Repetitions in two-pattern strings

František Franěk Weilin Lu W. F. Smyth

Algorithms Research Group
Department of Computing & Software
McMaster University
Hamilton, Ontario
Canada L8S 4L7

August 2, 2000

Abstract

A recent paper shows that it can be determined in $O(n)$ time whether or not a given string $\boldsymbol{x}$ of length n is a substring of an infinite Sturmian string; further, if $\boldsymbol{x}$ is such a substring, that the repetitions in $\boldsymbol{x}$ can be computed in $\Theta(n)$ time, generalizing a similar result for Fibonacci strings. In her M.Sc. thesis W. Lu extended these results to "two-pattern" strings formed recursively from concatenations of strings $\boldsymbol{p}^i\boldsymbol{q}$ and $\boldsymbol{p}^j\boldsymbol{q}$, where $\boldsymbol{p}$ and $\boldsymbol{q}$ are so-called "suitable patterns". Sturmian strings thus constitute the special case when $\boldsymbol{p} = a$, $\boldsymbol{q} = b$, $|j-i| = 1$, while Fibonacci strings constitute the special case of Sturmian strings when $i = 1$. In this paper we significantly relax the conditions for "suitable patterns" while showing that the repetitions can still be determined in $\Theta(n)$ time, thus significantly extending the class of strings whose repetitions can be calculated in linear time. This result is a part of an ongoing two-pronged research effort to identify and describe the class of strings whose repetitions can be determined in linear time and to show (or refute) that, in general, repetitions in any string can be listed in a list of a linear length using a succinct notation of "runs" even though the list itself cannot be calculated in a linear time.

1 INTRODUCTION

It was shown in [IMS97] that repetitions in Fibonacci strings can be calculated in linear time. This result was generalized to Sturmian strings, of which Fibonacci strings form a proper subclass, in [FKS00]. In her M.Sc. thesis [McMaster University, 2000] W. Lu generalized the result to so-called "two-pattern" strings, i.e. strings recursively formed by blocks made of two "suitable patterns". Her class of two-pattern strings properly extended the class of block-complete Sturmian strings still allowing for a linear computation of repetitions. In this work we significantly relax the conditions on "suitable patterns", thus properly enlarging the class of "two-pattern" strings.

Given a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ of length n, we show that $O(n)$ time is required to decide whether or not a given string $\boldsymbol{x}$ is a two-pattern string. Further, in the case that $\boldsymbol{x}$ is indeed two-pattern, we show that all the repetitions in $\boldsymbol{x}$ can be computed in $\Theta(n)$ time.

As in [FKS00] we adopt an algorithmic, rather than a mathematical, point of view: we imagine always that the string $\boldsymbol{x}$ is unknown, and that our objective is to design efficient algorithms to process it, rather than study its mathematical properties.

We begin by defining "suitable patterns":

Definition 1 *A binary string* $\boldsymbol{q}$ *is said to be* $\boldsymbol{p}$**-regular**, $\boldsymbol{p}$ *a binary string, if there are binary strings* $\boldsymbol{u} \neq \varepsilon$ *and* $\boldsymbol{v}$ *and integers* $l \geq 0$, $n, m > 1$ *so that* $\boldsymbol{q} = (\ \boldsymbol{u}\boldsymbol{p}^l\boldsymbol{v}\boldsymbol{p}^n)^m\boldsymbol{u}$ *and if* $l = 0$, *then* $\boldsymbol{v} = \varepsilon$.

Definition 2 *An (ordered) pair of binary strings* $\boldsymbol{p}, \boldsymbol{q}$ *is said to be a* **suitable pair of patterns** *if*

1. $\boldsymbol{p}$ *is primitive, i.e. has no non-trivial borders,*

2. $\boldsymbol{p}$ *is neither a prefix nor a suffix of* $\boldsymbol{q}$,

3. $\boldsymbol{q}$ *is neither a prefix nor a suffix of* $\boldsymbol{p}$,

4. $\boldsymbol{q}$ *is not* $\boldsymbol{p}$*-regular.*

Note: this is a significant relaxation of conditions as used in W. Lu's thesis where it was required among other requirements that $\boldsymbol{p}$ is not a substring of $\boldsymbol{q}$ and that $\boldsymbol{p}$ and $\boldsymbol{q}$ have no common suffix. As we shall see later, most of combinatorial difficulties we had to overcome in this paper stem from this relaxation.

Definition 3 *A* **morphism** *is a mapping* $\sigma : a \to \boldsymbol{p}^i\boldsymbol{q}, b \to \boldsymbol{p}^j\boldsymbol{q},\ 0 \le i < j$, $\boldsymbol{p}$ *and* $\boldsymbol{q}$ *a suitable pair of patterns.*

We call σ an ***expansion*** and observe that we may extend it naturally to arbitrary strings $\boldsymbol{x} = \boldsymbol{uv}$ of $\{a,b\}^+$ by defining

$$\sigma(\boldsymbol{uv}) = \sigma(\boldsymbol{u})\sigma(\boldsymbol{v}).$$

Definition 4 *Let* $\Sigma = \{\sigma_1, \sigma_2, \ldots \sigma_n\}$ *denote an* **expansion sequence**, *where the choices of* $\boldsymbol{p}$, $\boldsymbol{q}$, i *and* j *for distinct expansions* σ_k *and* $\sigma_{k'}$ *are not required to be the same. Suppose that* $\boldsymbol{x} = \sigma_n \cdots \sigma_2 \cdot \sigma_1(a)$. *Then* $\boldsymbol{x}$ *is called a* **two-pattern string of scope** λ *if each* $\sigma_k : a \to \boldsymbol{p}^i\boldsymbol{q}, b \to \boldsymbol{p}^j\boldsymbol{q}$ *satisfies* $|\boldsymbol{p}|, |\boldsymbol{q}| \le \lambda$.

In Section 2 we show that two-pattern strings with scope λ can be recognized in $O(\lambda^4 n)$ time; furthermore, that when $\boldsymbol{x}$ turns out in fact to be a two-pattern string with scope λ, the expansion sequence of $\boldsymbol{x}$ can also be generated in $O(\lambda^4 n)$ time. Then in Section 3 we show how to compute all of the repetitions in a two-pattern string with scope λ in $O(Kn)$ time where the constant K depends on λ.

Recall from [FKS00] that a Sturmian string is said to be ***block-complete*** if and only if it can be generated by a sequence of expansions (Definition 3) in which

$$\boldsymbol{p} = a,\ \boldsymbol{q} = b,\ |j-i| = 1, \text{and } \lambda = 1. \tag{1}$$

Thus the restrictions (1) identify the special case of block-complete Sturmian strings. Observe also that the use of the scope does not really restrict the generality of the problem, since all of the possible two-pattern strings with scope λ are included in the set of two-pattern strings with scope $\lambda+1$.

We conclude this section with an example. The two-pattern strings generated from the sequence of expansions

$$\begin{array}{llll}
\sigma_1 : & a \to (a)^2 b, & b \to (a)^3 b & (\boldsymbol{p}_1 = a, \boldsymbol{q}_1 = b, i = 2, j = 3) \\
\sigma_2 : & a \to (ab)^2 bb, & b \to (ab)^3 bb & (\boldsymbol{p}_2 = ab, \boldsymbol{q}_2 = bb, i = 2, j = 3) \\
\sigma_3 : & a \to ab, & b \to (a)^3 b & (\boldsymbol{p}_3 = a, \boldsymbol{q}_3 = b, i = 1, j = 3)
\end{array}$$

are as follows:

$$\begin{array}{ll}
\sigma_1(a) : & aab \\
\sigma_2 \cdot \sigma_1(a) : & ababbbababbbabababbb \\
\sigma_3 \cdot \sigma_2 \cdot \sigma_1(a) : & baaababaaabaaabaaababaaababaaabaaabaaababaaababaaab \\
 & abaaabaaabaaab
\end{array}$$

2 THE RECOGNITION ALGORITHM

We begin by outlining our approach to recognizing whether or not $\boldsymbol{x}$ is a two-pattern string for a given scope λ. At each step we identify the elements of a 4-tuple $(\boldsymbol{p}, \boldsymbol{q}, i, j)$, if it exists, such that

- $\boldsymbol{p}$ and $\boldsymbol{q}$ is a suitable pair of patterns;
- $|\boldsymbol{p}|, |\boldsymbol{q}| \leq \lambda$;
- for some string $\boldsymbol{y}$, $\boldsymbol{x}$ is an expansion of $\boldsymbol{y}$ under the morphism σ defined by the 4-tuple;

The 4-tuple then also defines an inverse morphism, or ***reduction***,

$$\sigma^{-1}: \ \boldsymbol{p}^i\boldsymbol{q} \rightarrow a, \boldsymbol{p}^j\boldsymbol{q} \rightarrow b, \tag{2}$$

such that $\sigma^{-1}(\boldsymbol{x}) = \boldsymbol{y}$. The recognition algorithm executes simply by performing successive reductions on $\boldsymbol{x}$, while recording at each step the corresponding 4-tuple. If at any step no 4-tuple can be found within the given scope λ, we conclude that $\boldsymbol{x}$ is not a two-pattern string for that scope; while if a sequence of reductions to the letter a can be found, then $\boldsymbol{x}$ is in fact a two-pattern string of scope λ. It is convenient to introduce the idea of a 2***-cover*** of $\boldsymbol{x}$; that is, a pair of strings $\boldsymbol{u}, \boldsymbol{v}$ such that $\boldsymbol{x}$ may be constructed by concatenating both of them, with each string occurring at least once. A straightforward algorithm may then be outlined as shown in Figure 1. It utilizes two simple algorithms, PRIMITIV($\boldsymbol{p}$) that returns `true` whenever the input string $\boldsymbol{p}$ is primitive (in $O(|\boldsymbol{p}|)$ steps), and SUITABLE($\boldsymbol{p}$,$\boldsymbol{q}$) that returns `true` whenever $\boldsymbol{p}$, $\boldsymbol{q}$ is a suitable pair of patterns (in $O(|\boldsymbol{p}||\boldsymbol{q}|)$ steps). Their design is left to the interested reader.

