Two-Pattern Strings I — A Recognition Algorithm

Frantisek Franek Weilin Lu W. F. Smyth *

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

October 16, 2002

Abstract

This paper introduces a new class of strings on $\{a, b\}$, called ***two-pattern strings***, that constitute a substantial generalization of Sturmian strings while at the same time sharing many of their nice properties. In particular, we show in this paper that two-pattern strings can be recognized in time proportional to their length. This result is a prelude to showing that the repetitions in these substrings can also be computed in linear time, further that two-pattern strings occur in some sense frequently in the class of all strings on $\{a, b\}$.

1 Introduction

In this paper we begin an investigation of the properties of a new class of strings on $\{a, b\}$, derived by the successive action of a sequence of morphisms on the single letter a. All of the strings so determined are finite, and we deal with them from a computational point of view: specifically, we are

*also at School of Computing, Curtin University, Perth WA 6845, Australia, and Department of Computer Science, King's College London.

interested in efficient algorithms to recognize such strings and to compute the repetitions in them; last but not least, we seek to estimate their frequency of occurrence among all strings on $\{a, b\}$.

A previous paper [FKS00] specified linear-time algorithms to recognize and compute repetitions in finite substrings of Sturmian strings; the class of strings discussed here significantly extends this work.

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ denote two distinct nonempty strings on $\{a, b\}$ such that $|\boldsymbol{p}| \leq \lambda$ and $|\boldsymbol{q}| \leq \lambda$, where λ is an integer called the ***scope***. We call $\boldsymbol{p}$ and $\boldsymbol{q}$ ***patterns of scope*** λ. For any pair of finite positive integers i and j such that $i < j$, consider the morphism σ that maps single letters into ***blocks***:

$$a \to \boldsymbol{p}^i\boldsymbol{q}, \quad b \to \boldsymbol{p}^j\boldsymbol{q}. \tag{1}$$

We call σ an ***expansion of scope*** λ and observe that it is specified by a 4-tuple $[\boldsymbol{p}, \boldsymbol{q}, i, j]$. Observe also that an expansion can be applied to any (finite or infinite) string on $\{a, b\}$ to yield an expanded string

$$\boldsymbol{y} = \sigma(\boldsymbol{x}).$$

Given any two expansions σ_1 and σ_2, the composition $\sigma_1 \circ \sigma_2$ is therefore well defined:

$$\boldsymbol{z} = \sigma_1(\boldsymbol{y}) = \sigma_1\big(\sigma_2(\boldsymbol{x})\big) = (\sigma_1 \circ \sigma_2)(\boldsymbol{x}).$$

Definition 1 *Suppose a positive integer λ, a nonnegative integer k, and a finite sequence*

$$\sigma_1, \sigma_2, \ldots, \sigma_k$$

of expansions of scope λ are given, where

$$\sigma_r = [\boldsymbol{p_r}, \boldsymbol{q_r}, i_r, j_r]$$

for every $r = 1, 2, \ldots, k$. Then the string

$$\boldsymbol{x} = (\sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_k)(a)$$

is a ***complete two-pattern string of scope*** *λ if and only if every pair $(\boldsymbol{p_r}, \boldsymbol{q_r})$ of patterns is* ***suitable*** *(defined in Section 2).*

The definition of a suitable pair of patterns is deferred till Section 2 because it is somewhat technical. However, the main idea of a suitable pair is simple: $\boldsymbol{p}$ and $\boldsymbol{q}$ should be dissimilar enough that they can be efficiently distinguished from each other by an algorithm that recognizes complete two-pattern strings. Note that for $k = 0$ in Definition 1, the single letter a (b would do as well) is included as a two-pattern string.

We can easily provide examples of complete two-pattern strings. If we suppose that $\lambda = 3$ and $\sigma_1 = [ab, ba, 2, 3]$, $\sigma_2 = [abb, aa, 1, 4]$, then since the patterns are suitable, all compositions of σ_1 and σ_2 will be two-pattern strings, including in particular:

$$\begin{aligned}
\sigma_1(a) &= (ab)^2ba; \\
(\sigma_1 \circ \sigma_1)(a) &= (ab)^2ba(ab)^3ba(ab)^2ba(ab)^3ba(ab)^3ba(ab)^2ba; \\
(\sigma_1 \circ \sigma_2)(a) &= (ab)^2ba(ab)^3ba(ab)^3ba(ab)^2ba(ab)^2ba; \\
(\sigma_2 \circ \sigma_1)(a) &= (abb)aa(abb)^4aa(abb)aa(abb)^4aa(abb)^4aa(abb)aa.
\end{aligned}$$

Observe further that when the scope $\lambda = 1$, the choice of $\boldsymbol{p}$ and $\boldsymbol{q}$ is restricted to

$$(\boldsymbol{p}, \boldsymbol{q}) = (a, b) \quad \text{or} \quad (\boldsymbol{p}, \boldsymbol{q}) = (b, a). \tag{2}$$

If the further restriction is imposed that $j = i+1$, then all the strings generated by any finite sequence of expansions are finite substrings of Sturmian strings; in fact, in the terminology of [FKS00], these strings are exactly the set of "block-complete" finite substrings of Sturmian strings. (For authoritative information about these much-studied strings, we refer the reader to [L02, Ch. 2] and the references cited therein.) We note that every complete two-pattern string of scope λ is also a complete two-pattern string of scope $\lambda+1$; thus in particular every block-complete finite substring of a Sturmian string is a complete two-pattern string.

As noted above, our initial interest in complete two-pattern strings is computational, following similar studies of Fibonacci [IMS97, FS99, KK00] and Sturmian [BF84, FKS00] strings. We pose two sets of questions:

(Q1) What is the complexity of determining whether or not a given string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a fragment of a complete two-pattern string? Can an efficient algorithm be found to make this determination for every $\boldsymbol{x}$?

(Q2) Given a fragment $\boldsymbol{x}$ of a complete two-pattern string, can an algorithm be found that computes all the repetitions in $\boldsymbol{x}$ in linear time?

Since complete two-pattern strings constitute a much more general class of strings than block-complete Sturmian strings, the following questions also become of interest:

(Q3) What is the frequency of occurrence of fragments $\boldsymbol{x}$ of complete two-pattern strings among all strings on $\{a, b\}$ of length n? What is the asymptotic frequency of occurrence of complete two-pattern strings among all strings on $\{a, b\}$?

In this paper we provide a partial answer to (Q1) by exhibiting an algorithm that in $\Theta(n)$ time determines whether or not a given string $\boldsymbol{x}[1..n]$ is complete two-pattern. Similar to the recognition algorithm in [FKS00], this algorithm outputs the sequence of expansions (1) by which a is transformed into $\boldsymbol{x}$ — or more precisely, the sequence of ***reductions***

$$\boldsymbol{p}^i\boldsymbol{q} \rightarrow a, \quad \boldsymbol{p}^j\boldsymbol{q} \rightarrow b \tag{3}$$

by which $\boldsymbol{x}$ is reduced to a. This sequence provides a complete specification of $\boldsymbol{x}$. Since by (1) each reduction decreases string length by a factor that exceeds

$$i|\boldsymbol{p}|+|\boldsymbol{q}| \geq 2, \tag{4}$$

the recognition algorithm thus yields as a byproduct a potential data compression technique for complete two-pattern strings $\boldsymbol{x}$.

In companion papers [FLS02, FJS01] we are also able to provide partial answers to (Q2) and (Q3). Before going on to discuss (Q1), we pause to provide an introduction and context for these three questions, as well as an outline of the main results.

In dealing with (Q1), we need to cope with the possibility that at any stage of the reduction of $\boldsymbol{x}$, there may be more than one reduction satisfying (3): it then becomes possible that one of these reductions is a part of a sequence that reduces $\boldsymbol{x}$ to a, while another one is not. As long as this possibility exists, any recognition algorithm would be obliged to include provision for backtracking, leading possibly to an execution time exponential in the number of reductions. Our main result in this paper is to show however that backtracking is not required, and that therefore the algorithm that recognizes complete two-pattern strings requires only $\Theta(n)$ time.

Of course this does not yet fully solve (Q1). We conjecture that, just as for Sturmian strings [FKS00], there exists a $\Theta(n)$-time algorithm to determine whether or not a string is a fragment of a complete two-pattern string of scope λ. If such an algorithm were found, it would greatly extend the class of strings that could be efficiently compressed using reductions, or whose repetitions could be efficiently computed.

