i'') \) with substitutions of length \( \beta \), then, for any \( z' \in [1..r] \), the substitution of \( x_{z'} \) in this instance is a substring of \( t[i_z..j_{z'}] \).

We check in \( O(1) \) time whether \( d_1 = d_2 = \cdots = d_r \) (if not, then there cannot exist such positions \( i \) and \( i' \) corresponding to two instances of \( p \)). Denote by \( \ell \) and \( \ell' \), respectively, the starting position of a Lyndon root of \( t[i_z..j_{z'}] \) and the ending position of a reversed Lyndon root of \( t[i_{z'}..j_{z'}] \); \( \ell \) and \( \ell' \) can be computed in \( O(1) \) time by Lemma 3. Obviously, we necessarily have \( \ell + d - 1 = \ell' - d + 1 \). We check this condition using the lcp data structure on the string \( t \Gamma \). It follows from Lemma 3 that the distance between \( \ell \) and the starting position of \( w \) in \( t[i_z..j_{z'}] \) must be equal to the distance between \( \ell' \) and the ending position of \( w \) in \( t[i_{z'}..j_{z'}] \) modulo \( d \), i.e., \( s(i) + |s_1| - \ell \equiv \ell' - (s(i) + |s_1| - s_1s_2 \cdots s_r + z\beta - 1) \mod d \) (see Fig. 4). The latter is equivalent to the equality \( 2s(i) \equiv \ell + \ell' - |s_1| - s_1s_2 \cdots s_r - z\beta + 1 \mod d \). The right-hand side of this equality, denoted \( \eta \), can be calculated in \( O(r) \) time. Thus, since the choice of \( i \) and \( i' \) was arbitrary, we have \( 2s(i'') \equiv \eta \mod d \) for any \( i'' \in [(y - 1)\lceil \frac{q}{8} \rceil..y\lceil \frac{q}{8} \rceil] \) such that the string \( t[s(i'')..s(i')] + |s_1s_2 \cdots s_r| + (r - 1)\beta - 1 \) is an instance of \( p \).

It turns out that, in a sense, the converse is also true. Suppose that \( i \in [(y - 1)\lceil \frac{q}{8} \rceil..y\lceil \frac{q}{8} \rceil] \), \( D[i] = 1 \), \( 2s(i) \equiv \eta \mod d \), and, for each \( z' \in [1..r] \), the string \( w_{z'} = t[s(i') + |s_1s_2 \cdots s_r| + \lceil \frac{|s_1s_2 \cdots s_r| + (r - 1)\beta - 1}{d} \rceil] \)
$(z' - 1)\beta + |s_1s_2 \cdots s_{z'}| + z'\beta - 1)$, which is a “suspected” substitution of $x_{z'}$ in the string $t(l[i]..s(i)) + |s_1s_2 \cdots s_{r_1}| + (r - 1)\beta - 1)$, is contained in the run $t[i_{z'}..j_{z'}]$. Choose $z'$, $z'' \in [1, \ldots , n]$ such that $x_{z'} = x$ and $x_{z''} = \overline{x}$. If $z' = 1$ and $z'' = z$, then the equalities $2s(i) \equiv \eta \pmod{d}$ and $t[l[f \cdot d - 1] = t[l_0 \cdot d + 1] + d]$ imply that $w_{z'} = w_z$. It follows from the equality $D[i] = 1$ that, for any $z', z'' \in [1, \ldots , n]$, if $x_{z'} = x_{z''}$, then $w_{z'} = w_{z''}$. Therefore, the string $t(l[i]..s(i)) + |s_1s_2 \cdots s_{r_1}| + (r - 1)\beta - 1)$ is an instance of $p$.

It is easy to verify that $2s(i) \equiv \eta \pmod{d}$ iff either $s(i) \equiv \frac{y}{2} \pmod{d}$ or $s(i) \equiv \frac{y+d}{2} \pmod{d}$. So, using an appropriate bit mask and the bitwise “and” operation on $\Theta(\log n)$-bit machine words, we can assign $D[i'] = 0$ for all $i' \in [(y - 1)[\frac{\beta}{8}]..y[\frac{\beta}{8}]]$ such that $2s(i') \not\equiv \eta \pmod{d}$ in $O(\frac{n}{\log n})$ time. Then, according to the starting and ending positions of the runs $t[l[i_1..j_1]], \ldots , t[l[i_{r-1}..j_{r-1}]]$, we calculate in an obvious way in $O(r)$ time the exact subrange $[k_1..k_2] \subset [(y - 1)[\frac{\beta}{8}]..y[\frac{\beta}{8}]]$ such that, for any $i \in [k_1..k_2]$ and any $z' \in [1, \ldots , n]$, the string $t(l[i]..s(i)) + |s_1s_2 \cdots s_z| + (z' - 1)\beta + s(i) + |s_1s_2 \cdots s_{z'}| + z'\beta - 1$ (a “suspected” substitution of $x_{z'}$) is a substring of $t[l[i_{z'}..j_{z'}]]$. Finally, we fill the subarrays $D[(y - 1)[\frac{\beta}{8}]..k_1 - 1]$ and $D[k_2 + 1..y[\frac{\beta}{8}]]$ with zeros in $O(\frac{n}{\log n})$ time.

If something was wrong in the above scenario or simply the final array $D[(y - 1)[\frac{\beta}{8}]..y[\frac{\beta}{8}]]$ contains only zeros, then we still can have exactly one position $\mu$ (mod $d)$ in $O(\frac{n}{\log n})$ time. Then, according to the starting and ending positions of the runs $t[l[i_1..j_1]], \ldots , t[l[i_{r-1}..j_{r-1}]]$, we calculate in an obvious way in $O(r)$ time the exact subrange $[k_1..k_2] \subset [(y - 1)[\frac{\beta}{8}]..y[\frac{\beta}{8}]]$ such that $t[l[i]..s(i)] + |s_1s_2 \cdots s_z| + (r - 1)\beta - 1$ is an instance of $p$. The minimal period of the substitution of $x$ in such instance will necessarily be greater than $\frac{\beta}{4}$ (otherwise, we would find this instance by the above filtering algorithm). Note that such instance of $p$ should contain the substring $t[l[s(y[\frac{\beta}{8}]) + |s_1s_2 \cdots s_{r_1}| + (r - 1)\beta - 1]$ in the substitution of $x_1$. Thus, the substitution of $x_{z' - 1} = x$ (recall that $x_1 = x$ and $z$ is the minimal number such that $x_2 = \overline{x}$) in such instance should contain the substring $\mu = t[q..q] + [\frac{\beta}{4}] = s(y[\frac{\beta}{8}]) + |s_1s_2 \cdots s_{z' - 1}| + (z - 2)\beta$.