For $n > 2\lambda$, Algorithm RECOGNIZE$_\lambda$ considers in increasing order of length the possible choices of $\boldsymbol{p}$ and $\boldsymbol{q}$, $|\boldsymbol{p}| \leq \lambda$, $|\boldsymbol{q}| \leq \lambda$. The processing depends critically on the assumption that $\boldsymbol{p}$ is not a prefix of $\boldsymbol{q}$: this assumption permits an exact count to be made of the number k of consecutive occurrences of $\boldsymbol{p} = \boldsymbol{x}[1..r]$ at the beginning of $\boldsymbol{x}$. Thus for each choice of r the starting position of the first occurrence of $\boldsymbol{q}$ is well defined, and so the only possible instances of $\boldsymbol{q}$ are $\boldsymbol{x}[kr+1..kr+s]$, $s = 1, 2, \ldots, \lambda$. For each of the at most λ^2 possible choices of $\boldsymbol{p}$ and $\boldsymbol{q}$, a simple left-to-right scan of $\boldsymbol{x}$ determines in $O(|\boldsymbol{x}|)$ time whether there exist integers $0 \leq i < j$ such that $(\boldsymbol{p}^i\boldsymbol{q}, \boldsymbol{p}^j\boldsymbol{q})$ is a 2-cover of $\boldsymbol{x}$. RECOGNIZE$_\lambda$ is a recursive algorithm that

```
boolean RECOGNIZE_λ (x)

  — Deal first with trivial cases
if |x| = 1 then return true
if x = a^|x| or b^|x| then return false

for r ← 1 to min(λ, |x|) do
   p ← x[1..r]   — a candidate for p
   if not PRIMITIV(p) then continue forloop for next r
   compute the maximum k such that x[1..kr] = p^k
   for s ← 1 to min(λ, |x|) do
      q ← x[kr+1..kr+s]   — a candidate for q
      if not SUITABLE(p,q) then
         continue forloop for next s
         if for some 0 ≤ i < j, {p^i q, p^j q} is a
2-cover of
         x then
            output (p, q, i, j)   — which defines σ
            return RECOGNIZE_λ (σ^{-1}(x))
      endif
   enfdor
   return false   — as we did not find a suitable q
endfor
return false   — as we did not find a suitable p
```

Figure 1: Recursive Recognition of a Two-Pattern String

reduces the length of $\boldsymbol{x}$ by a factor of at least 2 at each step, and thus over all recursive calls, at most $2|\boldsymbol{x}|$ positions need to be scanned. Based on this discussion, and knowing that each call to PRIMITIVE requires at most λ steps and each call to SUITABLE at most λ^2 steps, we state formally the main result of this section:

Theorem 1 *For every fixed integer $\lambda \geq 1$, Algorithm RECOGNIZE$_\lambda$ determines in $O(\lambda^4 n)$ time whether or not $\boldsymbol{x}$ is a two-pattern string of scope λ, and, if so, outputs its reduction sequence, also in $O(\lambda^4 n)$ time.* □

A more detailed investigation of RECOGNIZE reveals that in fact only one pair of suitable patterns (if found) is ever tried (of course, on each level of reduction) due to the lexicographic order of generating the candidates for $\boldsymbol{p}$ and candidates for $\boldsymbol{q}$. It is, in a sence, a minimal such pair, where the minimality is first determined by the length of $\boldsymbol{p}$ and then by the length of $\boldsymbol{q}$. This observation raises the question whether RECOGNIZE in fact recognizes all strings defined in 4. Is it possible that there was another suitable pair (with either a bigger $\boldsymbol{p}$ or the same $\boldsymbol{p}$ but bigger $\boldsymbol{q}$) forming a 2-cover of $\boldsymbol{x}$ and allowing the whole recursion to complete successfully, while the recursion with the minimal pair did not finish successfully? If it was possible, the algorithm would either not recognize all two-pattern strings, or would have to include a backracking blowing its complexity to $O(n\ log\ n)$, both rather undesirable. Luckily, the following argument shows that it cannot happen.

So let $\boldsymbol{x}$ be a concatenation of $\boldsymbol{p}_1^{i_1}\boldsymbol{q}_1$ and $\boldsymbol{p}_1^{j_1}\boldsymbol{q}_1$, and at the same time a concatenation of $\boldsymbol{p}_2^{i_2}\boldsymbol{q}_2$ and $\boldsymbol{p}_2^{j_2}\boldsymbol{q}_2$, $\boldsymbol{p}_1, \boldsymbol{q}_1$ and $\boldsymbol{p}_2, \boldsymbol{q}_2$ being suitable pairs of patterns, the former "smaller" than the latter. If $|\boldsymbol{p}_1| = |\boldsymbol{p}_2|$, then clearly $\boldsymbol{p}_1 = \boldsymbol{p}_2$. It follows that either $\boldsymbol{q}_1 = \boldsymbol{q}_2$ or one of them is $\boldsymbol{p}_1$-regular, a contradiction. So we have to assume that $|\boldsymbol{p}_1| < |\boldsymbol{p}_2|$. If ever a $\boldsymbol{p}_2$ located to the left of a $\boldsymbol{p}_1$ overlaps it, then we have a contradiction with the primitivness of $\boldsymbol{p}_2$. Thus each right end of a $\boldsymbol{p}_2$ is located within a $\boldsymbol{q}_1$. Thus $\boldsymbol{p}_2 = \boldsymbol{p}_1^{k_1}\boldsymbol{q}_1 \cdots \boldsymbol{p}_1^{k_n}\boldsymbol{u}$ for some $n \geq 1$, $\boldsymbol{u}$ is a prefix of $\boldsymbol{q}_1$, and each $k_1 \cdots k_n$ equals either i_1 or j_1. If we ever had two adjacent $\boldsymbol{p}_2$'s, it would either violate the primitivness of $\boldsymbol{p}_1$ or imply that $\boldsymbol{q}_1$ is $\boldsymbol{p}_1$-regular, both forbidden. Therefeore $\boldsymbol{x} = (\boldsymbol{p}_2\boldsymbol{q}_2)^m$ for some m, which would reduce $\boldsymbol{x}$ to a^m, which would fail the algorithm.

3 COMPUTING THE REPETITIONS

In this section we describe the algorithm $\mathcal{REP}_\lambda$ that, given a two-pattern string $\boldsymbol{x}$ with a scope λ and its reduction sequence, computes all the repetitions in $\boldsymbol{x}$ in time $\Theta(K_\lambda|\boldsymbol{x}|)$, where the constant K_λ depends only on λ.

The main idea of the algorithm can be described in the following way: if $\boldsymbol{y}$ is a reduction of $\boldsymbol{x}$, then all repetitions in $\boldsymbol{x}$ can be derived through a formula from repetitions in $\boldsymbol{y}$ and other linear configurations in $\boldsymbol{y}$. The repetitions in $\boldsymbol{y}$ and the linear configurations are themselves derived (in a recursive manner) from repetitions and linear configurations in its reduction. Since the required contribution at each level of reduction (recursion) is linear, the whole algorithm finishes the task in a linear time.

First we recall the Crochemore encoding of a ***repetition*** [C81]: for a given string $\boldsymbol{x}$, (s,l,e), $e \geq 2$, means that $\boldsymbol{x}[s+i] = \boldsymbol{x}[s+jl+i]$ for any $0 \leq i < l$, $0 \leq j < e$. We say that a repetition (s,l,e) in $\boldsymbol{x}$ can be **shifted** r **positions to the right (left)** if $(s+r,l,e)$ (respectively, $(s-r,l,e)$) is a repetition in $\boldsymbol{x}$. We say further that (s,l,e,r) is a **run** in $\boldsymbol{x}$ if (s,l,e) is a repetition in $\boldsymbol{x}$ that can be shifted at most r positions to the right and 0 positions to the left. A repetition (s,l,e) (or run (s,l,e,r)) is **irreducible** if $\boldsymbol{x}[i..i+l-1]$ is not a repetition.