The view may be taken that interest in (Q2) has been superseded by other work. It has recently become clear that, as a result of research extending over a period of a quarter-century, the repetitions in any string $\boldsymbol{x}[1..n]$ on an ***indexed*** alphabet — that is, an alphabet of size $\alpha \in O(n)$ that maps onto the integers $1..\alpha$ — can be computed in $\Theta(n)$ time. The main steps in this development are as follows:

- an algorithm to compute the suffix tree of $\boldsymbol{x}$ in $\Theta(n)$ time [F97];
- an algorithm to compute the s-factorization of $\boldsymbol{x}$, given the suffix tree of $\boldsymbol{x}$, in $\Theta(n)$ time [LZ76, ZL77];
- the identification of ***maximal periodicities*** or ***runs*** as a suitable encoding of repetitions in strings, and the computation of the leftmost occurrence of every distinct run in $\boldsymbol{x}$ in $\Theta(n)$ time, based on the s-factorization [M89];
- the proof that the number of runs in any string is $O(n)$, and the extension of the algorithm [M89] to compute all occurrences of every run in $\boldsymbol{x}$ in $\Theta(n)$ time, still based on the s-factorization [KK00].

Impressive as this intellectual edifice is, it nevertheless appears, at least in the context of strings on the alphabet $\{a, b\}$, to be rather indirect in its approach, perhaps involving more sophistication than is really required. Indeed, it is not clear that the $\Theta(n)$-time algorithm given in [F97] is preferable in practice to classical $O(n \log n)$-time algorithms for suffix-tree construction. Further, the very long and technical proof that number of runs is linear in string length shows that a constant of proportionality exists, but provides no information about its size; at the same time, computer experiments described in [KK00] provide convincing evidence that the maximum number of runs in any string is less than n, and that this maximum occurs in strings on $\{a, b\}$! Thus, in a sense, the existing theory serves to remind us of how little, rather than how much, we know of periodicity in strings, perhaps especially those on $\{a, b\}$.

In [FLS02] we adopt a more direct approach to (Q2), an extension of the methodology used in [FKS00] for Sturmian strings. Making use of the reduction sequence computed by the recognition algorithm, we show how to compute all the runs in complete two-pattern strings $\boldsymbol{x}[1..n]$ in $\Theta(n)$ time. Essentially, we show that if $\boldsymbol{y}$ is derived from $\boldsymbol{x}$ by a reduction (3), then the nontrivial runs in $\boldsymbol{x}$ can be computed directly from certain special configurations occurring in $\boldsymbol{y}$; thus, over the whole reduction sequence, the runs in $\boldsymbol{x}$ can be computed on a step-by-step basis, from one reduction to the next. It is the special configurations that are of interest here, since they provide insight into the way in which repetitions are formed.

Finally, in [FJS01], we report on progress with (Q3), in estimating the frequency of occurrence of complete two-pattern strings among all strings on $\{a, b\}$. We show that for λ sufficiently large with respect to n, complete two-pattern strings are dense in the set of all strings. We show also that for some values of k and fixed λ, the number of distinct strings of length k (the ***complexity***) can exceed $2k$ — can in fact even be exponential in k.

In this paper we devote Section 2 to proving that for given $\boldsymbol{x}$ there exists a canonical reduction sequence if and only if any reduction sequence exists. Then in Section 3 we present a linear-time algorithm that computes a canonical reduction sequence. Section 4 briefly discusses doubly-infinite two-pattern strings, and we conclude in Section 5 with a remark.

2 A Canonical Reduction

Before proceeding with our development, we need to provide a definition of the term "suitable pair" mentioned in Section 1. Essentially we require that the patterns $\boldsymbol{p}$ and $\boldsymbol{q}$ are sufficiently different from each other that an algorithm reading a two-pattern string is able to distinguish efficiently between them.

Definition 2 *A string $\boldsymbol{q}$ is said to be* **$\boldsymbol{p}$-regular** *if and only if $\boldsymbol{q} = \boldsymbol{upvu}$ for some choice of (possibly empty) substrings $\boldsymbol{u}$ and $\boldsymbol{v}$.*

Observe then that $\boldsymbol{q}$ is certainly $\boldsymbol{p}$-regular if it includes $\boldsymbol{p}$ as a prefix (we just choose $\boldsymbol{u} = \boldsymbol{\varepsilon}$); on the other hand, it is not $\boldsymbol{p}$-regular if it does not include $\boldsymbol{p}$ at all.

Definition 3 *An ordered pair $(\boldsymbol{p}, \boldsymbol{q})$ of nonempty strings is said to be* **suitable** *if and only if*

- *$\boldsymbol{p}$ is* **primitive** *(that is, in our use of the term, $\boldsymbol{p}$ has no nonempty border);*

- *$\boldsymbol{p}$ is not a suffix of $\boldsymbol{q}$;*

- $\boldsymbol{q}$ *is neither a prefix nor a suffix of* $\boldsymbol{p}$*;*
- $\boldsymbol{q}$ *is not* $\boldsymbol{p}$*-regular.*

Using these definitions, we can now show how to reduce a nontrivial complete two-pattern string. Let

$$\begin{aligned}\boldsymbol{x} &= abbabaaabbababbababbababbabaaabbababbabaaabbababbababbababbaba \\ &\quad aabbababbababbababbabaaabbababbababbababbabaaabbababbabaaabbab.\end{aligned}$$

Consider the reduction $\rho_1 = [a, bbab, 1, 3]$ and observe that according to Definition 3, $(a, bbab)$ is a suitable pair. (Observe however that $(a, babb)$ and (a, bab) are *not* suitable pairs!) Then applying the reduction

$$a(bbab) \to a, \quad a^3(bbab) \to b,$$

we find that

$$\boldsymbol{x_1} = \rho_1(\boldsymbol{x}) = abaaababaaabaaabaaabab.$$

Similarly for $\rho_2 = [a, b, 1, 3]$, we find

$$\boldsymbol{x_2} = \rho_2(\boldsymbol{x_1}) = ababbba,$$

while for $\rho_3 = [ab, bba, 2, 3]$,

$$\boldsymbol{x_3} = \rho_3(\boldsymbol{x_2}) = a.$$

Thus $\boldsymbol{x} = \boldsymbol{x}[1..124]$ is completely described by the three-term reduction sequence

$$[a, bbab, 1, 3],\ [a, b, 1, 3],\ [ab, bba, 2, 3],$$

and so is a complete two-pattern string of scope 4.

In the context of possible applications to data compression, it is worth remarking that the number of terms in a reduction sequence is by (4) at most $\lceil \log_2 n \rceil$ and may be much smaller. In our example, $n = 124$ and sequence length $3 = \log_{4.39} 124$.

We turn now to the main object of this section, a proof that in attempting to compute a reduction sequence of scope λ for a given string $\boldsymbol{x}$, we can avoid backtracking. We begin by establishing a restriction on the way in which occurrences of $\boldsymbol{q}$ can overlap with each other. Suppose that a string $\boldsymbol{x}$ is a concatenation of blocks $\boldsymbol{p}^i\boldsymbol{q}$, $\boldsymbol{p}^j\boldsymbol{q}$, where i and j are fixed and $1 \le i < j$. Let us say that an occurrence of $\boldsymbol{q}$ in $\boldsymbol{x}$ is **restrained** if indeed it occurs as a suffix of one of the constituent blocks of $\boldsymbol{x}$; on the other hand, occurrences of $\boldsymbol{q}$ in $\boldsymbol{x}$ need not necessarily occur only at the end of blocks — for such a non-block occurrence, let us say that the occurrence of $\boldsymbol{q}$ is **free**. Further, as a notational convenience, we introduce

$$\boldsymbol{u} \wedge \boldsymbol{v}$$

to mean the substring determined by the overlap

$$\boldsymbol{x}\left[j_1, \min\{i_2, j_2\}\right]$$

of substrings $\boldsymbol{u} = \boldsymbol{x}[i_1..i_2]$ and $\boldsymbol{v} = \boldsymbol{x}[j_1..j_2]$ of $\boldsymbol{x}$, where $i_1 \leq j_1$. Usually $\boldsymbol{x}$ will be clear from the context. For example, if $\boldsymbol{x} = abaabb$, we could write

$$\boldsymbol{x}[2..5] \wedge \boldsymbol{x}[4..6] = ab, \;\; \boldsymbol{x}[2..5] \wedge \boldsymbol{x}[3..4] = aa, \;\; \boldsymbol{x}[2..3] \wedge \boldsymbol{x}[4..5] = \boldsymbol{\varepsilon}.$$

The following lemma will have multiple uses. It will be used initially as a component of the proof of Lemma 2, hence of Theorem 1, then repeatedly to establish properties of the repetitions algorithm described in [FLS02]. The lemma describes positions in $\boldsymbol{x}$ at which a free occurrence of $\boldsymbol{q}$ is impossible — roughly speaking, positions following any occurrence of $\boldsymbol{p}$ as a substring of $\boldsymbol{q}$.