Suppose that the minimal period of $\mu$ is greater than $\frac{\beta}{4}$; since $\frac{\beta}{4} \leq \frac{\log n}{n}$, this condition can be checked in $O(1)$ time by Lemma 3. Since $\mu$ is supposed to be a substring of a substitution of $x_{z' - 1} = x$ in the required instance of $p$, the string $\mu$ must occur in the string $t[l[q + [\frac{\beta}{4}]] + |s_1..q + 2\beta + s_z|]$ as a substring of a substitution of $x_2 = \overline{x}$ (of course, if there exists such an instance of $p$). Since the minimal period of $\mu$ is greater than $\frac{\beta}{4}$, any two occurrence of $\mu$ cannot overlap on $|\mu| - \frac{\beta}{4} \geq \frac{\beta}{4}$ letters. Therefore, there are at most $2\beta/\beta = 8$ occurrences of $\mu$ in $t[q + [\frac{\beta}{4}]] + |s_1..q + 2\beta + s_z|$. We find all these occurrences of $\mu$ using a slightly modified algorithm from the proof of Lemma 4 (putting $\lambda = |s_z|$; it takes $O(r + \frac{|\mu|}{\log n})$ time plus $O(\frac{n\log n}{\log n})$ time, if $\frac{\log n}{\log n} \leq \frac{\log n}{\log n} \leq |\mu| \leq \log n$). Each found occurrence of $\mu$ specifies exactly one possible instance of $p$ with substitutions of length $\beta$ and the starting position of this instance can be easily calculated. To test whether the substitutions of the variables $x$ and $\overline{x}$ in this instance respect each other, we utilize the lcp data structure.

Finally, suppose that the minimal period of $\mu$ is less than or equal to $\frac{\beta}{4}$. Since $\frac{\beta}{4} \leq \frac{\log n}{n}$, by Lemma 3 we can find in $O(1)$ time a run $t[l[i''..j'']]$ containing $\mu$ and having the same minimal period as $\mu$. Denote $\mu_1 = t[l[i'' - 1]..s(y[\frac{\beta}{8}]) + |s_1s_2 \cdots s_{z' - 1}| + (z - 2)\beta + y[\frac{\beta}{8}]]$ and $\mu_2 = t[l[s(y[\frac{\beta}{8}]) + |s_1s_2 \cdots s_{z' - 1}| + (z - 2)\beta] + 1]$. Note that $\mu_1$ [resp., $\mu_2$] is the minimal extension of $\mu$ to the left [resp., right] that “breaks” the minimal period of $\mu$. It is well known that both $\mu_1$ and $\mu_2$ are not periodic. Since the minimal period of the substitution of $x$ in the instance of $p$ that we are searching for must be greater than $\frac{\beta}{4}$, this substitution must contain either $\mu_1$ or $\mu_2$. So, to find this instance, it suffices to execute the algorithm similar to that described above putting $\mu = \mu_1$ and $\mu = \mu_2$. □
Detecting Unary Patterns

To Section 2.2

Lemma 21 ([19, Lemma 14]). For any primitive string $w$, there exists at most one pair of palindromes $u$ and $v$ such that $v \neq \epsilon$ and $w = uv$.

Lemma 22. Suppose that two strings $w$ and $\tilde{w}$ of length $\geq 3d$ both lie in a run with the minimal period $d$. Then, there exists a unique pair of palindromes $u, v$ such that $|uv| = d$, $v \neq \epsilon$, and $\tilde{w}$ is a prefix of the infinite strings $(vu)^\omega$.

Proof. Since $\tilde{w}[1..d]$ and $w[1..d]$ are substrings of the same run with period $d$, we have $\tilde{w}[1..d] = vu$ and $w[1..d] = uv$ for some strings $u$ and $v$ such that $v \neq \epsilon$, i.e., $vu = \tilde{w}v = \tilde{w}u$. Hence, $u$ and $v$ are palindromes. By Lemma 21, this pair of palindromes is unique. ▶

Lemma 23. Let $t[i'..j']$ be a run with period $d$. If, for $h \in [i' + d..j'+1]$, $t[h-..h+1] = uv$ for some palindromes $u$ and $v$ such that $v \neq \epsilon$, then, for any $h' \in [i' + d..j'+1]$, we have $t[h'+d..h'-1] = u'v'$ for palindromes $u'$ and $v'$ such that $|u'| = (|u| - 2(h' - h)) \mod d$.

Proof. Let $h' = h + 1$ (the case $h' = h - 1$ is analogous). It suffices to prove that $t[h'+d..h'-1] = u'v'$ for palindromes $u'$ and $v'$ such that $|u'| = (|u| - 2) \mod d$. Since $t[h'+d..h'-1]$ is a suffix of $uv \cdot u[1]$ (or $v \cdot v[1]$ if $|u| = 0$), we have $t[h'+d..h'-1] = u'v'$, where $u' = u, v' = v$, if $d = 1$, and $u', v'$ are defined as follows, if $d > 1$: $u = u[1]\cdot u'[1]$ if $|u| > 1$, $u' = v$ if $|u| = 1$, $v = v[1]\cdot u'[1]$ if $|u| = 0$, $v' = u[1]\cdot v[1]$ if $|u| > 1$, $v' = u$ if $|u| = 1$, $v' = v[1]\cdot v[1]$ if $|u| = 0$. ▶

Lemma 24. Assuming $O(n)$ time preprocessing, one can find for any substring $t[i..j]$ in $O(1)$ time a pair of palindromes $u, v$ such that $t[i..j] = uv$ and $v \neq \epsilon$ or decide that there is no such pair.