Before we can present $\mathcal{REL}_\lambda$, we need to state and prove eight theorems Theorem 2-Theorem 9 that are used to show that (i) the algorithm works correctly (i.e. calculates all runs in $\boldsymbol{x}$), and (ii) works in a linear time. The proofs of these theorems are not very hard, though they are extremly tedious and lengthy, as they have to cover many possibilities that must be dealt with individually. Thus we decided to skip the proofs for this publication, the interested reader can find the full version of this paper with all proofs at the web site of one of the authors, at URL `http://www.cas.mcmaster.ca/~franya`

The following theorem shows that runs with "small" generators (i.e. smaller or equal to a fixed κ, which for $\mathcal{REL}_\lambda$ will in fact be 3λ) can be calculated in time that solely depends on κ only, and thus we shall have to concentrate on calculating runs with "large" (i.e. bigger than κ) generators.

Theorem 2 *There is an algorithm that for any $\kappa \geq 1$ and any input binary string $\boldsymbol{x}$ outputs all runs in $\boldsymbol{x}$ whose generator is of a size $\leq \kappa$ in $\leq (2^{\kappa+1}-2)8\kappa^2|\boldsymbol{x}|$ steps.*

The following theorem summarizes how repetitions of "large" substrings of $\boldsymbol{x}$ relate to repetitions and certain linear configurations in $\boldsymbol{y}$, the reduction of $\boldsymbol{x}$.

Theorem 3 *Let $\boldsymbol{p},\boldsymbol{q}$ be a suitable pair of patterns. Let $\boldsymbol{x}$ be a concatenation of blocks $\boldsymbol{p}^i\boldsymbol{q}$, $\boldsymbol{p}^j\boldsymbol{q}$, $0 \leq i < j$. Let $\boldsymbol{y}$ be a reduction of $\boldsymbol{x}$ determined by the inverse of the morphism $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}$, $b \to \boldsymbol{p}^j\boldsymbol{q}$. Let λ be so that $|\boldsymbol{p}|, |\boldsymbol{q}| \leq \lambda$. Then for every run in $\boldsymbol{x}$ with a generator of a size $> 3\lambda$, one of the following holds:*

1. *the run is an expansion (by α^{-1}) of a run from $\boldsymbol{y}$;*
2. *$j = i+l$ and the run is a square $\boldsymbol{uu}$ derived from a configuration $a\boldsymbol{v}b\boldsymbol{v}a$ in $\boldsymbol{y}$ in the following way: $\boldsymbol{uu} = \boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}\boldsymbol{p}^l\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}\boldsymbol{p}^l$, which is a substring of $\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}\boldsymbol{p}^j\boldsymbol{w}\boldsymbol{p}^j$, which is an expansion of $a \cdots b \cdots a$;*

3. *the run is a square* $\boldsymbol{uu}$ *derived from a configuration* $b\boldsymbol{v}a\boldsymbol{v}b$ *in* $\boldsymbol{y}$ *in the following way:* $\boldsymbol{uu} = \boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}$*, which is a substring of* $\boldsymbol{p}^j\boldsymbol{q}\boldsymbol{w}\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}$*, the expansion of* $b\cdots a\cdots$*;*

4. *the run is a square* $\boldsymbol{uu}$ *derived from a configuration* $b\boldsymbol{v}a\boldsymbol{v}$ *that is a suffix of* $\boldsymbol{x}$ *in the following way:* $\boldsymbol{uu} = \boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}$*, which is a substring of* $\boldsymbol{p}^j\boldsymbol{q}\boldsymbol{w}\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{w}$*, the expansion of* $b\cdots a\cdots$*;*

5. $\boldsymbol{q} = \boldsymbol{p}_1\boldsymbol{p}_2$*,* $\boldsymbol{p}_1$ *is a prefix of* $\boldsymbol{p}$*, and* $\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$ *and the run is a square* $\boldsymbol{uu}$ *derived from configurations aa, ab, ba, bb in* $\boldsymbol{y}$ *(*$\boldsymbol{uu} = (\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1)(\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1)$*, which is a substring of* $\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{q}\boldsymbol{p}^r\boldsymbol{p}_1$*, which is a substring of* $\boldsymbol{p}^{r+1}\boldsymbol{q}\boldsymbol{p}^{r+1}$*, which is a substring of* $\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}$ *if* $r < i$*, of* $\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$ *if* $r < i$*, of* $\boldsymbol{p}^j\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}$ *if* $r < i$*, and of* $\boldsymbol{p}^j\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$ *if* $r < j$*).*

6. $\boldsymbol{q} = \boldsymbol{p}_1\boldsymbol{p}_2$*,* $\boldsymbol{p}_1$ *is a prefix of* $\boldsymbol{p}$*, and* $\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$ *and the run is a square* $\boldsymbol{uu}$ *derived from a configuration* $b\boldsymbol{v}aa\boldsymbol{v}b$ *in* $\boldsymbol{y}$ *(*$\boldsymbol{uu} = (\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1)(\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1)$*, which is a substring of* $\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j$*, which is a substring of* $\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}$*).*

7. $\boldsymbol{p} = \boldsymbol{q}_1\boldsymbol{q}_2$*,* $\boldsymbol{q}_1$ *is a prefix of* $\boldsymbol{q}$*,* $\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$*, i or j is odd, and the run is a square* $\boldsymbol{uu}$ *derived from configurations aa, ab, ba, bb in* $\boldsymbol{y}$ *(*$\boldsymbol{uu} = (\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)(\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)$*, which is a substring of* $\boldsymbol{q}_2\boldsymbol{p}^{2r+1}\boldsymbol{q}_1$*, which is a substring of* $\boldsymbol{q}\boldsymbol{p}^{2r+1}\boldsymbol{q}$*, which is a substring of* $\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$ *if* $i = 2r+1$*, of* $\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$ *if* $j = 2r+1$*, of* $\boldsymbol{p}^j\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}$ *if* $i = 2r+1$*, of* $\boldsymbol{p}^j\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$ *if* $j = 2r+1$*).*

8. $\boldsymbol{p} = \boldsymbol{q}_1\boldsymbol{q}_2$*,* $\boldsymbol{q}_1$ *is a prefix of* $\boldsymbol{q}$*,* $\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$*,* $j = 2i+1$*, and the run is a square* $\boldsymbol{uu}$ *derived from a configuration* $a\boldsymbol{v}b\boldsymbol{v}a$ *in* $\boldsymbol{y}$ *(*$\boldsymbol{uu} = (\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)(\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)$*, which is a substring of* $\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1)(\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1)$*, which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{2i+1}\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}$*).*

Theorem 4 *Let* $\boldsymbol{p}, \boldsymbol{q}$ *be a suitable pair of patterns. Let* $\boldsymbol{x}$ *be a concatenation of blocks* $\boldsymbol{p}^i\boldsymbol{q}$*,* $\boldsymbol{p}^j\boldsymbol{q}$*,* $0 \le i < j$*. Let* $\boldsymbol{y}$ *be a reduction of* $\boldsymbol{x}$ *determined by the inverse of the morphism* $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}$*,* $b \to \boldsymbol{p}^j\boldsymbol{q}$*. Let* λ *be so that* $|\boldsymbol{p}|, |\boldsymbol{q}| \le \lambda$*. Then for every configuration* $a\boldsymbol{u}b\boldsymbol{u}a$ *in* $\boldsymbol{x}$ *with* $|\boldsymbol{u}| > 3\lambda$*, one of the following holds:*

1. *the configuration* $a\boldsymbol{u}b\boldsymbol{u}a$ *can be derived from aa, ab, or ba configurations in* $\boldsymbol{y}$*: if* $\boldsymbol{q} = \boldsymbol{p}_1 b\boldsymbol{p}_2$*,* $a\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$*,* $\boldsymbol{p}_1 a$ *is a prefix of* $\boldsymbol{p}$*, then for any* $0 \le r < i$*,* $a\boldsymbol{u}b\boldsymbol{u}a = a\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 a = a\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{q}\boldsymbol{p}^r\boldsymbol{p}_1 a$ *which is a substring of* $\boldsymbol{p}^{r+1}\boldsymbol{q}\boldsymbol{p}^{r+1}$*;*

2. *the configuration* $a\boldsymbol{u}b\boldsymbol{u}a$ *can be derived from a bb configuration in* $\boldsymbol{y}$*: if* $\boldsymbol{q} = \boldsymbol{p}_1 b \boldsymbol{p}_2$, $a\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 a$ *is a prefix of* $\boldsymbol{p}$, *then for any* $0 \leq r < j$, $a\boldsymbol{u}b\boldsymbol{u}a = a\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 a = a\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{q}\boldsymbol{p}^r\boldsymbol{p}_1 a$, *which is a substring of* $\boldsymbol{p}^{r+1}\boldsymbol{q}\boldsymbol{p}^{r+1}$*;*