Lemma 1 *Consider a string $\boldsymbol{x}$ that is a concatenation of blocks $\mathbf{p}^i\boldsymbol{q}$, $\mathbf{p}^j\boldsymbol{q}$, $1 \leq i < j$, where the pair $(\boldsymbol{p}, \boldsymbol{q})$ is suitable. Suppose that $\boldsymbol{q}$ contains at least one occurrence of $\boldsymbol{p}$, and so can be expressed in the form $\boldsymbol{upv}$. Let $\boldsymbol{q_2}$ denote a free occurrence of $\boldsymbol{q}$ in $\boldsymbol{x}$ that overlaps with a preceding restrained occurrence $\boldsymbol{q_1}$ of $\boldsymbol{q}$. Then $\boldsymbol{q_1} \wedge \boldsymbol{q_2} \neq \boldsymbol{v}$.*

Proof Observe that since $(\boldsymbol{p}, \boldsymbol{q})$ is suitable, $\boldsymbol{u} \neq \boldsymbol{\varepsilon}$, $\boldsymbol{v} \neq \boldsymbol{\varepsilon}$. Observe further that since $\boldsymbol{p}$ is primitive and not a prefix of $\boldsymbol{q}$, it follows that $\boldsymbol{p}$ is not a prefix of $\boldsymbol{v}$.

Now suppose that the lemma is false, hence that $\boldsymbol{q_1} \wedge \boldsymbol{q_2} = \boldsymbol{v}$. Therefore $\boldsymbol{q_1}$ is followed by another block, hence by an occurrence of $\boldsymbol{p}$, and so $\boldsymbol{q} = \boldsymbol{q_2}$ has prefix $\boldsymbol{vp}$. Thus

$$\boldsymbol{q} = \boldsymbol{vpv'} = \boldsymbol{upv}$$

has border $\boldsymbol{v}$. If $|\boldsymbol{v}'| \geq |\boldsymbol{v}|$, then in fact

$$\boldsymbol{q} = \boldsymbol{vpwv}$$

for some $\boldsymbol{w}$, contradicting the condition that $\boldsymbol{q}$ is not $\boldsymbol{p}$-regular.

Accordingly we conclude that $|\boldsymbol{v}'| < |\boldsymbol{v}|$, so that $\boldsymbol{q} = \boldsymbol{v}^*\boldsymbol{v}'\boldsymbol{p}\boldsymbol{v}'$, where $\boldsymbol{v} = \boldsymbol{v}^*\boldsymbol{v}'$. Since $\boldsymbol{q}$ has border $\boldsymbol{v}$, one of the following three conditions must hold:

(a) $\boldsymbol{v} = \boldsymbol{v}''\boldsymbol{p}\boldsymbol{v}'$ where $\boldsymbol{v}''$ is a nonempty proper suffix of $\boldsymbol{v}$. But then $\boldsymbol{v}''$ is a border of $\boldsymbol{v}$, hence a border of $\boldsymbol{q}$, so that

$$\boldsymbol{q} = \boldsymbol{v}''\boldsymbol{p}\boldsymbol{w}\boldsymbol{v}''$$

for some string $\boldsymbol{w}$, again contradicting the condition that $\boldsymbol{q}$ is not $\boldsymbol{p}$-regular.

(b) $\boldsymbol{v} = \boldsymbol{p}\boldsymbol{v}'$, contradicting the condition that $\boldsymbol{p}$ is not a prefix of $\boldsymbol{v}$.

(c) $\boldsymbol{v} = \boldsymbol{p}'\boldsymbol{v}'$ for some nonempty proper suffix $\boldsymbol{p}'$ of $\boldsymbol{p}$, so that

$$\boldsymbol{q} = (\boldsymbol{p}'\boldsymbol{v}')\boldsymbol{p}\boldsymbol{v}' = (\boldsymbol{p}'\boldsymbol{v}')\boldsymbol{p}^*(\boldsymbol{p}'\boldsymbol{v}')$$

for some nonempty prefix $\boldsymbol{p}^*$ of $\boldsymbol{p}$. But recalling that by hypothesis

$$\boldsymbol{q} = \boldsymbol{u}\boldsymbol{p}\boldsymbol{v} = \boldsymbol{u}\boldsymbol{p}(\boldsymbol{p}'\boldsymbol{v}'),$$

we see that therefore $\boldsymbol{q_2}$ must be preceded by an occurrence of $\boldsymbol{p}$ that has $\boldsymbol{p}^*$ as suffix. Thus $\boldsymbol{p}^*$ is a nonempty border of $\boldsymbol{p}$, which is therefore not primitive, in contradiction to the hypothesis that $\boldsymbol{q}$ is $\boldsymbol{p}$-regular.

We have shown that each case leads to a contradiction, hence that our assumption $\boldsymbol{q_1} \wedge \boldsymbol{q_2} = \boldsymbol{v}$ is false, as required. $\square$

We remark that if no free occurrence of $\boldsymbol{q} = \boldsymbol{u}\boldsymbol{p}\boldsymbol{v}$ overlaps with a restrained occurrence of $\boldsymbol{q}$ in $\boldsymbol{x}$, then it can happen that $\boldsymbol{q}$ has prefix $\boldsymbol{v}$. To see this, consider the example

$$\boldsymbol{x} = \cdots bbbaabbabaabba \cdots\ , \ \boldsymbol{p} = b, \ \boldsymbol{q} = aabba,$$

where $\boldsymbol{q}$ has prefix $\boldsymbol{v} = a$. We remark also that Lemma 1 does not necessarily hold if $\boldsymbol{q_2}$ is a restrained occurrence overlapping with a preceding free occurrence $\boldsymbol{q_1}$: for $\boldsymbol{p} = aab$, $\boldsymbol{q} = bbaabb$, we see that $\boldsymbol{u} = bb$, $\boldsymbol{v} = b$, and the sequence

$$\boldsymbol{x} = \cdots \boldsymbol{q}\boldsymbol{p}\boldsymbol{q_2} = \cdots bbaa\underline{bbaabb}baabb \cdots$$

gives rise to an underlined free occurrence $\boldsymbol{q_1}$ such that $\boldsymbol{q_1} \wedge \boldsymbol{q_2} = \boldsymbol{v}$.

We now extend the use of the terms "restrained" and "free" to blocks. The next lemma essentially tells us that there can be no free occurrences of blocks $\boldsymbol{p}^i\boldsymbol{q}, \boldsymbol{p}^j\boldsymbol{q}$, $1 \leq i < j$, in $\boldsymbol{x}$, when $\boldsymbol{x}$ is a concatenation of these blocks and $(\boldsymbol{p}, \boldsymbol{q})$ is suitable: if a block is identified in $\boldsymbol{x}$, it must be one of the restrained blocks that are concatenated to form $\boldsymbol{x}$.

Lemma 2 *Let $\boldsymbol{x}$ be a concatenation of blocks $\boldsymbol{p}^i\boldsymbol{q}$ and $\boldsymbol{p}^j\boldsymbol{q}$, $1 \leq i < j$, where $(\boldsymbol{p}, \boldsymbol{q})$ is suitable. Let $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ be occurrences of blocks such that $\boldsymbol{B_1} \wedge \boldsymbol{B_2} \neq \boldsymbol{\varepsilon}$. Then $\boldsymbol{B_2}$ is a suffix of $\boldsymbol{B_1}$.*

Proof We consider first the case in which $\boldsymbol{B_1}$ is restrained and $\boldsymbol{B_2}$ free.

Let $\boldsymbol{B_1} = \boldsymbol{p_1}^r\boldsymbol{q_1}$, $\boldsymbol{B_2} = \boldsymbol{p_2}^s\boldsymbol{q_2}$, where $\boldsymbol{p} = \boldsymbol{p_1} = \boldsymbol{p_2}$ and $\boldsymbol{q} = \boldsymbol{q_1} = \boldsymbol{q_2}$. There are two cases to consider:

(a) $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ start at the same position in $\boldsymbol{x}$.

Suppose without loss of generality that $|\boldsymbol{B_1}| < |\boldsymbol{B_2}|$. Then either $\boldsymbol{p_2}$ is a prefix of $\boldsymbol{q_1}$ or $\boldsymbol{q_1}$ is a prefix of $\boldsymbol{p_2}$, both contradicting the suitability of $(\boldsymbol{p}, \boldsymbol{q})$. Hence $|\boldsymbol{B_1}| = |\boldsymbol{B_2}|$ and so $\boldsymbol{B_1} = \boldsymbol{B_2}$.