Proof. In [11, Lemma C4] it was proved that, for any substring $t[i..j]$, if there exist palindromes $u$ and $v$ such that $t[i..j] = uv$, then there exist palindromes $u'$ and $v'$ such that $t[i..j] = u'v'$ and either $u'$ is the longest palindromic prefix of $t[i..j]$ or $v'$ is the longest palindromic suffix of $t[i..j]$. To test in $O(1)$ time whether a given substring is a palindrome, we can use the data preprocessed by Manacher’s algorithm [21]; so, it suffices to describe a data structure that allows to find the longest palindromic prefix/suffix of any substring in $O(1)$ time. Without loss of generality, we consider the case of palindromic suffixes.

Our main tool is the data structure called ctree [26]. The ctree of $t$ can be built in $O(n)$ time by [26, Proposition 11]. The main body of ctree of $t$ consists of nodes; any node $a$ represents a palindromic prefix of $pal[a]$ that is a substring of $t$ and, conversely, any palindromic prefix is represented by some node. Denote by link$[a]$ the node representing the longest proper palindromic suffix of $pal[a]$ (if any). In the proof of [26, Proposition 9], for each node $a$, there was defined a series link $\text{seriesLink}[a]$ such that $\text{seriesLink}[a]$ is either a node $a'$ representing the longest palindromic suffix of $pal[a]$ such that $|pal[a]| - |pal[link[a]]| \neq |pal[a']| - |pal[link[a']]|$ or the node representing the empty palindrome if there is no such $a'$.

In [26, 19] it was shown that the tree that is induced by the series links with the root in the node representing the empty palindrome has height at most $O(\log n)$. We build on this tree the weighted ancestor data structure from [17] that allows, for any given node $a$ and number $\gamma \geq 0$, to find the farthest ancestor $a'$ of $a$ such that $|pal[a']| \geq \gamma$. The structure of [17] supports these weighted ancestor queries in $O(T_h)$ time, where $h$ is the height of the tree and $T_h$ is the time required to answer a predecessor query on a set of $O(h)$ elements, so, using the fusion heaps [9], we can answer weighted ancestor queries on our tree in $O(1)$ time. Finally, as it was proved in [26], during the construction of ctree, we can create an array...
psuf[1..n] such that, for any $j \in [1..n]$, psuf[$j$] is the node of ctrep representing the longest palindromic suffix of $t[i..j]$ (see (20)).

Now, to find the longest palindromic suffix of a given substring $t[i..j]$, we compute the farthest ancestor $a$ of psuf[$j$] such that $|pal[a]| \geq j - i + 1$; then, by the definition of the series links, the longest palindromic suffix of $t[i..j]$ is either $pal[\text{seriesLink}[a]]$ or the palindromic suffix of $pal[a]$ with the length $|pal[a]| - c(|pal[a]| - |pal[\text{link}[a]]|)$, where $c$ is the minimal integer such that $|pal[a] - c(|pal[a]| - |pal[\text{link}[a]]|)) \leq j - i + 1$.

Proof of Lemma 10 (continuation). Now assume that $x_{z-1} \neq x'_{z-1}$. Suppose that $x_z = x_{z'}$. Let us find all runs with period $d$ that can contain the substitution $w_{z'}$. Using the run $t[i''..j'']$, one can find three possible choices for such run in the same way as we found three choices for $t[i''..j'']$ using $t[i'..j']$; we should process each of these three choices. Let us fix one such run $t[i''..j'']$. So, suppose that $t[i''..j'']$ contains $w_{z'}$. Denote $h_0 = h + |s_z| - 1$ and $h_0' = h' + |s_{z'}| - 1$. Since $t[h_0+1..h_0'] = w_bl_{z+1}w_{z+1}..w_{z-1}s_{z'}$, if the number $h_0' - h_0$ is known, we can calculate $|w_1| = (h_0' - h_0 - |s_{z+1}..s_{z'}|)/(z' - z)$ and apply Lemma 7 putting $\beta = |w_1|$. Let $t''$ be the starting position of a Lyndon root of $t[i''..j'']$; $t''$ can be found in $O(1)$ time by Lemma 6. Now we can find a constant number of possible values for the number $h_0' - h_0$ doing the same case analysis on the positions $h_0$ and $h_0'$ as we did on $h$ and $h'$ but using the runs $t[i''..j'']$ and $t[i''..j'']$ instead of $t[i'..j']$ and $t[i'..j']$, the positions $t'$ and $t''$ instead of $t$ and $t'$, and a reversed version of Lemma 6.

Finally, suppose that $x_{z-1} \neq x_{z-1}$ and $x_z \neq x_{z'}$. Without loss of generality, assume that $x_{z-1} = x$. Then, we have either $x_{z-1}s_zx_z = x_{z-1}s_{z'}x_{z'} = x_{z-1}s_{z'}x_z$ or $x_{z-1}s_{z-1}x_z = x_{z-1}s_{z-1}x_{z'} = x_{z-1}s_{z-1}x_{z'}$. Suppose that $x_{z-1}s_{z-1}x_z = x_{z-1}s_{z'}x_{z} = x_{z-1}s_{z-1}x_{z'}$. Since $t[h'-d..h'-1] = t[h'+|s_z|..h'+|s_{z'}|+d-1]$, it follows from Lemma 1 that $h' - 1 - t' \equiv n'_{0}' - (h' + |s_z|)$ (mod $d$), where $n'_{0}'$ is the ending position of a reversed Lyndon root of $t[i''..j'']$; $n_0'$ can be computed in $O(1)$ time by Lemma 6. Thus, we obtain $2h' \equiv t' + n'_{0}' + 1 - |s_0| \bmod d$, i.e., either $h' \equiv t' + n'_{0}' + 1 - |s_0| \bmod d$. It is easy to see that any segment of length $d$ contains at most two positions $h'$ satisfying the latter equalities; moreover, one can find these positions in $O(1)$ time. So, since, by Lemma 3, $h' = j' - \text{preq}(z_0) + 1$ or $h'$ lies in one of two segments of length $d$, there are at most five values for $h'$ such that $2h' \equiv t' + n'_{0}' + 1 - |s_0| \bmod d$; we can find them all in $O(1)$ time. Symmetrically, we find at most five possible values for $h$ but using the runs $t[i''..j'']$ and $t[i''..j'']$ instead of $t[i''..j'']$ and $t[i''..j'']$, and the position $t_0'$ instead of $n_0'$. Finally, for each found value $h' - h$, we calculate $\beta = (h' - h - |s_{z+1}..s_{z-1}|)/(z' - z)$ and apply Lemma 7.