3. *the configuration* $a\boldsymbol{u}b\boldsymbol{u}a$ *can be derived from aa or ba configurations in* $\boldsymbol{y}$*: if* i *is odd,* $i \geq 1$, $\boldsymbol{p} = \boldsymbol{q}_1 b \boldsymbol{q}_2$, $a\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 a$ *is a prefix of* $\boldsymbol{q}$, *then for* $r = \frac{i-1}{2}$, $a\boldsymbol{u}b\boldsymbol{u}a = a\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 b\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 a = a\boldsymbol{q}_2\boldsymbol{p}^{2r+1}\boldsymbol{q}_1 a = a\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}_1 a$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}$*;*

4. *the configuration* $a\boldsymbol{u}b\boldsymbol{u}a$ *can be derived from ab or bb configurations in* $\boldsymbol{y}$*: if* j *is odd,* $\boldsymbol{p} = \boldsymbol{q}_1 b \boldsymbol{q}_2$, $a\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 a$ *is a prefix of* $\boldsymbol{q}$, *then for* $r = \frac{j-1}{2}$, $a\boldsymbol{u}b\boldsymbol{u}a = a\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 b\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 a = a\boldsymbol{q}_2\boldsymbol{p}^{2r+1}\boldsymbol{q}_1 a = a\boldsymbol{q}_2\boldsymbol{p}^j\boldsymbol{q}_1 a$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$*;*

5. *the configuration* $a\boldsymbol{u}b\boldsymbol{u}a$ *can be derived from a* $b\boldsymbol{u}aa\boldsymbol{u}b$ *configuration in* $\boldsymbol{y}$*: if* $\boldsymbol{q} = \boldsymbol{p}_1 b \boldsymbol{p}_2$, $a\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 a$ *is a prefix of* $\boldsymbol{p}$, *then* $a\boldsymbol{u}b\boldsymbol{u}a = a\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 b\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 a = a\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 a$, *which is a substring of* $\boldsymbol{p}^{i+1}\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{i+1}$, *which is a substring of* $\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}$*;*

6. *the configuration* $a\boldsymbol{u}b\boldsymbol{u}a$ *can be derived from a* $\cdot a\boldsymbol{u}b\boldsymbol{u}a$ *configuration in* $\boldsymbol{y}$*: if* $j = 2i+1$, $\boldsymbol{p} = \boldsymbol{q}_1 b \boldsymbol{q}_2$, $a\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 a$ *is a prefix of* $\boldsymbol{q}$, *then* $a\boldsymbol{u}b\boldsymbol{u}a = a\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 a =$
$a\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{2i+1}\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 a = a\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 a$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}$.

Theorem 5 *Let* $\boldsymbol{p}, \boldsymbol{q}$ *be a suitable pair of patterns. Let* $\boldsymbol{x}$ *be a concatenation of blocks* $\boldsymbol{p}^i\boldsymbol{q}$, $\boldsymbol{p}^j\boldsymbol{q}$, $0 \leq i < j$. *Let* $\boldsymbol{y}$ *be a reduction of* $\boldsymbol{x}$ *determined by the inverse of the morphism* $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}$, $b \to \boldsymbol{p}^j\boldsymbol{q}$. *Let* λ *be so that* $|\boldsymbol{p}|, |\boldsymbol{q}| \leq \lambda$. *Then for every configuration* $b\boldsymbol{u}a\boldsymbol{u}b$ *in* $\boldsymbol{x}$ *with* $|\boldsymbol{u}| > 3\lambda$, *one of the following holds:*

1. *the configuration* $b\boldsymbol{u}a\boldsymbol{u}b$ *can be derived from aa, ab, or ba configurations in* $\boldsymbol{y}$*: if* $\boldsymbol{q} = \boldsymbol{p}_1 a \boldsymbol{p}_2$, $b\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 b$ *is a prefix of* $\boldsymbol{p}$, *then for any* $0 \leq r < i$, $b\boldsymbol{u}a\boldsymbol{u}b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 a\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{q}\boldsymbol{p}^r\boldsymbol{p}_1 b$ *which is a substring of* $\boldsymbol{p}^{r+1}\boldsymbol{q}\boldsymbol{p}^{r+1}$*;*

2. *the configuration* $b\boldsymbol{u}a\boldsymbol{u}b$ *can be derived from a bb configuration in* $\boldsymbol{y}$*: if* $\boldsymbol{q} = \boldsymbol{p}_1 a \boldsymbol{p}_2$, $b\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 b$ *is a prefix of* $\boldsymbol{p}$, *then for any*

$0 \leq r < j$, $b\boldsymbol{u}a\boldsymbol{u}b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 a\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{q}\boldsymbol{p}^r\boldsymbol{p}_1 b$, *which is a substring of* $\boldsymbol{p}^{r+1}\boldsymbol{q}\boldsymbol{p}^{r+1}$;

3. *the configuration* $b\boldsymbol{u}a\boldsymbol{u}b$ *can be derived from* aa *or* ba *configurations in* $\boldsymbol{y}$: *if* i *is odd,* $i \geq 1$, $\boldsymbol{p} = \boldsymbol{q}_1 a\boldsymbol{q}_2$, $b\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 b$ *is a prefix of* $\boldsymbol{q}$, *then for* $r = \frac{i-1}{2}$, $b\boldsymbol{u}a\boldsymbol{u}b = b\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 a\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^{2r+1}\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}_1 b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}$;

4. *the configuration* $b\boldsymbol{u}a\boldsymbol{u}b$ *can be derived from* ab *or* bb *configurations in* $\boldsymbol{y}$: *if* j *is odd,* $\boldsymbol{p} = \boldsymbol{q}_1 a\boldsymbol{q}_2$, $b\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 b$ *is a prefix of* $\boldsymbol{q}$, *then for* $r = \frac{j-1}{2}$, $b\boldsymbol{u}a\boldsymbol{u}b = b\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 a\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^{2r+1}\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^j\boldsymbol{q}_1 b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$;

5. *the configuration* $b\boldsymbol{u}a\boldsymbol{u}b$ *can be derived from a* $b\boldsymbol{u}aa\boldsymbol{u}b$ *configuration in* $\boldsymbol{y}$: *if* $\boldsymbol{q} = \boldsymbol{p}_1 a\boldsymbol{p}_2$, $b\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 b$ *is a prefix of* $\boldsymbol{p}$, *then* $b\boldsymbol{u}a\boldsymbol{u}b = b\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 a\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 b = b\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 b$, *which is a substring of* $\boldsymbol{p}^{i+1}\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{i+1}$, *which is a substring of* $\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}$;

6. *the configuration* $b\boldsymbol{u}a\boldsymbol{u}b$ *can be derived from a* $\cdot a\boldsymbol{u}b\boldsymbol{u}a$ *configuration in* $\boldsymbol{y}$: *if* $j = 2i+1$, $\boldsymbol{p} = \boldsymbol{q}_1 a\boldsymbol{q}_2$, $b\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 b$ *is a prefix of* $\boldsymbol{q}$, *then* $b\boldsymbol{u}a\boldsymbol{u}b = b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 a\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{2i+1}\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}$.

Theorem 6 *Let* $\boldsymbol{p}, \boldsymbol{q}$ *be a suitable pair of patterns. Let* $\boldsymbol{x}$ *be a concatenation of blocks* $\boldsymbol{p}^i\boldsymbol{q}$, $\boldsymbol{p}^j\boldsymbol{q}$, $0 \leq i < j$. *Let* $\boldsymbol{y}$ *be a reduction of* $\boldsymbol{x}$ *determined by the inverse of the morphism* $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}$, $b \to \boldsymbol{p}^j\boldsymbol{q}$. *Let* λ *be so that* $|\boldsymbol{p}|, |\boldsymbol{q}| \leq \lambda$. *Then for every configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *in* $\boldsymbol{x}$ *with* $|\boldsymbol{u}| > 3\lambda$, *one of the following holds:*

1. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from* aa, ab, ba, *or* bb *configurations in* $\boldsymbol{y}$: *if* $\boldsymbol{q} = \boldsymbol{p}_1 aa\boldsymbol{p}_2$, $b\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 b$ *is a prefix of* $\boldsymbol{p}$, *then for any* $0 \leq r < i$, $b\boldsymbol{u}aa\boldsymbol{u}b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 aa\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{q}\boldsymbol{p}^r\boldsymbol{p}_1 b$ *which is a substring of* $\boldsymbol{p}^{r+1}\boldsymbol{q}\boldsymbol{p}^{r+1}$;

2. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from a* bb *configuration in* $\boldsymbol{y}$: *if* $\boldsymbol{q} = \boldsymbol{p}_1 aa\boldsymbol{p}_2$, $b\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 b$ *is a prefix of* $\boldsymbol{p}$, *then for any* $0 \leq r < j$, $b\boldsymbol{u}aa\boldsymbol{u}b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 aa\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{p}_1 b = b\boldsymbol{p}_2\boldsymbol{p}^r\boldsymbol{q}\boldsymbol{p}^r\boldsymbol{p}_1 b$, *which is a substring of* $\boldsymbol{p}^{r+1}\boldsymbol{q}\boldsymbol{p}^{r+1}$;

3. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from* aa *or* ba *configurations in* $\boldsymbol{y}$*: if* i *is even,* $i \geq 2$, $\boldsymbol{q} = b\boldsymbol{q}_1 b$ *or* $\boldsymbol{q} = b$, $\boldsymbol{p} = a$, *then for* $r = \frac{i-2}{2}$, $b\boldsymbol{u}aa\boldsymbol{u}b = b(\boldsymbol{p}^r)aa(\boldsymbol{p}^r)b = b\boldsymbol{p}^{2r+2}b = b\boldsymbol{p}^i b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}$*;*

4. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from* ab *or* bb *configurations in* $\boldsymbol{y}$*: if* j *is even,* $j \geq 2$, $\boldsymbol{q} = b\boldsymbol{q}_1 b$ *or* $\boldsymbol{q} = b$, $\boldsymbol{p} = a$, *then for* $r = \frac{j-2}{2}$, $b\boldsymbol{u}aa\boldsymbol{u}b = b(\boldsymbol{p}^r)aa(\boldsymbol{p}^r)b = b\boldsymbol{p}^{2r+2}b = b\boldsymbol{p}^j b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$*;*

5. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from* aa *or* ba *configurations in* $\boldsymbol{y}$*: if* i *is odd,* $i \geq 1$, $\boldsymbol{p} = \boldsymbol{q}_1 aa\boldsymbol{q}_2$, $b\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 b$ *is a prefix of* $\boldsymbol{q}$, *then for* $r = \frac{i-1}{2}$, $b\boldsymbol{u}aa\boldsymbol{u}b = b(\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)aa(\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)b = b\boldsymbol{q}_2\boldsymbol{p}^{2r+1}\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}_1 b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}$*;*

6. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from* ab *or* bb *configurations in* $\boldsymbol{y}$*: if* j *is odd,* $\boldsymbol{p} = \boldsymbol{q}_1 aa\boldsymbol{q}_2$, $b\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 b$ *is a prefix of* $\boldsymbol{q}$, *then for* $r = \frac{j-1}{2}$, $b\boldsymbol{u}aa\boldsymbol{u}b = b(\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)aa(\boldsymbol{q}_2\boldsymbol{p}^r\boldsymbol{q}_1)b = b\boldsymbol{q}_2\boldsymbol{p}^{2r+1}\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^j\boldsymbol{q}_1 b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^j\boldsymbol{q}$*;*

7. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from a* $b\boldsymbol{u}aa\boldsymbol{u}b$ *configuration in* $\boldsymbol{y}$*: if* $\boldsymbol{q} = \boldsymbol{p}_1 aa\boldsymbol{p}_2$, $b\boldsymbol{p}_2$ *is a suffix of* $\boldsymbol{p}$, $\boldsymbol{p}_1 b$ *is a prefix of* $\boldsymbol{p}$, *then* $b\boldsymbol{u}aa\boldsymbol{u}b = b\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 aa\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 b =$
$b\boldsymbol{p}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{p}_1 b$, *which is a substring of*
$\boldsymbol{p}^{i+1}\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{i+1}$, *which is a substring of* $\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}$*;*

8. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from a* $\cdot a\boldsymbol{u}b\boldsymbol{u}a$ *configuration in* $\boldsymbol{y}$*: if* $j = 2i{+}2$, $\boldsymbol{q} = b\boldsymbol{q}_1 b$ *or* $\boldsymbol{q} = b$, $\boldsymbol{p} = a$, *then* $b\boldsymbol{u}aa\boldsymbol{u}b =$
$b)\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i)aa(\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i)b = b\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{2i+2}\boldsymbol{q}\cdots\boldsymbol{p}^i b =$
$b\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}$.

9. *the configuration* $b\boldsymbol{u}aa\boldsymbol{u}b$ *can be derived from a* $\cdot a\boldsymbol{u}b\boldsymbol{u}a$ *configuration in* $\boldsymbol{y}$*: if* $j = 2i{+}1$, $\boldsymbol{p} = \boldsymbol{q}_1 aa\boldsymbol{q}_2$, $b\boldsymbol{q}_2$ *is a suffix of* $\boldsymbol{q}$, $\boldsymbol{q}_1 b$ *is a prefix of* $\boldsymbol{q}$, *then* $b\boldsymbol{u}aa\boldsymbol{u}b = b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 aa\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 b =$
$b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^{2i+1}\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 b = b\boldsymbol{q}_2\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}_1 b$, *which is a substring of* $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}$.

Theorem 7 *Let* $\boldsymbol{p}, \boldsymbol{q}$ *be a suitable pair of patterns. Let* $\boldsymbol{x}$ *be a concatenation of blocks* $\boldsymbol{p}^i\boldsymbol{q}$, $\boldsymbol{p}^j\boldsymbol{q}$, $0 \leq i < j$. *Let* $\boldsymbol{y}$ *be a reduction of* $\boldsymbol{x}$ *determined by the inverse of the morphism* $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}$, $b \to \boldsymbol{p}^j\boldsymbol{q}$. *Let* λ *be so that* $|\boldsymbol{p}|, |\boldsymbol{q}| \leq \lambda$. *Then every configuration* $b\boldsymbol{u}a\boldsymbol{u}$ *that is a suffix of* $\boldsymbol{x}$ *with* $|\boldsymbol{u}| > 3\lambda$ *is derived from a* $\cdot a\boldsymbol{v}b\boldsymbol{v}$ *configuration that is a suffix of* $\boldsymbol{y}$*: if* $\boldsymbol{q} = \boldsymbol{q}_1 b\boldsymbol{q}_2$, $\boldsymbol{p} = a\boldsymbol{q}_2$,

$j = i{+}1$, $b\boldsymbol{u}a\boldsymbol{u} = b(\boldsymbol{q}_2\cdots\boldsymbol{q})a(\boldsymbol{q}_2\cdots\boldsymbol{q}) = b\boldsymbol{q}_2\cdots\boldsymbol{q}\boldsymbol{p}\cdots\boldsymbol{q}$ which is a substring of $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}$.

Theorem 8 *Let $\boldsymbol{p},\boldsymbol{q}$ be a suitable pair of patterns. Let $\boldsymbol{x}$ be a concatenation of blocks $\boldsymbol{p}^i\boldsymbol{q}$, $\boldsymbol{p}^j\boldsymbol{q}$, $0 \le i < j$. Let $\boldsymbol{y}$ be a reduction of $\boldsymbol{x}$ determined by the inverse of the morphism $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}$, $b \to \boldsymbol{p}^j\boldsymbol{q}$. Let λ be so that $|\boldsymbol{p}|, |\boldsymbol{q}| \le \lambda$. Then every configuration $a\boldsymbol{u}b\boldsymbol{u}$ that is a suffix of $\boldsymbol{x}$ with $|\boldsymbol{u}| > 3\lambda$ is derived from a $\cdot a\boldsymbol{v}b\boldsymbol{v}$ configuration that is a suffix of $\boldsymbol{y}$: if $\boldsymbol{q} = \boldsymbol{q}_1a\boldsymbol{q}_2$, $\boldsymbol{p} = b\boldsymbol{q}_2$, $j = i{+}1$, $a\boldsymbol{u}b\boldsymbol{u} = a(\boldsymbol{q}_2\cdots\boldsymbol{q})b(\boldsymbol{q}_2\cdots\boldsymbol{q}) = a\boldsymbol{q}_2\cdots\boldsymbol{q}\boldsymbol{p}\cdots\boldsymbol{q}$ which is a substring of $\boldsymbol{q}\boldsymbol{p}^i\boldsymbol{q}\cdots\boldsymbol{p}^j\boldsymbol{q}\cdots\boldsymbol{p}^i\boldsymbol{q}$.*

The last missing piece before we can describe the repetition algorithm involves a description of how to "expand" runs from a reduction $\boldsymbol{y}$ to $\boldsymbol{x}$ using the inverse of the morphism $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}, b \to \boldsymbol{p}^j\boldsymbol{q}$. Of course, we assume that $\boldsymbol{p},\boldsymbol{q}$ is a suitable pair of patterns.

We shall keep track of a position not in the usual way using index 1 to the size of the string, but rather in the number of a's and b's that precede the given position. This is necessary, for the expansion is uneven in that a will be expanded to $\boldsymbol{p}^i\boldsymbol{q}$ while b to $\boldsymbol{p}^j\boldsymbol{q}$, hence they will have different lengths. Thus a run for us is an ordered 4-tuple $((s_a, s_b), (l_a, l_b), e, (r_a, r_b))$, where the pair (s_a, s_b) determines the position at which the generator of the run starts (s_a being the number of a's and s_b the number of b's that precede the position where the generator starts, hence the start is at the position $s_a{+}s_b{+}1$), the pair (l_a, l_b) determines the length of the generator (i.e. the length is $l_a{+}l_b$), e is the power of the run, and the pair (r_a, r_b) determines the number of positions it can be shifted to the right (i.e. we can shift $r_a{+}r_b$ positions to the right).