(b) $\boldsymbol{B_2}$ starts to the right of $\boldsymbol{B_1}$.

(i) Suppose the prefix $\boldsymbol{p_2}^s$ of $\boldsymbol{B_2}$ is a substring of the prefix $\boldsymbol{p_1}^r$ of $\boldsymbol{B_1}$. Since $\boldsymbol{p}$ is primitive, the occurrences of $\boldsymbol{p_2}$ must be aligned with those of $\boldsymbol{p_1}$. Then if $\boldsymbol{p_2}^s$ is a suffix of $\boldsymbol{p_1}^r$, it follows that $\boldsymbol{B_2}$ is a suffix of $\boldsymbol{B_1}$, as required; otherwise, as in (a), either $\boldsymbol{p_1}$ is a prefix of $\boldsymbol{q_2}$ or $\boldsymbol{q_2}$ is a prefix of $\boldsymbol{p_1}$, both impossible.

(ii) Suppose the prefix $\boldsymbol{p_2}^s$ has a nonempty overlap with $\boldsymbol{p_1}^r$ but is not a substring of it. Again the occurrences of $\boldsymbol{p_2}$ must align with those of $\boldsymbol{p_1}$, and again there is a contradiction: either $\boldsymbol{p_2}$ is a prefix of $\boldsymbol{q_1}$ or $\boldsymbol{q_1}$ is a prefix of $\boldsymbol{p_2}$.

(iii) Suppose $\boldsymbol{p_2}^s$ does not overlap with $\boldsymbol{p_1}^r$. Of course $\boldsymbol{p_2}^s$ cannot be a suffix or a prefix of $\boldsymbol{q_1}$. If it is a proper substring of $\boldsymbol{q_1}$, then $\boldsymbol{q_1} = \boldsymbol{u}\boldsymbol{p_2}^s\boldsymbol{v}$ for nonempty strings $\boldsymbol{u}$ and $\boldsymbol{v}$, where moreover $\boldsymbol{v}$ must be a prefix of $\boldsymbol{q_2}$. Since $\boldsymbol{q_1}$ is restrained and $\boldsymbol{q_2}$ is free, this contradicts Lemma 1. The only remaining possibility is that $\boldsymbol{p_2}^s$ overlaps with $\boldsymbol{q_1}$; but in this case there must be an occurrence of $\boldsymbol{p}$ to the right of $\boldsymbol{q_1}$ (since $\boldsymbol{q_1}$ is restrained), and this occurrence must align with an occurrence of $\boldsymbol{p_2}$. This implies that an occurrence of $\boldsymbol{p_2}$ is a suffix of $\boldsymbol{q_1}$, again contradicting suitability.

Thus the result is true in the case that $\boldsymbol{B_1}$ is restrained and $\boldsymbol{B_2}$ free. For $\boldsymbol{B_1}$ free and $\boldsymbol{B_2}$ restrained, cases (a) and (b) again arise as above. But the lemma still holds in case (a), and in case (b) there must exist a restrained $\boldsymbol{B_2}'$ preceding $\boldsymbol{B_2}$ such that $\boldsymbol{B_2}' \wedge \boldsymbol{B_1}$ is nonempty and not a suffix of $\boldsymbol{B_2}'$, the situation just shown to be impossible. We conclude that the lemma holds for all $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$. $\Box$

We introduce the notation $\rho = [\boldsymbol{p}, \boldsymbol{q}, i, j]_\lambda$ to specify a reduction for which the pair $(\boldsymbol{p}, \boldsymbol{q})$ is suitable, with $|\boldsymbol{p}|, |\boldsymbol{q}| \leq \lambda$.

Lemma 3 *For any $\lambda \geq 1$, any $r \geq 2$, and any nonempty string $\boldsymbol{u}$, $\boldsymbol{u}^r$ is not a complete two-pattern string of scope λ.*

Proof Observe first that if $|\boldsymbol{u}| = 1$, then $\boldsymbol{u}^r$ is not a complete two-pattern string. Suppose then that $|\boldsymbol{u}| > 1$, and suppose further that $\boldsymbol{u}^r$ is a complete two-pattern string, so that at least one reduction $[\boldsymbol{p}, \boldsymbol{q}, i, j]_\lambda$ can be performed on it. Observe that $\boldsymbol{p}$ must be a prefix and $\boldsymbol{q}$ a suffix of $\boldsymbol{u}^r$.

If $\boldsymbol{u}$ is a proper prefix of $\boldsymbol{p}^h$ for some $h \geq 1$, then $\boldsymbol{p}$ must have a nonempty border, an impossibility since $(\boldsymbol{p}, \boldsymbol{q})$ is suitable. If on the other hand $\boldsymbol{u} = \boldsymbol{p}^h$ for some $h \geq 1$, then either $\boldsymbol{p}$ is a suffix of $\boldsymbol{q}$ or $\boldsymbol{q}$ a suffix of $\boldsymbol{p}$, again contradicting suitability. Hence $\boldsymbol{u} = \boldsymbol{p}^h\boldsymbol{v}$ for some $h \geq i \geq 1$ and some nonempty $\boldsymbol{v}$, where either $\boldsymbol{v}$ is a prefix of $\boldsymbol{q}$ or $\boldsymbol{q}$ is a prefix of $\boldsymbol{v}$.

Observe that an occurrence $\boldsymbol{q_1}$ of $\boldsymbol{q}$ must coincide with the leftmost occurrence of $\boldsymbol{v}$ in $\boldsymbol{u}^r$. The right end of $\boldsymbol{q_1}$ cannot occur in any following occurrence of $\boldsymbol{p}^h$, because then either $\boldsymbol{q_1}$ would have suffix $\boldsymbol{p}$, or else the following occurrence of $\boldsymbol{p}$ would have a border, contradicting primitivity. Nor can $\boldsymbol{q_1}$ end in the interior of any following occurrence of $\boldsymbol{v}$, because then it would be followed by an occurrence of $\boldsymbol{p}$ and so would take the form $\boldsymbol{wpw'p^hw}$ for some nonempty strings $\boldsymbol{w}$ and $\boldsymbol{w'}$, forcing $\boldsymbol{q}$ to be $\boldsymbol{p}$-regular. Similarly $\boldsymbol{q}$ cannot take the $\boldsymbol{p}$-regular form $\boldsymbol{vp^hv}$. We conclude therefore that the first occurrence of $\boldsymbol{q}$, hence every occurrence of $\boldsymbol{q}$, is a border of $\boldsymbol{v}$. Thus, in view of Lemma 1,

$$\boldsymbol{u}^r = \left(\boldsymbol{p}^{i_1}\boldsymbol{q}\boldsymbol{p}^{i_2}\boldsymbol{q}\cdots\boldsymbol{p}^{i_k}\boldsymbol{q}\right)^r.$$

Since $\boldsymbol{u}^r$ is assumed to be two-pattern, we can perform a reduction on it, yielding $\rho(\boldsymbol{u}^r) = \big(\rho(\boldsymbol{u})\big)^r$, where $\big|\rho(\boldsymbol{u}^r)\big| < |\boldsymbol{u}^r|/2$. Then if $\big|\rho(\boldsymbol{u}\big| > 1$, we can repeat the above argument, substituting $\rho(\boldsymbol{u})$ for $\boldsymbol{u}$. Thus after a finite number of such reductions, we must reduce to the non-two-pattern form a^r, and so we conclude by induction that $\boldsymbol{u}^r$ cannot be a complete two-pattern string. $\Box$

We are now in a position to propose a strategy for choosing a reduction sequence of a two-pattern string that does not require backtracking (see the discussion of (Q1) in Section 1). The phenomenon we have to deal with involves strings $\boldsymbol{x}$ that can be λ-reduced in two distinct ways:

$$\rho_1 = [\boldsymbol{p_1}, \boldsymbol{q_1}, i_1, j_1]_\lambda \neq \rho_2 = [\boldsymbol{p_2}, \boldsymbol{q_2}, i_2, j_2]_\lambda.$$

For example, $\boldsymbol{x} = babaabb$ can be 5-reduced to a using either $\rho_1 = [ba, abb, 2, *]$ or $\rho_2 = [babaa, bb, 1, *]$.