It remains to process the case when $x_{z-1}s_{z+1}x_z = x_{z-1}s_{z+1}x_{z'} = x_{z-1}s_{z+1}x_{z'}$. Since in this case $w$ and $w'$ both are substrings of $t[i''..j'']$ as substrings of $x = x$ and $x_{z-1} = x$, by Lemma 22 there exist palindromes $u$ and $u'$ such that $|uw| = d$, $v \neq \epsilon$, and $w$ is a prefix of the infinite string $(vw)^\infty$. In $O(r)$ time we find a number $z'' \in (z..z')$ such that $x_{z''-1}s_{z''}x_{z''} = x_{z''-1}s_{z''}x_{z''}$. Since $w_{z''}w$ is a substring of $t[i''..j'']$, we have $s_{z''} = u(vw)^k$ for some $k > 0$. Therefore, we can compute the length of $u$: $|u| = |s_{z''}| \bmod d$. Given a segment $[f_1..f_2] \subset [i''+d,j'']+1$ of length $d$, it follows from Lemmas 21 and 23 that there exist at most two positions $f \in [f_1..f_2]$ such that $t[f-d..f-1] = u'v'u'$ for some palindromes $u'$ and $v'$ such that $|u'| = |u|$. We can find these positions in $O(1)$ time using the equality from Lemma 23 provided, for some $f' \in [f_1..f_2]$, we know palindromes $u'$ and $v'$ such that $t[j''-d..j''-1] = u''v''$. The palindromes $u''$ and $v''$ can be computed for arbitrary $j''$ in $O(1)$ time by Lemma 24. Since, by Lemma 9, $h$ either equals $j' - \text{preq}(s_z) + 1$ or lies in one of the segments $(j' + 1 - d, j' + 1)$ or $(j' - |s_z| - d, j' - |s_z|)$ of length $d$, there are at most five possible values for $h$ and each of them can be found in $O(1)$ time. In the same way we find at
most five possible values for $h'$. Finally, for each obtained possible value $h' - h$, we calculate $\beta = (h' - h - |s_2s_{l+1}\cdots s_{r-1}|)/(\zeta' - 2)$ and apply Lemma 7.

Proof of Lemma 13 (continuation). Suppose that $x_{r-1}s_{r-1}x_{r'} = \bar{x}s_{r}x$ and $x_{r''-1}s_{r''}x_{r''} = x_{s'}x$. Since $w$ and $\bar{w}$ both are substrings of $t[i..j']$ and $|w| \geq 3d$, it follows from Lemma 22 that there are palindromes $u$ and $v$ such that $|uv| = d$, $v \neq \epsilon$, and $w$ is a prefix of the infinite string $(vu)^\infty$. Since $\bar{w}s_{r}x$ is a substring of $t[i..j']$ and the strings $vu$ and $w$ are primitive, it follows from Lemma 1 that $s_{r''} = u(vu)^k$ for an integer $k'$ and hence $|v| = |s_{r''}| \mod d$, $|v| = d - |u|$. Similarly, since $\bar{w}s_{r}w$ is a substring of $t[i..j']$, we have $\bar{w}s_{r}w = (vu)^k$ for an integer $k'$ and therefore $2|w| \equiv |v| - |s_{r''}| \pmod{d}$. Thus, either $|w| \equiv \frac{|v| - |s_{r''}|}{2} \pmod{d}$ or $|w| \equiv \frac{d^2 - |s_{r''}|^2}{2} \pmod{d}$. Since $|v| = (\pm |s_{r''}|) \pmod{d}$, we obtain either $|w| \equiv \frac{|v| - |s_{r''}|}{2} \pmod{d}$ or $|w| \equiv \frac{d^2 - |s_{r''}|^2}{2} \pmod{d}$.

The continuation of the discussion about in-a-run instances of $p$. For a given segment $[b_1..b_2] \subset [i..j]$ of length $d$, let us consider how to find all instances $t[i..j]$ of $p$ satisfying (3) and such that $j - |s_p| + 1 \in [b_1..b_2]$ when $p$ contains both variables $x$ and $\bar{x}$.

Proof of Lemma 16 (continuation). While in the case $x_1 = \cdots = x_{r-1}$ it was sufficient to rely on the periodicity of $t[i..j']$ to test whether corresponding substitutions are equal (as in Lemma 15), in the case when $p$ contains both $x$ and $\bar{x}$ it is not clear how to test for all $h \in [b_1..b_2]$ simultaneously whether corresponding substitutions of $x$ and $\bar{x}$ respect each other. However, it turns out that there are at most two positions $h \in [b_1..b_2]$ for which there might exist a string $t[i..j]$ satisfying (3) and such that $j - |s_p| + 1 = h$. We find these two positions in $O(1)$ time and process each of them separately in $O(r)$ time.

Let $t[i..j]$ be a string satisfying (3) and such that $j - |s_p| + 1 \in [b_1..b_2]$. Denote by $w$ the substitution of $x$ in $t[i..j]$. By Lemma 22 there exist palindromes $u$ and $v$ such that $v \neq \epsilon$ and $\bar{w}[1..d] = vu$. Let us find the lengths of $u$ and $v$. Choose a number $z' \in (1..r)$ such that $x_{r-1}s_{r-1}x_{r'} = \bar{x}s_{r}x$ (it exists because $x_{r-1} = x$). Since $p \neq s_1x_2s_3\bar{s}_3s_4$, and $r \geq 3$, there is $z'' \in (1..r)$ such that $x_{r-1}s_{r-1}x_{r''} = x_{s'}x_{r'}$ is equal to either $x_{s'}\bar{x}$ or one of the strings $x_{r''}x_{r}$ or $\bar{x}s_{r}x_{r'}$. Suppose that $x_{r''-1}s_{r''}x_{r''} = x_{s'}x_{r'}$. Since the strings $vu$ and $w$ are primitive, it follows from Lemma 1 that $s_{r''} = u(vu)^k$ for an integer $k'$. Therefore, we can compute the length of $w$: $|w| = |s_{r''}| \pmod{d}$. Now suppose that $x_{r''-1}s_{r''}x_{r''} = x_{s'}x_{r'}$ [resp. $x_{r''-1}s_{r''}x_{r''} = \bar{x}s_{r}x_{r'}$]. It follows from Lemma 1 that the distance between any two occurrence of $w$ [resp. $\bar{w}$] in $t[i..j']$ is a multiple of $d$; thus, we have $|w| \equiv -|s_{r''}| \pmod{d}$. Since $\bar{w}s_{r}w$ is a substring of $t[i..j']$, by Lemma 1 we have $\bar{w}s_{r}w = (vu)^k$ for an integer $k'$. Therefore, we can compute the length of $v$: $|v| = |s_{r''} + 2|w| \pmod{d} = (|s_{r''} - 2|s_{r''}|) \pmod{d}$ assuming $|v| = d$ if $|s_{r''} - 2|s_{r''}| \equiv 0 \pmod{d}$.