$\boldsymbol{p}_a$ denotes the number of a's in $\boldsymbol{p}$, while $\boldsymbol{p}_b$ denotes the number of b's in $\boldsymbol{p}$. Similarly $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$. $GCS(\boldsymbol{p},\boldsymbol{q})$ denotes the greatest common suffix of $\boldsymbol{p}$ and $\boldsymbol{q}$ and $GCS(\boldsymbol{p},\boldsymbol{q})_a$ (or $GCS(\boldsymbol{p},\boldsymbol{q})_b$) denotes the number of a's (or b's) in $GCS(\boldsymbol{p},\boldsymbol{q})$. $GCP(\boldsymbol{p},\boldsymbol{q})$ denotes the greatest common prefix of $\boldsymbol{p}$ and $\boldsymbol{q}$, and $GCP(\boldsymbol{p},\boldsymbol{q})_a$ (or $GCP(\boldsymbol{p},\boldsymbol{q})_b$) denotes the number of a's (or b's) in $GCP(\boldsymbol{p},\boldsymbol{q})$.

Let $((s'_a, s'_b), (l'_a, l'_b), e', (r'_a, r'_b))$ be a run in $\boldsymbol{y}$. Then $LC = \boldsymbol{y}[s'_a{+}s'_b{+}l'_a{+}l'_b]$ is the last character of the generator of the run, while $RP = s'_a{+}s'_b{+}e'(l'_a{+}l'_b){+}r'_a{+}r'_b$ is the position of the last character in the run in its rightmost shift, or, equivalently, the length of $\boldsymbol{y}$ the run covers from its leftmost position to its rightmost position.

Now expand $\boldsymbol{y}$ back to $\boldsymbol{x}$ by the inverse of the morphism α. Since every a is expanded to $\boldsymbol{p}^i\boldsymbol{q}$, after the expansion it contributes $i(\boldsymbol{p}_a+\boldsymbol{q}_a)$ a's and $i(\boldsymbol{p}_b+\boldsymbol{q}_b)$ b's, while each b after the expansion contributes $j(\boldsymbol{p}_a+\boldsymbol{q}_a)$ a's and $j(\boldsymbol{p}_b+\boldsymbol{q}_b)$ b's.

If the generator was not at the beginning of $\boldsymbol{y}$, it could not be shifted left for only one reason, namely that the character just before the generator was different from the last character of the generator. But once we expand these characters, they have a common suffix $\boldsymbol{p}^i\boldsymbol{q}$, so we can shift left by at least that much. If the last character of the generator is a and the generator has size ≥ 2, then there is still more possibility for shifting left, namely the $GCS(\boldsymbol{p},\boldsymbol{q})$. If the last character of the generator is b and there are at least 2 characters to the left of the generator, then again there is still more possibility for shifting left, $GCS(\boldsymbol{p},\boldsymbol{q})$.

c_a, c_b represent the extra left shift we gain after the expansion:

if $(LC = a)$ then *generator ends with 'a'*
  if $(s'_a+s'_b = 0)$ then *generator at the beginning of* $\boldsymbol{x}$
    $c_a = c_b = 0$
  elseif $(l'_a = 1 \& l'_b = 0)$ then *generator contains just one letter - 'a'*
    $c_a = i\boldsymbol{p}_a+\boldsymbol{q}_a$, $c_b = i\boldsymbol{p}_b+\boldsymbol{q}_b$
  else
    $c_a = i\boldsymbol{p}_a+\boldsymbol{q}_a+GCS(\boldsymbol{p},\boldsymbol{q})_a$, $c_b = i\boldsymbol{p}_b+\boldsymbol{q}_b+GCS(\boldsymbol{p},\boldsymbol{q})_b$
else *generator ends with 'b'*
  if $(s'_a+s'_b = 0)$ then *generator at the beginning of* $\boldsymbol{x}$
    $c_a = c_b = 0$
  elseif $(s'_a = 1 \& s'_b = 0)$ then *there is only 'a' left of the generator*
    $c_a = i\boldsymbol{p}_a+\boldsymbol{q}_a$, $c_b = i\boldsymbol{p}_b+\boldsymbol{q}_b$
  else
    $c_a = i\boldsymbol{p}_a+\boldsymbol{q}_a+GCS(\boldsymbol{p},\boldsymbol{q})_a$, $c_b = i\boldsymbol{p}_b+\boldsymbol{q}_b+GCS(\boldsymbol{p},\boldsymbol{q})_b$

Thus, we move the generator to the left as far as it goes
$s_a = (s'_a i+s'_b j)\boldsymbol{p}_a+(s'_a+s'_b)\boldsymbol{q}_a - c_a$, $s_b = (s'_a i+s'_b j)\boldsymbol{p}_b+(s'_a+s'_b)\boldsymbol{q}_b - c_b$.

It is clear, that

$l_a = (l'_a i + l'_b j)\boldsymbol{p}_a + (l'_a + l'_b)\boldsymbol{q}_a$, $s_b = (l'_a i + l'_b j)\boldsymbol{p}_b + (l'_a + l'_b)\boldsymbol{q}_b$.

Even if there was no possible right shift, the extra left shift we gained by the expansion and the subsequent move of the generator to the left means that we gained at least (c_a, c_b) shift to the right. If the original shift to the right stopped at the position RC, then we gain an extra shift to the right defined by the $GCP(\boldsymbol{p}^j\boldsymbol{q}, \boldsymbol{p}^i\boldsymbol{q})$ which is $\boldsymbol{p}^i + GCP(\boldsymbol{p}, \boldsymbol{q})$.

k_a, k_b represent the extra right shift we gain after the expansion:

if $(RC = |y|)$ then *max. right shift ends at the end of* $\boldsymbol{y}$
  $k_a = k_b = 0$
else *max. right shift doesn't end at the end of* $\boldsymbol{y}$
  $k_a = i\boldsymbol{p}_a + \boldsymbol{q}_a + GCP(\boldsymbol{p}, \boldsymbol{q})_a$, $k_b = i\boldsymbol{p}_b + \boldsymbol{q}_b + GCP(\boldsymbol{p}, \boldsymbol{q})_b$

It may happen that the extra left shift and the extra right shift together with the expanded original right shift are bigger than the size of the generator. In such a case, we have to increment the exponent and decrease the right shift accordingly.

Thus
$r_a = (r'_a i + r'_b j)\boldsymbol{p}_a + (r'_a + r'_b)\boldsymbol{q}_a + c_a + k_a$,
$r_b = (r'_a i + r'_b j)\boldsymbol{p}_b + (r'_a + r'_b)\boldsymbol{q}_b + c_b + k_b$.
if $((r_a + r_b) \geq (l_a + l_b))$ then
  $r_a = r_a - l_a$, $r_b = r_b - l_b$, $e = e' + 1$
else
  $e' = e$.

Now we are ready to describe (in a very high-level language to foster comprehension) the repetition algorithm for two-pattern strings with scope λ:

$\mathcal{REP}_\lambda(\boldsymbol{x}, \mathcal{S}\ L_{aa}, L_{ab}, L_{ba}, L_{bb}, L_{aubua}, L_{buaub}, L_{buaaub}, L_{aubu}, L_{buau}, L_{uu})$, where $\boldsymbol{x}$ is the input two-pattern string with scope λ and $\mathcal{S}$ its reduction sequence, while the rest of arguments are output results. L_{aa} is the list of positions in $\boldsymbol{x}$ where aa configuration occurs, similarly for all others. L_{uu} is the list of runs in $\boldsymbol{x}$.

1. Set $L_{aa} \cdots L_{uu}$ to empty sets.

2. If $|\boldsymbol{x}| = 1$, return. *(the base case of recursion)*

3. Remove $(\boldsymbol{p}, \boldsymbol{q}, i, j)$ from $\mathcal{S}$ producing $\mathcal{S}'$. Reduce $\boldsymbol{x}$ to $\boldsymbol{y}$ using the inverse of the morphism $\alpha : a \to \boldsymbol{p}^i\boldsymbol{q}, b \to \boldsymbol{p}^j\boldsymbol{q}$. Set $L'_{aa} \cdots L'_{uu}$ to empty sets. Recursively call $\mathcal{REP}_\lambda(\boldsymbol{y}, \mathcal{S}', L'_{aa}, L'_{ab}, L'_{ba}, L'_{bb}, L'_{aubua}, L'_{buaub}, L'_{buaaub}, L'_{aubu}, L'_{buau}, L'_{uu})$.

In instructions 4-12 all runs in $\boldsymbol{x}$ are calculated.

4. Calculate (using the algorithm from Theorem 2) all runs in $\boldsymbol{x}$ with generators of size $\leq 3\lambda$ and add them to L_{uu}.

5. Expand all runs from L'_{uu} into runs in $\boldsymbol{x}$ using the expansion formula (as described above) and add them to L_{uu}.