For ρ_1 and ρ_2 to be distinct, we may suppose without loss of generality that either $|\boldsymbol{p_1}| < |\boldsymbol{p_2}|$ or that $|\boldsymbol{p_1}| = |\boldsymbol{p_2}|$ with $|\boldsymbol{q_1}| < |\boldsymbol{q_2}|$. But we argue that the second case is impossible, since it requires that $\boldsymbol{p_1} = \boldsymbol{p_2} = \boldsymbol{p}$, say, hence that the first block in each reduction begins with $\boldsymbol{p}^r$, $r \geq 1$. But the $\boldsymbol{p_1}$-regularity of $\boldsymbol{q_1}$ requires that neither of $\boldsymbol{p_1}, \boldsymbol{q_1}$ can be a prefix of the other, and so it follows that in the leftmost ρ_2-block, $\boldsymbol{q_1}$ is a proper prefix of $\boldsymbol{q_2}$. But $\boldsymbol{q_2}$ must be followed by an occurrence of $\boldsymbol{p}$ and can therefore not terminate in any subsequent occurrence of $\boldsymbol{p}$ without violating the primitivity of $\boldsymbol{p}$. Thus $\boldsymbol{q_2}$ must terminate in an occurrence of $\boldsymbol{q_1}$, and in fact, since $\boldsymbol{q_1}$ is a suffix of $\boldsymbol{q_2}$, must therefore take the form $\boldsymbol{q_1 p w q_1}$. Thus $\boldsymbol{q_2}$ is $\boldsymbol{p_2}$-regular, a contradiction, and so we need consider only the case $|\boldsymbol{p_1}| < |\boldsymbol{p_2}|$. Since our strategy will be always to choose the reduction with the shortest $\boldsymbol{p}$-pattern, we state our main definition as follows:

Definition 4 *A reduction $\rho_1 = [\boldsymbol{p_1}, \boldsymbol{q_1}, i_1, j_1]_\lambda$ of a binary string $\boldsymbol{x}$ is* **$\boldsymbol{\lambda}$-canonical** (**canonical** *for short when no confusion arises*) *if and only if for every other reduction $\rho_2 = [\boldsymbol{p_2}, \boldsymbol{q_2}, i_2, j_2]_\lambda \neq \rho_1$ of $\boldsymbol{x}$, $|\boldsymbol{p_2}| > |\boldsymbol{p_1}|$.*

Note that a λ-canonical reduction of $\boldsymbol{x}$ exists whenever any reduction of scope λ exists, and that when it exists, it is unique. For a given reduction $\rho = [\boldsymbol{p}, \boldsymbol{q}, i, j]_\lambda$, we use the generic term **ρ-block** to describe either the **short block $\boldsymbol{p}^i\boldsymbol{q}$** or the **long block $\boldsymbol{p}^j\boldsymbol{q}$**. Further, we say that any substring $\boldsymbol{u}$ of $\boldsymbol{x}$ is **ρ-faithful** if $\boldsymbol{u}$ begins and ends with a ρ-block — thus by Lemma 2 a

ρ-faithful substring in a complete two-pattern string $\boldsymbol{x}$ is necessarily a concatenation of ρ-blocks. The next lemma establishes properties of ρ-faithful substrings that will be required for the proof of the main result of this section.

Lemma 4 *Suppose $\boldsymbol{x} \neq a$ is a complete two-pattern string of scope λ. Let $\rho_1 = [\boldsymbol{p_1}, \boldsymbol{q_1}, i_1, j_1]_\lambda$ be the λ-canonical reduction of $\boldsymbol{x}$ and $\rho_2 = [\boldsymbol{p_2}, \boldsymbol{q_2}, i_2, j_2]_\lambda$, $|\boldsymbol{p_2}| > |\boldsymbol{p_1}|$, another reduction of $\boldsymbol{x}$. If $\boldsymbol{x} \neq \boldsymbol{p_2 q_2}$, then*

(a) *no ρ_1-block is ρ_2-faithful;*

(b) if a ρ_2-block is ρ_1-faithful, then both $\boldsymbol{p_2}$ and $\boldsymbol{q_2}$ are ρ_1-faithful.

Proof Notice first that since by Lemma 3 $\boldsymbol{x}$ cannot be a repetition, $\boldsymbol{x} = \boldsymbol{p_2 q_2}$ and $\boldsymbol{x} = \cdots \boldsymbol{p_2}^2 \cdots$ are mutually exclusive conditions: $\boldsymbol{x} = \boldsymbol{p_2 q_2}$ if and only if $\boldsymbol{p_2}^2$ occurs nowhere in $\boldsymbol{x}$. We may assume therefore that $\boldsymbol{p_2}^2$ is a substring of $\boldsymbol{x}$, further that both long and short blocks occur both for ρ_1 and ρ_2.

To prove (a), suppose that there exists a ρ_2-faithful ρ_1-block $\boldsymbol{C}$ in $\boldsymbol{x}$. If $\boldsymbol{C}$ is short, then every short ρ_1-block is ρ_2-faithful, including the short ρ_1-blocks that are suffixes of every long ρ_1-block. We conclude that the prefixes $\boldsymbol{p_1}^{j_1-i_1}$ of every long ρ_1-block are also ρ_2-faithful. On the other hand, if $\boldsymbol{C}$ is long, then every long ρ_1-block is ρ_2-faithful, so that the intervening sequences of short ρ_1-blocks, say $(\boldsymbol{p_1}^{i_1}\boldsymbol{q_1})^r$, $r \geq 1$, must also be ρ_2-faithful. However, this implies that *each* of these short ρ_1-blocks must individually be ρ_2-faithful; assuming the contrary leads to the conclusion that the ρ_2-blocks would overlap, in contradiction to Lemma 2.

In both cases, then, we conclude that all the ρ_1-blocks in $\boldsymbol{x}$ are ρ_2-faithful, hence that the nonempty substrings $\boldsymbol{P_1} = \boldsymbol{p_1}^{j_1-i_1}$ are also ρ_2-faithful. Then $\boldsymbol{P_1}$ begins with an occurrence of $\boldsymbol{p_2}$. But recall that $|\boldsymbol{p_2}| > |\boldsymbol{p_1}|$, so that $j_1 - i_1 > 1$, implying that $\boldsymbol{p_2}$ has some nonempty prefix of $\boldsymbol{p_1}^{j_1-i_1-1}$ as a border, and so cannot be primitive. We conclude that there exists no ρ_2-faithful ρ_1-block in $\boldsymbol{x}$.

To prove (b), we assume that there exists a ρ_1-faithful ρ_2-block in $\boldsymbol{x}$; it follows then as in the proof of (a) that every occurrence of $\boldsymbol{P_2} = \boldsymbol{p_2}^{j_2-i_2}$ must be ρ_1-faithful. Therefore $\boldsymbol{q_1}$ is a suffix of $\boldsymbol{P_2}$. If we suppose that $|\boldsymbol{q_1}| \geq |\boldsymbol{p_2}|$, then $\boldsymbol{p_2}$ is a suffix of $\boldsymbol{q_1}$, hence a suffix of $\boldsymbol{x}$, so that one of $\boldsymbol{p_2}, \boldsymbol{q_2}$ is a suffix of the other, contradicting the conditions for a suitable pair. We conclude that $|\boldsymbol{q_1}| < |\boldsymbol{p_2}|$, hence further that $\boldsymbol{q_1}$ is a proper suffix of every occurrence of $\boldsymbol{p_2}$, which also has $\boldsymbol{p_1}$ as a proper prefix. Applying Lemma 2, we conclude that $\boldsymbol{p_2}$ must be ρ_1-faithful, hence that $\boldsymbol{q_2}$ is ρ_1-faithful also. $\square$

The next lemma provides the justification for always choosing the λ-canonical reduction at each algorithmic step.

Lemma 5 *Suppose $\boldsymbol{x} \neq a$ is a complete two-pattern string of scope λ. Let $\rho_1 = [\boldsymbol{p_1}, \boldsymbol{q_1}, i_1, j_1]_\lambda$ be the λ-canonical reduction of $\boldsymbol{x}$ and $\rho_2 = [\boldsymbol{p_2}, \boldsymbol{q_2}, i_2, j_2]_\lambda$, $|\boldsymbol{p_2}| > |\boldsymbol{p_1}|$, another reduction of $\boldsymbol{x}$. Then*

(a) *if $\boldsymbol{p_2}^2$ is a substring of $\boldsymbol{x}$, $\boldsymbol{p_2}$ and $\boldsymbol{q_2}$ are both ρ_1-faithful;*

(b) *otherwise, $\boldsymbol{x} = \boldsymbol{p_2}\boldsymbol{q_2}$ is ρ_1-faithful.*

Proof Of course (b) is immediate, and therefore we focus on (a), so that the conditions of Lemma 4 hold. Essentially, we show that there must exist a ρ_1-faithful ρ_2-block in $\boldsymbol{x}$, from which, by virtue of Lemma 4(b), (a) follows.