Using Lemma 21 we find in $O(1)$ time palindromes $u''$ and $v''$ such that $t[b_2-d+1..b_2] = u''v''$ (they exist by Lemma 23). Since $b_2 - 1 \equiv 0 \pmod{d}$, it follows from Lemmas 21 and 23 that there are at most two positions $h \in [b_1..b_2]$ such that $t[h-d..h-1] = u''v''$ for some palindromes $u''$ and $v''$ satisfying $|u''| = |u|$ and $|v''| = |v|$; these positions $h$ can be found using the equality from Lemma 23 and the lengths $|u''|$ and $|v''|$. Fix one such position $h \in [b_1..b_2]$.

By Lemma 14 there exists an instance $t[i..j] = s_1w_1s_2w_2\cdots w_{r-1}s_{r-1}s_r$ of $p$ satisfying (3) and such that $i \geq i'$, $|w_1| \equiv \delta \pmod{d}$, and $j - |s_p| + 1 = h$ if the string $t[i..j]$ is an instance of $p$ (recall that $i_h = h - |s_1s_2\cdots s_{r-1}| - (r - 1)\delta$, $j_h = h + |s_r| - 1$, and $c_h = \frac{i_h - j_h}{r - 1}$); moreover, in this case $t[i..j] = t[i_h - (r - 1)(\delta + cd)..j_h]$ for some $c \in [0..c_h]$. So, we test whether $t[i..j]$ is an instance of $p$ in $O(r)$ time using the arrays $D_z$ for $z = 1$ and the lcp data structure of the string $t$. So, if $t[i_h..j]$ is an instance of $p$, then we put $E[h] = 1$; if $h$ is the smallest of the two positions such that $t[h-d..h-1] = u''v''$ for some palindromes $u''$ and $v''$.
such that \(|u'| = |u|\) and \(|v'| = |v|\), then we put \(h' = h\) and \(d' = c_h\); otherwise, we put \(d'' = c_h\). (So, \(E[i..b_2]\) contains at most two non-zero positions).

To find all instances \(t[i..j]\) of \(p\) such that \(j - |s_i| + 1 = h\) and \(i < i'\), we use a case analysis relying on a symmetric version of Lemma 8 similar to the analysis described above for the case \(x_1 = \cdots = x_{r-1} = 0\). By Lemma 8 we have either \(i + |s_i| = i' + |\text{suffix}(s_1)|\) or \(i + |s_i| \in [i', i' + d]\). First, suppose that \(i + |s_i| \in [i', i' + d]\). Since we must have \(h - (i + |s_i|) \equiv |s_2 s_3 \cdots s_{r-1}| + (r - 1)\delta \pmod{d}\), we find in \(O(1)\) time at most one possible position \(h' \in [i', i' + d]\) such that \(h' - h' \equiv |s_2 s_3 \cdots s_{r-1}| + (r - 1)\delta \pmod{d}\) (we suspect that \(h' = i + |s_i|\)) and test whether \(t[h' - |s_1|..h + |s_i| - 1]\) is an instance of \(p\) in \(O(r)\) time with the aid of the arrays \(\{D_z\}_{z=1}^r\) and the lcp data structure of \(t[i']\); if this string is an instance, then we set \(F[h] = 1\), \(a' = (h - h' - |s_2 s_3 \cdots s_{r-1}| - (r - 1)\delta)/d\), \(h'' = b_2\). (So, \(F\) trivially encodes at most one instance of \(p\)) Finally, we test in \(O(r)\) time whether \(t[i' + |\text{suffix}(s_1)|] - |s_1|..h + |s_i| - 1\) is an instance of \(p\); thus, we can find an additional instance of \(p\) that is not encoded in \(E\) or \(F\).

Now consider the special case \(p = s_1 x s_2 x s_3\). Let us count all instances \(t[i..j]\) = \(s_1 \tilde{w}s_2 w s_3\) of \(p\) satisfying (7) and such that \(i + |s_1| \in [b_1, b_2]\). Suppose that \(t[i..j]\) = \(s_1 \tilde{w}s_2 w s_3\) is such instance. By Lemma 22 there exist palindromes \(u\) and \(v\) such that \(u \neq \epsilon\) and \(w[r..d] = uv\). It follows from Lemma 14 that \(s_2 = (uv)'\tilde{v}\) for some integer \(k' \geq 0\). Hence, we can calculate the length of \(v: |v| = |s_2| \pmod{d}\) assuming \(|v| = d\) if \(|s_2| \pmod{d} = 0\). Using Lemma 24 we find in \(O(1)\) time palindromes \(u''\) and \(v''\) such that \(t[b_2 - d + 1..b_2] = u''v''\) (they exist by Lemma 23). Since \(b_2 - b_1 + 1 = d\), it follows from Lemmas 21 and 23 that there are at most two positions \(h \in [b_1, b_2]\) such that \(t[h - d..h - 1] = v'u'\) for some palindromes \(u'\) and \(v'\) satisfying \(|u'| = |u|\) and \(|v'| = |v|\); these positions \(h\) can be easily found using the equality from Lemma 23 and the lengths \(|u'\) and \(|v'|\).