6. Derive new runs in $\boldsymbol{x}$ using configurations in L'_{aubua} according to Theorem 3.2 and add them to L_{uu}.

7. Derive new runs in $\boldsymbol{x}$ using configurations in L'_{buaub} according to Theorem 3.3 and add them to L_{uu}.

8. Derive new runs in $\boldsymbol{x}$ using configurations in L'_{buau} according to Theorem 3.4 and add them to L_{uu}.

9. Derive new runs in $\boldsymbol{x}$ using configurations in L'_{aa}, L'_{ab}, L'_{ba}, and L'_{bb} according to Theorem 3.5 and add them to L_{uu}.

10. Derive new runs in $\boldsymbol{x}$ using configurations in L'_{buaaub} according to Theorem 3.6 and add them to L_{uu}.

11. Derive new runs in $\boldsymbol{x}$ using configurations in L'_{aa}, L'_{ab}, L'_{ba}, and L'_{bb} according to Theorem 3.7 and add them to L_{uu}.

12. Derive new runs in $\boldsymbol{x}$ using configurations in L'_{aubua} according to Theorem 3.8 and add them to L_{uu}.

In instruction 13 L_{aa}, L_{ab}, L_{ba}, and L_{bb} are calculated.

13. Calculate all occurrences of aa, ab, ba, and bb respectively and store them in L_{aa}, L_{ab}, L_{ba}, and L_{bb} respectively.

In instructions 14-20 L_{aubua} is calculated.

14. Calculate all occurrences of $aubua$ in $\boldsymbol{x}$ where $|u| \leq 3\lambda$ and store them in L_{aubua}.

15. Derive new configurations $aubua$ in $\boldsymbol{x}$ from configurations in L'_{aa}, L'_{ab}, and L'_{ba} according to Theorem 4.1 and store them in L_{aubua}.

16. Derive new configurations $aubua$ in $\boldsymbol{x}$ from configurations in L'_{bb} according to Theorem 4.2 and store them in L_{aubua}.

17. Derive new configurations $aubua$ in $\boldsymbol{x}$ from configurations in L'_{aa} and L'_{ba} according to Theorem 4.3 and store them in L_{aubua}.

18. Derive new configurations $aubua$ in $\boldsymbol{x}$ from configurations in L'_{ab} and L'_{bb} according to Theorem 4.4 and store them in L_{aubua}.

19. Derive new configurations $aubua$ in $\boldsymbol{x}$ from configurations in L'_{buaaub} according to Theorem 4.5 and store them in L_{aubua}.

20. Derive new configurations $aubua$ in $\boldsymbol{x}$ from configurations in L'_{aubua} according to Theorem 4.6 and store them in L_{aubua}.

In instructions 21-27 L_{buaub} is calculated.

21. Calculate all occurrences of $buaub$ in $\boldsymbol{x}$ where $|u| \leq 3\lambda$ and store them in L_{buaub}.

22. Derive new configurations $buaub$ in $\boldsymbol{x}$ from configurations in L'_{aa}, L'_{ab}, and L'_{ba} according to Theorem 5.1 and store them in L_{buaub}.

23. Derive new configurations $buaub$ in $\boldsymbol{x}$ from configurations in L'_{bb} according to Theorem 5.2 and store them in L_{buaub}.

24. Derive new configurations $buaub$ in $\boldsymbol{x}$ from configurations in L'_{aa} and L'_{ba} according to Theorem 5.3 and store them in L_{buaub}.

25. Derive new configurations $buaub$ in $\boldsymbol{x}$ from configurations in L'_{ab} and L'_{bb} according to Theorem 5.4 and store them in L_{buaub}.

26. Derive new configurations $buaub$ in $\boldsymbol{x}$ from configurations in L'_{buaaub} according to Theorem 5.5 and store them in L_{buaub}.

27. Derive new configurations $buaub$ in $\boldsymbol{x}$ from configurations in L'_{aubua} according to Theorem 5.6 and store them in L_{buaub}.

In instructions 28-37 L_{buaaub} is calculated.

28. Calculate all occurrences of configurations $buaaub$ in $\boldsymbol{x}$ with $|\boldsymbol{u}| \leq 3\lambda$ and store them in L_{buaaub}.

29. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{aa}, L'_{ab}, L'_{ba}, and L'_{bb} according to Theorem 6.1 and store them in L_{buaaub}.

30. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{bb} according to Theorem 6.2 and store them in L_{buaaub}.

31. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{aa} and L'_{ba} according to Theorem 6.3 and store them in L_{buaaub}.

32. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{ab} and L'_{bb} according to Theorem 6.4 and store them in L_{buaaub}.

33. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{aa} and L'_{ba} according to Theorem 6.5 and store them in L_{buaaub}.

34. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{ab} and L'_{bb} according to Theorem 6.6 and store them in L_{buaaub}.

35. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{buaaub} according to Theorem 6.7 and store them in L_{buaaub}.

36. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{aubua} according to Theorem 6.8 and store them in L_{buaaub}.

37. Derive new configurations $buaaub$ in $\boldsymbol{x}$ from configurations in L'_{aubua} according to Theorem 6.9 and store them in L_{buaaub}.

In instructions 38-39 L_{buau} is calculated.

38. Calculate all occurrences in $\boldsymbol{x}$ of configurations $buau$ with $|\boldsymbol{u}| \leq 3\lambda$ and store them in L_{buau}.

39. Derive new configurations $buau$ in $\boldsymbol{x}$ from configurations in L'_{aubu} according to Theorem 7 and store them in L_{buau}.

In instructions 40-41 L_{aubu} is calculated.

40. Calculate all occurrences in $\boldsymbol{x}$ of configurations $aubu$ with $|\boldsymbol{u}| \leq 3\lambda$ and store them in L_{aubu}.

41. Derive new configurations $aubu$ in $\boldsymbol{x}$ from configurations in L'_{aubu} according to Theorem 8 and store them in L_{aubu}.

42. Return.

Theorem 9 *For any integer $\lambda \geq 1$ there is an integer constant C_λ so that for any two-pattern string $\boldsymbol{x}$ with a scope $\leq \lambda$ and its reduction sequence $\mathcal{S}$ given as input to $\mathcal{REP}_\lambda$, the algorithm $\mathcal{REP}_\lambda$ in $\leq C_\lambda|\boldsymbol{x}|$ steps calculates all output arguments with*

1. $|L_{aa}|$, $|L_{ab}$, $|L_{ba}|$, $|L_{bb}| \leq |\boldsymbol{x}|$;

2. $|L_{aubua}|$, $|L_{buaub}|$, $|L_{buaaub}| \leq |\boldsymbol{x}| \leq (6\lambda+17)(\lambda+1)|\boldsymbol{x}|$;

3. $|L_{aubu}|$, $|L_{buau}| \leq (12\lambda+4)|\boldsymbol{x}|$.

4. $|L_{uu}| \leq ((2^{3\lambda+1} - 2)144\lambda^2 + 8(6\lambda+17)(\lambda+1) + 2(12\lambda+4) + 16)|\boldsymbol{x}|$.

Proof It is clear that $|L_{aa}(\boldsymbol{x})| \leq |\boldsymbol{x}|$, $|L_{ab}(\boldsymbol{x})| \leq |\boldsymbol{x}|$, $|L_{ba}(\boldsymbol{x})| \leq |\boldsymbol{x}|$, and $|L_{bb}(\boldsymbol{x})| \leq |\boldsymbol{x}|$.

L_{aubua} is computed in instructions 14-20 of the algorithm. s denotes the number of $\boldsymbol{p}^i\boldsymbol{q}$ blocks in $\boldsymbol{x}$, while l the number of $\boldsymbol{p}^j\boldsymbol{q}$ blocks. Then $|\boldsymbol{x}| = s(i|\boldsymbol{p}|+|\boldsymbol{q}|) + l(j|\boldsymbol{p}|+|\boldsymbol{q}|)$, while $|\boldsymbol{y}| = s+l$.

As the induction hypothesis assume that both $|L_{aubua}(\boldsymbol{z})|, |L_{buaaub}(\boldsymbol{z})| \leq A|\boldsymbol{z}|$ for any $\boldsymbol{z}$ of size smaller than $\boldsymbol{x}$.

In instruction 14, it can be easily seen that we can compute at most $(3\lambda+2)|\boldsymbol{x}|$ new configurations.

In instruction 15, from each element of each list L'_{aa}, L'_{ab}, and L'_{ba} we can derive at most i new configurations $aubua$, thus from each list we can derive at most $i|\boldsymbol{y}| = is+il \leq |\boldsymbol{x}|$, thus in instruction 15 we can derive at most $3|\boldsymbol{x}|$ new configurations.

In instruction 16, we can derive at most $2j|\boldsymbol{y}| = 2j(s+l) \leq 2|\boldsymbol{x}|$ new configurations.

In instruction 17, we can derive at most $2|\boldsymbol{y}| \leq 2|\boldsymbol{x}|$ new configurations.

In instruction 18, we can derive at most $2|\boldsymbol{y}| \leq 2|\boldsymbol{x}|$ new configurations.