Let $\boldsymbol{C}$ and $\boldsymbol{C}^*$ denote the leftmost and rightmost canonical ρ_1-blocks, respectively, in $\boldsymbol{x}$, and let $\boldsymbol{B}$ and $\boldsymbol{B}^*$ denote the leftmost and rightmost ρ_2-blocks in $\boldsymbol{x}$. We deal with two main cases:

(1) $|\boldsymbol{C}| \leq |\boldsymbol{B}|$

Then since both $\boldsymbol{C}$ and $\boldsymbol{B}$ are prefixes of $\boldsymbol{x}$, $\boldsymbol{C}$ is therefore a prefix of $\boldsymbol{B}$ and we need to consider two subcases:

(i) $|\boldsymbol{B}| \leq |\boldsymbol{B}^*|$

Here by Theorem 2 $\boldsymbol{B}$ is a suffix of $\boldsymbol{B}^*$, hence of $\boldsymbol{x}$, and two further subsubcases arise. If $|\boldsymbol{C}^*| \leq |\boldsymbol{B}|$, $\boldsymbol{B}$ has two ρ_1-blocks, $\boldsymbol{C}$ and $\boldsymbol{C}^*$, as prefix and suffix, respectively, and so is ρ_1-faithful, the required result. On the other hand, if $|\boldsymbol{C}^*| > |\boldsymbol{B}|$, $\boldsymbol{B}$ must be a suffix of $\boldsymbol{C}^*$. But since $\boldsymbol{C}$ is a prefix of $\boldsymbol{B}$, it follows that $\boldsymbol{C}^*$ contains $\boldsymbol{C}$, possible by Lemma 2 only if $\boldsymbol{C}$ is a suffix of $\boldsymbol{C}^*$, hence also of $\boldsymbol{B}$. This implies that $\boldsymbol{C} = \boldsymbol{B}$, and so $\boldsymbol{C}$ is ρ_2-faithful, impossible by Lemma 4(a).

(ii) $|\boldsymbol{B}| > |\boldsymbol{B}^*|$

In this case $\boldsymbol{B}^*$ is a proper suffix of $\boldsymbol{B}$. If $|\boldsymbol{C}^*| \leq |\boldsymbol{B}^*|$, then $\boldsymbol{B}$ has prefix $\boldsymbol{C}$ and suffix $\boldsymbol{B}^*$, that in turn has suffix $\boldsymbol{C}^*$; thus $\boldsymbol{B}$ is ρ_1-faithful, as required.

Suppose then that $|\boldsymbol{C}^*| > |\boldsymbol{B}^*|$. We need to consider two final cases. If $|\boldsymbol{C}| \leq |\boldsymbol{C}^*|$, then either

* $\boldsymbol{C}$ is a suffix of $\boldsymbol{B}^*$, hence a suffix of $\boldsymbol{B}$, which implies, since $\boldsymbol{C}$ is also a prefix of $\boldsymbol{B}$, that $\boldsymbol{B}$ is ρ_1-faithful; or
* $\boldsymbol{B}^*$ is a suffix of $\boldsymbol{C}$, which is a prefix of $\boldsymbol{B}$, so that $\boldsymbol{B}^*$ is a substring of $\boldsymbol{B}$, thus by Lemma 2 a suffix of $\boldsymbol{B}$, implying that $\boldsymbol{C} = \boldsymbol{B}$ and that $\boldsymbol{C}$ is therefore ρ_2-faithful, an impossibility by Lemma 4(a).

On the other hand, if $|\boldsymbol{C}| > |\boldsymbol{C}^*|$, then by Lemma 2 $\boldsymbol{C}^*$ is a suffix of $\boldsymbol{C}$, hence a substring of $\boldsymbol{B}$, implying that $\boldsymbol{B}^*$ also is a substring of $\boldsymbol{B}$, and so again we must conclude that $\boldsymbol{C} = \boldsymbol{B}$, as we have seen an impossibility.

(2) $|\boldsymbol{C}| > |\boldsymbol{B}|$

In this case $\boldsymbol{B}$ is a proper prefix of $\boldsymbol{C}$, and we discover after another detailed breakdown into subcases that in fact $|\boldsymbol{C}| > |\boldsymbol{B}|$ is impossible. We omit the details.

Thus we claim that in every possible case that arises, there must exist a ρ_1-faithful ρ_2-block, and so (a) follows from Lemma 4(b). $\square$

Since under the conditions of Lemma 5(a), both $\boldsymbol{p_2}$ and $\boldsymbol{q_2}$ are ρ_1-faithful, it follows that there exists a sequence of λ-canonical reductions that carries a reduction by ρ_1 into a reduction by ρ_2. When the conditions of Lemma 5(b) hold, a single λ-canonical reduction reduces $\boldsymbol{x}$ to a. Thus a sequence of λ-canonical reductions of a two-pattern string achieves the same result as any sequence of reductions. We have therefore

Theorem 1 *$\boldsymbol{x}$ is a complete two-pattern string of scope λ if and only if there is a sequence of λ-canonical reductions $\{\rho_1, \rho_2, \cdots, \rho_n\}$ reducing $\boldsymbol{x}$ to a string a.*

It is worth remarking that in Lemmas 4 and 5 the assumption that $\boldsymbol{p_1}$ is the *least* $\boldsymbol{p}$-pattern is not used; it is required only that there exist two $\boldsymbol{p}$-patterns $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ with $|\boldsymbol{p_1}| < |\boldsymbol{p_2}|$. Thus we are assured that whenever two distinct patterns are possible, either one may be chosen.

$twopat \leftarrow$ `TRUE`; $k \leftarrow 0$;
while $\boldsymbol{x} \neq a$ **and** $twopat$ **do**
  $recognize(\lambda, \boldsymbol{x}; \boldsymbol{p}, \boldsymbol{q}, i, j)$
  if $i+j = 0$ **then**
    $twopat \leftarrow$ `FALSE`
  else
    $k \leftarrow k+1$; $R[k] \leftarrow \{\boldsymbol{p}, \boldsymbol{q}, i, j\}$
    $reduce(\boldsymbol{x}, \boldsymbol{y}, |\boldsymbol{p}|, |\boldsymbol{q}|, i, j)$
    $\boldsymbol{x} \leftarrow \boldsymbol{y}$

Figure 1: Compute a reduction sequence R of $\boldsymbol{x}$

3 The Recognition Algorithm

Theorem 1 essentially tells us that we can choose *any* pattern $\boldsymbol{p}$ that gives rise to a suitable pair $(\boldsymbol{p}, \boldsymbol{q})$ in order to perform a reduction on a given string $\boldsymbol{x} = \boldsymbol{x}[1..n]$. If in fact $\boldsymbol{x}$ is a complete two-pattern string, we can reduce using $(\boldsymbol{p}, \boldsymbol{q})$ and be assured that there exists a subsequent sequence of reductions that will ultimately reduce $\boldsymbol{x}$ to a. Thus no backtracking is required and, as shown in Figure 1, the step-by-step reduction process is straightforward, requiring at each step only a single call to a recognition procedure, called *recognize*, that, given λ and $\boldsymbol{x}$, computes if possible a reduction $[\boldsymbol{p}, \boldsymbol{q}, i, j]_\lambda$ and if not returns $i = j = 0$.

However, as shown in Figure 2, the *recognize* procedure is still not entirely straightforward. It consists of a main loop that considers one-by-one candidate $\boldsymbol{p}$-patterns of lengths $1..\lambda$, with an interior loop that for each $\boldsymbol{p}$ considers candidate $\boldsymbol{q}$-patterns. Observe that since the main loop is in increasing order of length of $\boldsymbol{p}$, therefore any reduction identified will be canonical; however, decreasing order of length could have been used just as well, and in fact, in cases where more than one reduction was possible, would output fewer reductions, hence require fewer steps.