So, fix one such position \(h \in [b_1, b_2]\). The position \(h\) is a suspected starting position of \(s_2\) in an instance of \(p\). By the procedure similar to that used in the proof of Lemma 20 we compute in \(O(1 + \frac{d}{\text{length}})\) time a bit array \(D'[h - |s_1| - d..h - |s_i|]\) such that \(D'[h'] = 1\) iff there is an instance \(s_1 \tilde{w}s_2 w s_3\) of \(p\) starting at position \(h'\) and such that \(h' + |s_1| \in h\). It follows from Lemma 14 that any string \(t[i..i + |s_1| + s_2 + 2y]\) such that \(y \geq d, i' \leq i \in |s_1| + s_2 + 2y \leq j', i + |s_1| + s_2 + 2y \leq h\), and \(i + |s_1| + s_2 + 2y \leq h\) is an instance of \(p\) iff the string \(t[i + \frac{d}{2}]..j - \frac{d}{2}\) is an instance of \(p\), i.e., iff \(D'[i + \frac{d}{2}] = 1\). So, once the array \(D'\) is computed, one can easily count the number of all instances \(t[i..i + |s_1| + s_2 + 2y]\) of \(p\) such that \(y \geq d, i' \leq i \in |s_1| + s_2 + 2y \leq j', i + |s_1| + s_2 + 2y \leq h\). It remains to count the number of all instances \(t[i..i + |s_1| + s_2 + 2y]\) of \(p\) such that \(y \geq d, i' \leq i \in |s_1| + s_2 + 2y \leq h\), and \(i + |s_1| + s_2 + 2y \leq h\), and \(i + |s_1| + s_2 + 2y \leq h\). We can do this with the same case analysis as for the case \(p \neq s_1 x s_2 x s_3\).

The cases \(p = s_1 x s_2 \bar{x} s_3\) and \(p = s_1 \bar{x} s_2 x s_3\). Suppose that \(p = s_1 x s_2 \bar{x} s_3\) (the case \(p = s_1 x s_2 x \bar{s}_3\) is symmetrical). Consider an instance \(t[i..j]\) = \(s_1 \tilde{w}s_2 w s_3\) of \(p\) whose substitutions of \(x\) and \(\bar{x}\) have length \(\geq 3d\) and lie in distinct runs \(t[i'..j']\) and \((t[i'..j']')\) with period \(d\). As above, by Lemma 8 we have \(h \in (j' + d - j + 1)\) or \(h \in (j' - |s_{z+1} - d..j' - |s_{z+1}|)\) or \(h = j' - |\text{prefix}(s_{z+1})| + 1\), where \(h = i + |s_1| \in h\). Suppose that \(h \in (j' + d - j + 1)\) (other cases are analogous) and denote \(b_1 = j' + 2 - d\) and \(b_2 = j' + 1\).

Denote by \(\ell\) and \(\ell_0\) the starting position of a Lyndon root of \(t[i'..j']\) and the ending position of a reversed Lyndon root of \(t[i'..j']\), respectively; \(\ell\) and \(\ell_0\) can be found in \(O(1)\) time by Lemma 6. In order to synchronize parts of \(p\) that are contained in \(t[i'..j']\) and \((t[i'..j']')\), we check in \(O(1)\) time using the lcp data structure whether \(t[\ell..\ell + d - 1] = t[\ell_0 - d + 1..\ell_0]\); if not, then there cannot be any instances of \(p\) such as \(t[i..j]\). It follows from Lemma 1 that \(h - \ell \equiv \ell_0 - (h + |s_2| - 1) \pmod{d}\). Hence, we obtain \(2h \equiv \ell + \ell_0 - |s_2| + 1 \pmod{d}\). Since
Detecting Unary Patterns

\[ b_2 - b_1 + 1 = d, \]
we can find in \( O(1) \) time at most two positions \( h \) in \([b_1..b_2]\) satisfying the latter equality. Fix one such position \( h_0 \in [b_1..b_2] \); \( h_0 \) is a suspected starting position of \( s_2 \) in an instance of \( p \).

Applying Lemma 20 with \( h_1 = h_0 - d, b_2 = h_0 - 1, q = h_0 + |s_2| \) (see Fig. 3), we compute a bit array \( \text{occ}[h_1 - d - |s_1|..h_1 - |s_1|] \) such that, for any \( h' \in [h_1 - d - |s_1|..h_1 - |s_1|] \), we have \( \text{occ}[h'] = 1 \) if \( t[h'.n] \) has a prefix \( s_1 \overset{w}{\rightarrow} s_2 \overset{s_3}{\rightarrow} \) such that \( h' + |s_1| \overset{w}{\rightarrow} h_0 \). Since \( t[i..\ell+d-1] = t[i..d+1..\ell_0'] \), by the definition of \( h_0 \), it follows that any string \( t[i..j] \) such that \( i' \leq i, j \leq j'' \), and \( i + |s_1| + \ell_w = h_0 \), where \( \ell_w = (j - i + 1 - |s_1|s_2s_3)/2 \), is an instance of \( p \) iff \( \text{occ}[h_0 - d - (\ell_w \mod d) - |s_1|] = 1 \). So, in this way we found all instances of \( p \) that correspond to \( h_0 \) and do not cross the boundaries \( i' \) and \( j'' \).

Now suppose that \( t[i..j] = s_1 \overset{w}{\rightarrow} s_2 \overset{s_3}{\rightarrow} \) is an instance of \( p \) such that \( i < i' \leq i + |s_1| \) and \( i + |s_1| \overset{w}{\rightarrow} h_0 \) (the case \( j > j'' \) is symmetrical). By Lemma 8, we have either \( i + |s_1| \in [i'..i'+d] \) or \( i + |s_1| = i' + |s_\text{uf}(s_1)| \), which, in the first case, we can check whether \( t[i'+|s_\text{uf}(s_1)|] - |s_1|..h_0 + |s_2| + (h_0 - i' - |s_\text{uf}(s_1)|) + |s_1| - 1 \) is an instance of \( p \) in \( O(1) \) time using the lcp data structure and the arrays \( D_1, D_2, D_3 \). Secondly, we find all instances \( t[i..j] = s_1 \overset{w}{\rightarrow} s_2 \overset{s_3}{\rightarrow} \) of \( p \) satisfying \( i + |s_1| \overset{w}{\rightarrow} h_0 \) and \( i + |s_1| \in [i'..i'+d] \) using Lemma 20 with \( h_1 = i' + d, h_2 = i' + 2d - 1 \), \( q = h_0 + |s_2| + (h_0 - i' - 2d) \) (see Fig. 3).