In instruction 19, we can derive at most $|\boldsymbol{q}|A|\boldsymbol{y}| \leq |\boldsymbol{q}|A\frac{|\boldsymbol{x}|}{|\boldsymbol{q}|+1}$ new configurations.

In instruction 20, we can derive at most $|\boldsymbol{p}|A|\boldsymbol{y}| \leq |\boldsymbol{p}|A\frac{|\boldsymbol{x}|}{|\boldsymbol{p}|+1}$ new configurations.

Note, that the conditions for deriving new configurations $aubua$ from elements of L'_{buaaub} (Theorem 4.5) and from elements of L'_{aubua} (Theorem 4.6) are mutually exclusive, so we either derive new configurations in instruction 19 and nothing in instruction 20 or vice versa.

First consider the case when instruction 19 derives the new configurations while instruction 20 does not. Thus $|L_{aubua}(\boldsymbol{x})| \leq (3\lambda+11)|\boldsymbol{x}| + A\frac{|\boldsymbol{q}|}{|\boldsymbol{q}|+1}|\boldsymbol{x}|$. So $|L_{aubua}(\boldsymbol{x})| \leq A|\boldsymbol{x}|$ provided $A|\boldsymbol{x}| \geq (3\lambda+11)|\boldsymbol{x}| + A\frac{|\boldsymbol{q}|}{|\boldsymbol{q}|+1}|\boldsymbol{x}|$, or $A \geq (3\lambda+11)(|\boldsymbol{q}|+1)$, which is satisfied by any $A \geq (3\lambda+11)(\lambda+1)$. For the case when instruction 20 derives the new configurations while instruction 20 does not just replace $|\boldsymbol{q}|$ with $|\boldsymbol{p}|$ to obtain the same result.

L_{buaaub} is computed in instructions 28-37 of the algorithm.

In instruction 28, it can be easily seen that we can compute at most $(6\lambda+4)|\boldsymbol{x}|$ new configurations.

In instruction 29, from each element of each list L'_{aa}, L'_{ab}, L'_{ba}, and L'_{ba} we can derive at most i new configurations $aubua$, thus from each list we can derive at most $i|\boldsymbol{y}| = is+il \leq |\boldsymbol{x}|$, thus in instruction 29 we can derive at most $4|\boldsymbol{x}|$ new configurations.

In instruction 30, we can derive at most $j|\boldsymbol{y}| = j(s+l) \leq |\boldsymbol{x}|$ new configurations.

In instruction 31, we can derive at most $2|\boldsymbol{y}| \leq 2|\boldsymbol{x}|$ new configurations.

In instruction 32, we can derive at most $2|\boldsymbol{y}| \leq 2|\boldsymbol{x}|$ new configurations.

In instruction 33, we can derive at most $2|\boldsymbol{y}| \leq 2|\boldsymbol{x}|$ new configurations.

In instruction 34, we can derive at most $2|\boldsymbol{y}| \leq 2|\boldsymbol{x}|$ new configurations.

In instruction 35, we can derive at most $|\boldsymbol{q}|A|\boldsymbol{y}| \leq |\boldsymbol{q}|A\frac{|\boldsymbol{x}|}{|\boldsymbol{q}|+1}$ new configurations.

In instruction 36, we can derive at most $|\boldsymbol{q}|A|\boldsymbol{y}| \le |\boldsymbol{q}|A\frac{|\boldsymbol{x}|}{|\boldsymbol{q}|+1}$ new configurations.

In instruction 37, we can derive at most $|\boldsymbol{p}|A|\boldsymbol{y}| \le |\boldsymbol{p}|A\frac{|\boldsymbol{x}|}{|\boldsymbol{p}|+1}$ new configurations.

Note, that the conditions for deriving new configurations $buaaub$ from elements of L'_{buaaub} (Theorem 4.7) and from elements of L'_{aubua} (Theorem 4.8 and Theorem 4.9) are mutually exclusive, so we only derive new configurations in exactly one of the instructions 35-37.

First consider the case when either instruction 35 or 36 derives the new configurations. Thus $|L_{buaaub}(\boldsymbol{x})| \le (6\lambda{+}17)|\boldsymbol{x}| + A\frac{|\boldsymbol{q}|}{|\boldsymbol{q}|+1}|\boldsymbol{x}|$. So $|L_{buaaub}(\boldsymbol{x})| \le A|\boldsymbol{x}|$ provided $A|\boldsymbol{x}| \ge (6\lambda{+}17)|\boldsymbol{x}| + A\frac{|\boldsymbol{q}|}{|\boldsymbol{q}|+1}|\boldsymbol{x}|$, or $A \ge (6\lambda{+}17)(|\boldsymbol{q}|{+}1)$, which is satisfied by any $A \ge (6\lambda{+}17)(\lambda{+}1)$. For the case when instruction 37 derives the new configurations just replace $|\boldsymbol{q}|$ with $|\boldsymbol{p}|$ to obtain the same result.

L_{aubu} is computed in instructions 40-41 of the algorithm. As the induction hypothesis assume that $|L_{aubua}(\boldsymbol{z})| \le (12\lambda{+}4)|\boldsymbol{z}|$ for any $\boldsymbol{z}$ of size smaller than $|\boldsymbol{x}|$.

In instruction 40, it can be easily seen that we can compute at most $(6\lambda{+}2)|\boldsymbol{x}|$ new configurations.

In instruction 41, we can derive at most $(12\lambda{+}4)|\boldsymbol{y}| \le (6\lambda{+}2)|\boldsymbol{x}|$ new configurations.

Hence $|L_{aubu}(\boldsymbol{x})| \le 2(6\lambda{+}2)|\boldsymbol{x}| = (12\lambda{+}4)|\boldsymbol{x}|$.

L_{uu} is computed in instructions 4-12 of the algorithm. As the induction hypthesis we assume that $|L_{uu}(\boldsymbol{z})| \le B|\boldsymbol{z}|$ for any $\boldsymbol{z}$ of size smaller that $|\boldsymbol{x}|$.

In instruction 4, according to Theorem 2, we compute at most $(2^{3\lambda+1} - 2)72\lambda^2|\boldsymbol{x}|$ new runs.

In instruction 5, we expand at most $B|\boldsymbol{y}| \le \frac{B}{2}|\boldsymbol{x}|$ new runs.

In each of instructions 6, 7 we derive at most $(6\lambda{+}17)(\lambda{+}1)|\boldsymbol{y}| \le$
$(6\lambda{+}17)(\lambda{+}1)|\boldsymbol{x}|$ new runs.

In instruction 8, we derive at most $(12\lambda{+}4)|\boldsymbol{y}| \le (12\lambda{+}4)|\boldsymbol{x}|$ new runs.

In instruction 9, we derive at most $4j|\boldsymbol{q}||\boldsymbol{y}| \le 4|\boldsymbol{x}|$ new runs.

In instruction 10, we derive at most $|\boldsymbol{q}|(6\lambda{+}17))(\lambda{+}1)|\boldsymbol{y}| \le (6\lambda{+}17)|\boldsymbol{x}|$ new runs.

In instruction 11, we derive at most $4|\boldsymbol{p}||\boldsymbol{y}| \leq 4|\boldsymbol{x}|$ new runs.

In instruction 12, we derive at most $|\boldsymbol{p}|(6\lambda+17)(\lambda+1)|\boldsymbol{y}| \leq (6\lambda+17)|\boldsymbol{x}|$ new runs.

Thus, $|L_{u,u}(\boldsymbol{x})| \leq \frac{B}{2}|\boldsymbol{x}| + (2^{3\lambda+1} - 2)72\lambda^2|\boldsymbol{x}| + 4(6\lambda+17)(\lambda+1)|\boldsymbol{x}| + (12\lambda+4)|\boldsymbol{x}| + 8|\boldsymbol{x}|$, which is true whenever $B \geq (2^{3\lambda+1} - 2)144\lambda^2 + 8(6\lambda+17)(\lambda+1) + 2(12\lambda+4) + 16$.

We have already established or it is easy to see that there is an integer constant D so that all instructions in the algorithm require $\leq D|\boldsymbol{x}|$ steps, with the exception of the recursive call. Thus, we are looking for a constant C_λ such that $C_\lambda|\boldsymbol{x}| \geq D|\boldsymbol{x}| + C_\lambda|\boldsymbol{y}|$. Since $|\boldsymbol{y}| \leq \frac{1}{2}|\boldsymbol{x}|$, any $C_\lambda \geq 2D$ will do.
$\square$

References

[C81] Maxime Crochemore, **An optimal algorithm for computing the repetitions in a word**, *IPL 12-5* (1981) 244-250.

[FKS00] František Franěk, Ayşe Karaman & W. F. Smyth, **Repetitions in Sturmian strings**, *TCS 249-2* (2000) to appear.

[IMS97] C.S.Iliopoulos, Dennis Moore & W. F. Smyth, **A characterization of the squares in a Fibonacci string**, *TCS 172* (1997) 281-291.