The main loop first computes all the blocks of $\boldsymbol{p}$'s in $\boldsymbol{x}$, identifying the length of each following gap. The first such block must be a prefix of the first ρ-block in $\boldsymbol{x}$, hence restrained, but in general the $\boldsymbol{p}$-blocks may be either free or restrained. The possible $\boldsymbol{q}$-patterns in $\boldsymbol{x}$ must be chosen from among $\boldsymbol{x}[m_1..n]$ and its borders, subject to lower and upper bounds on length; for each feasible $\boldsymbol{q}$, the gap array G is inspected to see if the $\boldsymbol{p}$ and $\boldsymbol{q}$ blocks alternate in the required fashion. Finally, if all other conditions are satisfied,

procedure *recognize*$(\lambda, \boldsymbol{x}; \boldsymbol{p}, \boldsymbol{q}, i, j)$
$i, j \leftarrow 0$
for $\ell \leftarrow 1$ **to** λ
  $\boldsymbol{p} \leftarrow \boldsymbol{x}[1..\ell]$
  if $\boldsymbol{p} = \boldsymbol{x}[n-\ell+1..n]$ **or** $\boldsymbol{p}$ not primitive **then next** ℓ
   — *Compute all occurrences h of $\boldsymbol{p}$ in $\boldsymbol{x}$ in ascending*
   — *order: since $\boldsymbol{p}$ is primitive, there are no overlaps.*
  $H = <h_1, h_2, \ldots, h_r>$
   — *From H compute occurrences m of $\boldsymbol{p}$-blocks in ascending*
   — *order of m and lengths g of the following $\boldsymbol{p}$-free gaps.*
  $G = <(m_1, g_1), (m_2, g_2), \ldots, (m_s, g_s)>$
   — *Compute the exponent of the first $\boldsymbol{p}$-block.*
  $\delta \leftarrow (m_1-1)/\ell \qquad (\delta = i \text{ or } j)$
   — *Compute the minimum length $\pi \geq \ell$ of a $\boldsymbol{p}$-block in G.*
  $\pi \leftarrow \min_{2 \leq t \leq s}\{m_1-1, m_t-m_{t-1}-g_{t-1}\}$
   — *Arrange the lengths β of $\boldsymbol{x}[m_1..n]$ and all its nonempty*
   — *borders in descending sequence; then choose those that*
   — *satisfy* $\max\{g_1, g_s\} \leq \beta \leq \lambda$.
  $B = <\beta_1, \beta_2, \ldots, \beta_k>$
  for $\ell' \leftarrow 1$ **to** k **do**
     — *Consider candidate $\boldsymbol{q}$-patterns.*
    $\boldsymbol{q} \leftarrow \boldsymbol{x}[m_1..m_1+\beta_{\ell'}-1]$
    if $\boldsymbol{q}$ satisfies
    (1) each occurrence falls at the end of a gap:
       $\boldsymbol{q} = \{\boldsymbol{x}[m_1..m_{t_1}+g_{t_1}-1],\ \boldsymbol{x}[m_{t_1+1}..m_{t_2}+g_{t_2}-1], \ldots\}$
    (2) each restrained $\boldsymbol{p}$-block has δ or δ' occurrences of $\boldsymbol{p}$:
       $(m_{t_1+1}-m_{t_1}-g_{t_1})/\ell,\ (m_{t_2+1}-m_{t_2}-g_{t_2})/\ell, \ldots$
       and both must exist for $\beta_{\ell'} \neq n-m_1+1$
    (3) (for $\beta_{k'} > \pi$) $\boldsymbol{q}$ does not take the $\boldsymbol{p}$-regular form $\boldsymbol{upvu}$
    then return $(\boldsymbol{p}, \boldsymbol{q}, \delta, \delta')$

Figure 2: Compute if possible a single reduction of $\boldsymbol{x}$

$\boldsymbol{q}$ is tested for $\boldsymbol{p}$-regularity.

The primitivity of $\boldsymbol{p}$ can be determined in $O(\lambda)$ time using a standard border array calculation [AHU74]. The array H can be computed by a standard pattern-matching algorithm such as Boyer-Moore [BM77] or one of its descendants in $O(n)$ time; similarly G and π can be computed in $O(n)$ time, as can the borders of $\boldsymbol{x}[m_1..n]$. In the interior loop, each of the three conditions on $\boldsymbol{p}$ can be evaluated in $O(n)$ time, and since each **for** loop is executed at most λ times, the overall time requirement of *recognize* is $O\big(\lambda(n+\lambda n)\big)$. Since the application of a reduction reduces the length of $\boldsymbol{x}$ by at least a factor of two, it follows that the asymptotic time requirement for computing the entire reduction sequence of $\boldsymbol{x}$ is no greater than it is for a single reduction. Hence:

Theorem 2 *It requires $O(\lambda^2 n)$ time to determine whether or not a given string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a complete two-pattern string of scope λ, and, if it is, to compute its reduction sequence.*

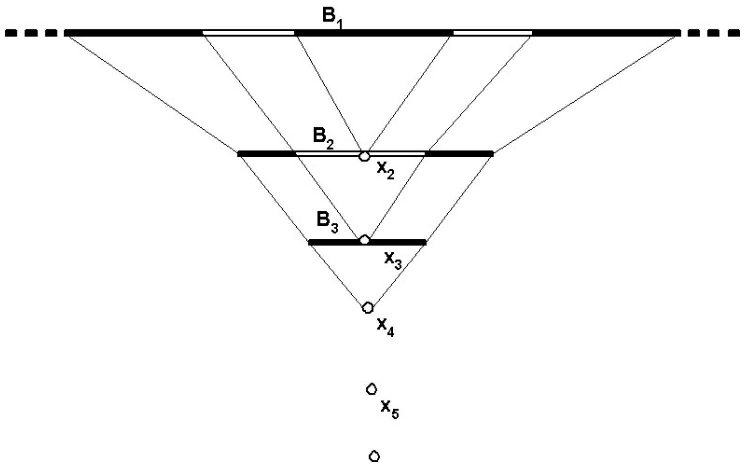
4 Infinite Two-Pattern Strings

Let us remark that two-pattern strings could be defined as doubly infinite strings (that is, indexed by the integers) and studied as mathematical objects. Traditionally, Sturmian strings are defined as infinite strings (so indexed by the natural numbers) that satisfy two properties: (a) not ultimately periodic and (b) balanced. An infinite string $\boldsymbol{S}$ is ***ultimately periodic*** if there are finite strings $\boldsymbol{u}$ and $\boldsymbol{v}$ such that for any finite prefix $\boldsymbol{S_1}$ of $\boldsymbol{S}$, there is an integer $k \geq 1$ such that $\boldsymbol{S_1}$ is a prefix of $\boldsymbol{uv}^k$; $\boldsymbol{S}$ is ***balanced*** if for any finite substrings $\boldsymbol{u}, \boldsymbol{v}$ of $\boldsymbol{S}$ for which $|\boldsymbol{u}| = |\boldsymbol{v}|$, it follows that $||\boldsymbol{u}|_a - |\boldsymbol{v}|_a| \leq 1$ (where $|\boldsymbol{u}|_a$ is the number of occurrences of the letter a in the string $\boldsymbol{u}$). It is not hard to see that the Fibonacci string satisfies these properties and hence is a Sturmian string. In [FKS00] a Sturmian string is presented as a concatenation of blocks $a^i b$ and $a^{i+1}b$ for some $i \geq 1$ (for the Fibonacci string $i = 1$, so it is in a sense the "simplest" Sturmian string), with the possible exception of the very first block that can be of the form $a^r b$ with $0 \leq r \leq i{+}1$ (and of course, a and b can interchange their roles). Furthermore, if a reduction $a^i b \to a$ and $a^{i+1}b \to b$ is applied to a Sturmian string (with the possible exception of the first block), the resulting string is again balanced and not ultimately periodic, hence Sturmian. See also [R99].

The structural definition of Sturmian strings can easily be adapted for doubly infinite strings: all we have to do is to modify the definition of ultimately periodic strings so that the definition of balanced strings can remain unchanged. To define ultimate periodicity for a doubly infinite string $\boldsymbol{S}$, fix for a moment an enumeration of $\boldsymbol{S}$ indexed by the integers. Thus $\boldsymbol{S}[i]$ for any integer i is meaningful. $\boldsymbol{S}$ is *ultimately right periodic* if for any integer i, the infinite string $\boldsymbol{S}[i, i+1, ...]$ is ultimately periodic; $\boldsymbol{S}$ is *ultimately left periodic* if for any integer i, the infinite string $\boldsymbol{S}[i]\boldsymbol{S}[i-1]\boldsymbol{S}[i-2]....$ is ultimately periodic. It is clear that the property of being ultimately right (left) periodic or otherwise does not depend on the choice of the beginning of the enumeration, so the definition is sound. Thus, a doubly infinite string is Sturmian if it is (a) neither ultimately right nor ultimately left periodic and (b) balanced. It follows that a doubly infinite Sturmian string consists of blocks of the form $a^i b$ and $a^{i+1}b$ (or with the roles of a and b interchanged) and hence when the reduction $a^i b \to a$, $a^{i+1}b \to b$ is applied to the string, the resulting string is again a doubly infinite Sturmian string. For correctness, we must define how to apply a reduction $a^i b \to a$, $a^{i+1}b \to b$ to a doubly infinite string $\boldsymbol{S}$: choose a start, move to the right until the first b is encountered, then move one position to the right. Enumerate $\boldsymbol{S}$ by the integers so that the position just selected is 0. Now inductively define the block replacement by letters moving from position 0 one block to the right in odd steps and one block to the left in even steps. Again, it may be shown that the result is independent of the selection of the start position; hence the definition is sound.