To Subsection 3

Proof of Theorem 18 The present format of encoding can be produced from the results of Lemmas 8, 10, 17 as follows. First, the resulting arrays \( \text{occ} \) from Lemma 7 can obviously be transformed into arrays \( F_1 \). Secondly, the arrays \( F \) from Lemmas 10 and 17 can be transformed into two or three arrays \( F_h \) according to the separating thresholds \( h^0 \) and \( b_0^0 \) from these lemmas. Finally, each individual instance of \( p \) found in Lemmas 10 and 17 (i.e., each instance that is not encoded in the arrays \( F \) and \( F \) from these lemmas) can be encoded by one array \( F_h \) of length 1. Consider now arrays \( F \) produced by Lemmas 10 and 17.

Suppose that we processed by Lemma 17 numbers \( d, \delta \geq d, d \in [1..r] \) and two runs \( t[i'..j'] \) and \( t[i''..j''] \) whose minimal period is \( d \) and \( d \leq \frac{\ell - i'+1}{3} \); thus, we obtained a bit array \( E[b_1..b_2] \) with corresponding numbers \( h', h_0, d', d'', d''' \). Note that \( d', d'', d''' \leq \frac{\ell - i'+1}{d} \). We encode all instances of \( p \) encoded in the array \( E \) in \( O(\frac{\ell - i'+1}{d}) \) arrays \( F_h \) each having length \( O(d) \) as follows: we build for each \( c \in [0..d'] \) (resp., \( c' \in [0..d''] \), \( c'' \in [0..d'''] \)) a tuple \((\beta_h, F_h)\) such that \( \beta_h = \delta + cd \) (resp., \( \beta_h = \delta + cd' \), \( \beta = \delta + cd'' \)) and \( F_h[b_1 - |s_1s_2s_3| - z\beta_h..h' - |s_1s_2s_3| - z\beta_h] = E[b_1..h'] \) (resp., \( F_h[h' + 1 - |s_1s_2s_3| - z\beta_h..h_0 - |s_1s_2s_3| - z\beta_h] = E[h'+1..b_0] \)).

Suppose now that we processed by Lemma 16 numbers \( d, \delta \geq d, \delta \leq |s_1| + 1 \) of length \( d \) and a segment \([b_1..b_2] \subset [i'..j']+1 \) of length \( d \), and a run \( t[i'..j'] \) whose minimal period is \( d \) and \( d \leq \frac{\ell - i'+1}{3} \); thus, we obtained a bit array \( E[b_1..b_2] \) with corresponding numbers \( h', d', d'' \). Note that \( d', d'', d''' \leq \frac{\ell - i'+1}{d} \). If \( b_2 + |s_1| - 1 > j' \) (i.e., instances \( t[i..s_1s_2s_3s_{r-1}] - (r-1)\delta, h, |s_1| - 1 \) of \( p \), \( h \in [b_1..b_2] \), that are encoded in the array \( E[b_1..b_2] \) are not guaranteed to be substrings of \( t[i'..j'] \)), then we encode \( E[b_1..b_2] \) as above in tuples \((\beta_h, F_h)\) such that \( \beta_h = \delta + cd \), for \( c \in [0..d'] \) (resp., \( c \in [0..d''] \), \( c'' \in [0..d'''] \)) and \( F_h[b_1 - |s_1s_2s_3s_{r-1}| - (r-1)\beta_h..h' - |s_1s_2s_3s_{r-1}| - (r-1)\beta_h] = E[b_1..h'] \) (resp., \( F_h[h' + 1 - |s_1s_2s_3s_{r-1}| - (r-1)\beta_h..b_2 - |s_1s_2s_3s_{r-1}| - (r-1)\beta_h] = E[h'+1..b_2] \)). We combine all other arrays \( E \) obtained by Lemma 16 for the fixed run \( t[i'..j'] \), the fixed number \( \delta \), and all segments \([b_1..b_2] \subset [i'..j']+1 \) such that \( b_2 + |s_1| - 1 \leq j' \) in one tuple \((\delta_h, d_h, t[i'..j'], E_h[b_h..b_2] - d_h-1 \)) where \( \delta_h = \delta, d_h = d_t[i'..j'], t[i'..j'] \), and, for \( i \in [i_h..i_h + d_h] \), \( E_h[i] = E[i + ad] \), where \( E[b_1..b_2] \) is any of such arrays \( E \) and \( a \) is an integer such that \( i + ad \in [b_1..b_2] \) (the correctness of this encoding follows from the periodicity of
with any shift equal to a multiple of the period \( p \) by Lemma 2, basic instances of \( w \) that always start to the right of the starting position of any other instance of \( w \) is one of the first instances of \( w \) from \( \mathcal{R} \). Denote \( \tau = |\mathcal{R}| \). Let \( \rho \) denote the sum of the numbers \( \frac{\ell - j - 1}{d_{\ell,j}} \) over all runs \( t[\ell'..j'] \) from \( \mathcal{R} \). In [10] it was proved that \( \tau = O(n) \) and \( \rho = O(n) \). Since Lemma 7 is called at most \( O(n) \) times and Lemma 17 is called at most \( O(1) \) times for each run \( t[\ell'..j'] \) from \( \mathcal{R} \), it follows from our algorithm and from the above discussion that the number \( |\mathcal{H}| \) is asymptotically bounded by \( O(n + \rho) \) plus the number of tuples \((\beta_h, F_h)\) constructed from the arrays \( E[b_1..b_2] \) produced by Lemma 16 and such that \( b_2 + |s_r| - 1 > j' \). By Lemma 8 for any fixed \( \delta \) and run \( t[\ell'..j'] \) from \( \mathcal{R} \), our algorithm produces with Lemma 16 at most three non-empty bit arrays \( E[b_1..b_2] \) such that \( b_2 + |s_r| - 1 > j' \). Hence, since, for any fixed run \( t[\ell'..j'] \), Lemma 16 is called only for a constant number of distinct numbers \( \delta \), we obtain \( |\mathcal{H}| = O(n + \rho) = O(n) \).

Since, for any fixed run \( t[\ell'..j'] \), Lemma 16 is called only for a constant number of distinct numbers \( \delta \), we obtain \( |\mathcal{H}| = O(\tau) = O(n) \) and, by Lemma 2, \( \sum_{h \in \mathcal{H}} (j_h - i_h + 1) = O(n \log n) \). Finally, the number \( \sum_{h \in \mathcal{H}} (b_h - a_h + 1) \) is asymptotically bounded by the sum of the lengths of all runs from \( \mathcal{R} \) and the sum of the lengths of all substrings \( v \) such that \( |v| = |\alpha^h| \) and \( v \) occurs at position \( i|v| + 1 \) or \( i|v| + \lfloor \frac{|v|}{|\alpha^h|} \rfloor \), where \( k \in [0..\log_\alpha n] \) and \( i \in [0..\frac{n}{|\alpha^h|}] \); hence, by Lemma 2, \( \sum_{h \in \mathcal{H}} (b_h - a_h + 1) = O(n \log n) \).

\[ \text{Proof of Theorem 19 (continuation).} \] We first describe the technical details of the approach described in the main part of the paper.

Let us first consider a fixed \( j \in [1..k] \), and produce the instances of \( p_j \) for \( t \) using the algorithm summed up in Theorem 18. By this theorem, the instances of \( p_j \) can be separated into two types; more precisely, we will deal separately with the instances of \( p_j \) stored as tuples \((\beta_h, F_h[a_h..b_h])\) with \( h \in \mathcal{H} \) and, respectively, with those stored as tuples \((\delta_h, d_h, t[i_h..j_h], E_h[i_h..i_h + d_h - 1])\) with \( h' \in \mathcal{H} \). Essentially, after an \( O(\sum_{h \in \mathcal{H}} (b_h - a_h + 1)) = O(n \log n) \) time preprocessing, the representation of the instances of \( p_j \) stored as tuples \((\beta_h, F_h[a_h..b_h])\) can be easily modified so that each instance of the pattern \( p_j \) where \( x \) is replaced by a string \( w \) is encoded as the triple \((|w|, \ell, i)\), where \( \ell \) is the position in the suffix array of \( t \) where a suffix starting with \( w \) occurs and \( i \) is the position where the pattern occurs. We sort all these instances according to \( w \), and in case of equality according to their starting position \( i \) (this can be done by radix-sorting the triples above); the total time needed to do this sorting is clearly linear with respect to the number of instances, i.e., \( O(n \log n) \). Now, we have, for all substitutions of \( x \) by \( w \), all the instances of \( p_j \) with the same \( w \), that can be represented as tuples \((\beta_h, F_h[a_h..b_h])\) with \( h \in \mathcal{H} \), grouped together and sorted according their starting position.

When dealing with instances of \( p_j \) represented as tuples \((\delta_h, d_h, t[i_h..j_h], E_h[i_h..i_h + d_h - 1])\), then, after an \( O(\sum_{h \in \mathcal{H}} d_h [i_h = i_h + 1]) = O(n \log n) \) time preprocessing, we can get a new encoding of all the basic instances of \( p_j \) starting within the run \( r_h = t[i_h..j_h] \), as triples \((|w|, \ell, i, r_h)\) where \( w \) is the string substituting \( x \), \( \ell \) is a position of the suffix array where a suffix starting with \( w \) occurs, and \( i \) is the starting position of the instances of \( p_j \) such that \( i \) is one of the first \( d_h \) positions of \( r \). Each such basic instance of \( p_j \) may be shifted inside \( r_h \) with any shift equal to a multiple of the period \( d_h \) of the run. Note that the number of the basic instances of \( p_j \) (counted for all values of \( w \) and for all runs) that we need to store is, by Lemma 2 \( O(n \log n) \), as it is upper bounded by the sum of the lengths of all runs. Again, we radix sort these instances according to the value of \( w \) and then according to their starting position \( i \). It is also immediate that the leftmost instance of \( p_j \), where \( x \) is replaced by \( w \), contained in a run \( r_1 \) always starts to the right of the starting position of any other instance of \( p_j \), with \( w \) replacing \( x \), contained in a run occurring to the left of \( r_1 \).
The total time to produce these sorted lists is clearly $O(n \log n)$.

Once the lists produced, we fix a $w$ and select the list storing the instances of the pattern $p_j$ for that respective $w$, for all $1 \leq j \leq k$. Exactly one of these lists is non-empty, as the instances of $p_j$ for the respective substitution $w$ may be represented either as tuples $(\beta_h, F_h[a_h..b_h])$ with $h \in \mathcal{H}$ (so put in the list of triples $(|w|, \ell, i)$) or, respectively, as tuples $(\delta_h, d_h, t[i_h..j_h], E_h[i_h..i_h + d_h - 1])$ with $h' \in \mathcal{H}$ (so put in the list of tuples $(|w|, \ell, i, r_h)$).

So, in our algorithm we start with the lists for $p_1$ and retrieve from them the first instance of this sub-pattern. This instance is easily obtained by selecting the first element of the list of instances of $p_1$. Then we compute the ending position of this instance, denoted by $i_1$ (which can be done in $O(1)$ time) and move to the instances of $p_2$. Let initially $\ell = 1$ and we look now at the list corresponding to the instances of $p_{\ell+1}$ where $x$ is replaced by $w$.

If this list is the one of triples $(|w|, \ell, i)$, we just discard from it those instances that start before $i_{\ell+1}$ (the list is ordered by the starting positions of the instances, so we just have to go through it and discard the elements that we meet and start too much to the left); then we select its first element, and compute its ending position $i_{\ell+1}$, increase $\ell \leftarrow \ell + 1$ and restart the same procedure.

If this list is the one of tuples $(|w|, \ell, i, r_h)$, we discard from it the basic instances that start to the left of $i_{\ell+1}$ and which cannot be shifted by a multiple of $d_h$ (the period of the run $r_h$) in order to obtain an instance of $p_{\ell+1}$ that starts on a position $\geq i_{\ell+1}$. As soon as we meet a basic instance that may lead to an instance of $p_{\ell+1}$ that starts on a position $\geq i_{\ell+1}$ we stop discarding elements from the list. Now, in our list of instances of $p_{\ell+1}$ we have as first element the leftmost instance of $p_{\ell+1}$ that starts to the right of $i_{\ell} + 1$. We select it, compute its ending position $i_{\ell+1}$, increase $\ell \leftarrow \ell + 1$ and restart the same procedure.

The time used in this analysis is proportional to the number of discarded instances. In total, once we considered all $\ell$ from 1 to $k$, we use time proportional with the number of instances of all sub-patterns for a fixed $w$. So, when we consider all possible images for $x$, the total time we use in this procedure will equal the total time of instances of all sub-patterns. Which, as we explained, is $O(\sum_{j=1}^{k} n \log n) = O(kn \log n) = O(rn \log n)$.

\[\square\]