Thus the class of doubly infinite Sturmian strings is closed under the operation of reduction where reduction for each Sturmian string exists and is determined by a unique parameter i. One can generalise this notion and define the class of *doubly infinite two-pattern strings of scope* λ as the maximum class containing the class of doubly infinite Sturmian strings that is closed under the operation of reduction where reduction $\boldsymbol{p}^i\boldsymbol{q} \to a$ and $\boldsymbol{p}^j\boldsymbol{q} \to b$ for each string exists and is determined by four parameters: a suitable pair of patterns $\boldsymbol{p}$ and $\boldsymbol{q}$ each of size $\leq \lambda$, and $0 < i < j$ (for Sturmian strings the four parameters reduce to the familiar ones $|\boldsymbol{p}| = |\boldsymbol{q}| = 1$ and $j = i + 1$). That such a maximum class exists follows from the axiom of choice given the simple fact that a union of all such classes is again such a class. The class definitely contains all doubly infinite Sturmian strings and it definitely does not contain all doubly infinite strings (it is easy to construct a doubly infinite string that does not consist of patterns $\boldsymbol{p}^i\boldsymbol{q}$ and $\boldsymbol{p}^j\boldsymbol{q}$ — for instance, take a doubly infinite Sturmian string $\boldsymbol{S}$, split it in two, and splice in between a finite string $aabb$). Does the class contain more than just doubly infinite Sturmian strings? Indeed, given any λ and any suitable pair of patterns $\boldsymbol{p}$ and $\boldsymbol{q}$, $|\boldsymbol{p}|, |\boldsymbol{q}| \leq \lambda$, and any i, j, $1 \leq i < j$, take a doubly infinite Sturmian string $\boldsymbol{S}$, "expand" each a of $\boldsymbol{S}$ to $\boldsymbol{p}^i\boldsymbol{q}$ and each b of $\boldsymbol{S}$ to $\boldsymbol{p}^j\boldsymbol{q}$. It is clear that the resulting doubly infinite string $\boldsymbol{S_1}$ is a member of the class of two-pattern strings of scope λ, yet it is not Sturmian (except of course for $|\boldsymbol{p}| = |\boldsymbol{q}| = 1$, $\boldsymbol{p} \neq \boldsymbol{q}$, and $j = i+1$). In fact, for any λ and any given infinite sequence of reductions $[\boldsymbol{p}_n, \boldsymbol{q}_n, i_n, j_n]_\lambda$, we can construct a doubly infinite string $\boldsymbol{S}$ that belongs to the class of two-pattern strings of scope λ with the property that the following sequence of reductions can be applied to it: for each $n \geq 2$ select a block $B_n = \boldsymbol{p}_n^{i_n}\boldsymbol{q}_n$ or $\boldsymbol{p}_n^{j_n}\boldsymbol{q}_n$ and a letter $x_n = a$ or b so that x_n occurs in the middle of B_n. Then start building a doubly infinite string in both directions by induction over levels as indicated by the diagram below.

It is clear that all finite fragments of the Fibonacci string and all finite fragments of Sturmian strings are also finite fragments of doubly infinite Sturmian strings. Moreover the complete two-pattern strings discussed in this paper are "block-complete" finite fragments of doubly infinite two-pattern strings of scope λ. In the same spirit as finite Sturmian strings, we can define finite two-pattern strings of scope λ as finite fragments of doubly infinite two-pattern strings of scope λ. And similarly, for Sturmian strings, any finite two-pattern string of scope λ is a substring of some complete two-pattern string of scope λ.



5 Remark

The results of this paper hold also if the definition of $\boldsymbol{p}$-regular is made more restrictive, as specified below. Since the suitability of a pair $(\boldsymbol{p}, \boldsymbol{q})$ of patterns depends on $\boldsymbol{q}$ being *not* $\boldsymbol{p}$-regular, a more restrictive definition of $\boldsymbol{p}$-regular results in more pairs $(\boldsymbol{p}, \boldsymbol{q})$ being accepted as suitable, thus increasing the frequency of occurrence of two-pattern strings among all strings on $\{a, b\}$. In this paper we have used a less restrictive definition of $\boldsymbol{p}$-regular in order to simplify the presentation. The alternate definition is as follows:

Definition 5 *A string* $\boldsymbol{q}$ *is said to be* **$\boldsymbol{p}$-regular** *if and only if there exist (possibly empty) strings* $\boldsymbol{u}, \boldsymbol{v}$ *together with nonnegative integers* $n_1, n_2, \ldots, n_k$, $k \geq 1$, $r \geq 1$, *such that*

- $\boldsymbol{p}$ *is neither a prefix nor a suffix of* $\boldsymbol{u}$*;*
- $\boldsymbol{p}$ *is neither a prefix nor a suffix of* $\boldsymbol{v}$*;*
- *the integers* n_i *assume at most two distinct values — that is,*

$$\left|\{n_i : \ i \in 1..k\}\right| \leq 2;$$

- $\boldsymbol{q} = (\boldsymbol{u}\boldsymbol{p}^r\boldsymbol{v}\boldsymbol{p}^{n_1})(\boldsymbol{u}\boldsymbol{p}^r\boldsymbol{v}\boldsymbol{p}^{n_2})\cdots(\boldsymbol{u}\boldsymbol{p}^r\boldsymbol{v}\boldsymbol{p}^{n_k})$.

References

[AHU74] Alfred V. Aho, John E. Hopcroft & Jeffey D. Ullman, *The Design & Analysis of Computer Algorithms*, Addison-Wesley (1974).

[BF84] M. Boshernitzan & Aviezri S. Fraenkel, **A linear algorithm for nonhomogeneous spectra of numbers**, *J. Algorithms 5* (1984) 187-198.

[BM77] Robert S. Boyer & J. Strother Moore, **A fast string searching algorithm**, *CACM 20–10* (1977) 762–772.

[F97] Martin Farach, **Optimal suffix tree construction with large alphabets**, *Proc.* 38^{th} *Annual IEEE Symp. FOCS* (1997) 137-143.

[FS99] Aviezri S. Fraenkel & R. Jamie Simpson, **The exact number of squares in Fibonacci words**, *TCS 218-1* (1999) 83-94.

[FJS01] František Franěk, Jiandong Jiang & W. F. Smyth, **Two-pattern strings III — frequency of occurrence**, submitted for publication.

[FKS00] František Franěk, Ayşe Karaman & W. F. Smyth, **Repetitions in Sturmian strings**, *TCS 249-2* (2000) 289-303.

[FLS02] František Franěk, Weilin Lu & W. F. Smyth, **Two-pattern strings II — computing all the repetitions**, submitted for publication.

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

[KK00] Roman Kolpakov & Gregory Kucherov, **On maximal repetitions in words**, *J. Discrete Algorithms 1* (2000) 159-186.

[LZ76] Abraham Lempel & Jacob Ziv, **On the complexity of finite sequences**, *IEEE Trans. Information Theory 22* (1976) 75–81.

[L02] M. Lothaire (ed.), *Algebraic Combinatorics on Words*, Encyclopedia of Mathematics & Its Applications, vol. 90, Cambridge University Press (2002). See also

`http://www-igm.univ-mlv.fr/~berstel/`

[M89] Michael G. Main, **Detecting leftmost maximal periodicities**, *Discrete Applied Maths. 25* (1989) 145-153.

[R99] G. Richomme, **Test words for Sturmian morphisms**, *Bull. Belg. Math. Soc. 6* (1999) 481-489.

[ZL77] Jacob Ziv & Abraham Lempel, **A universal algorithm for sequential data compression**, *IEEE Trans. Information Theory 23* (1977) 337-343.

Acknowledgements

The authors acknowledge the support provided by the Natural Sciences & Engineering Research Council of Canada. They would also like to thank the anonymous referees for their comments that have materially improved both the substance and the presentation of this paper.