REPETITIONS IN STURMIAN STRINGS

František Franěk

Department of Computing & Software
McMaster University
e-mail: `franya@cas.mcmaster.ca`

Ayşe Karaman

Department of Computing & Software
McMaster University
e-mail: `karamaa@mcmaster.ca`

W. F. Smyth

Department of Computing & Software
McMaster University
e-mail: `smyth@mcmaster.ca`

School of Computing
Curtin University of Technology
e-mail: `smyth@cs.curtin.edu.au`

ABSTRACT

In this paper we apply a simple representation of Sturmian strings, which we call a "reduction sequence", to three algorithms. The first algorithm accepts as input a given finite string x and determines in time $O(|x|)$ whether or not x is Sturmian. The second algorithm is a modification of the first that, in the case that x is Sturmian, outputs a reduction sequence for a superstring u of x that is a prefix of an infinite Sturmian string. The third algorithm uses the reduction sequence of u to compute all the repetitions in u in time $\Theta(|u|)$, thus extending a recent result for Fibonacci strings. The third algorithm is also based on a characterization of the repetitions in a Sturmian string that describes them compactly in terms of "runs". Finally, for every integer $r \geq 4$, we show how to construct an infinite Sturmian string that contains maximal repetitions of exponents $2, 3, \ldots, r-1$, but none of exponent r.

KEYWORDS: string, word, Sturmian, repetition, algorithm

1 INTRODUCTION

An *infinite Sturmian string* is a string on $\{a, b\}$ that extends to infinity in one direction (to the right, say) and that for every positive integer k contains exactly $k+1$ distinct substrings of length k (out of a total of 2^k possibilities). A string that for some integer k contains exactly k distinct substrings is necessarily a repetition of a primitive prefix $x[1..k]^p$. Thus infinite Sturmian strings can be thought of as the strings of minimum variation that are also interesting. In fact, they are *very* interesting: they derive from the Sturm sequences of numerical analysis via Christoffel

[3] and Morse/Hedlund [12], they have been much studied [14,13,11,10,9,8,1], and there are numerous quite different ways of characterizing them.

We will find it useful to recall one of these characterizations: an infinite string x on $\{a, b\}$ is Sturmian if and only if x satisfies the following conditions:

* x is *aperiodic* — that is, no suffix of x is a repetition;
* x is *balanced* — that is, if $\phi_a(u)$ denotes the number of a's in any substring u of x, then for any substring v of x such that $|v| = |u|$,

$$|\phi_a(v) - \phi_a(u)| \leq 1.$$

In this paper we discuss algorithms on Sturmian strings, and we focus therefore on *finite Sturmian strings* — that is, on finite substrings of infinite Sturmian strings. Specifically, given a finite string x, we show how to compute in time $O(|x|)$ whether or not x is Sturmian; and given a finite prefix u of an infinite Sturmian string, we show how to compute all the repetitions in u in time $\Theta(|u|)$. The efficiency of these algorithms depends on a simple representation of infinite Sturmian strings closely related to results given in [5].

Section 2 describes this representation, which we call a "reduction sequence", and explains its essential properties. Section 3 presents a simple algorithm that uses the idea of a reduction to determine whether or not x is Sturmian. Section 4 describes a modification of this algorithm that in addition outputs the reduction sequence of a superstring u of x, where u is a finite prefix of an infinite Sturmian string. In Section 5 we show how the reduction sequence of u can be used to compute all the repetitions in u in linear time. Finally, Section 6 raises some open problems.

2 A REPRESENTATION OF STURMIAN STRINGS

Some of the material in this section has already been developed in a different form in [5], and in a quite different context. Since an understanding of the material and its context is necessary to an understanding of our algorithms, we provide in this section a brief overview of the main ideas.

We begin with three simple observations about Sturmian strings:

(1) An infinite Sturmian string s contains exactly one of the substrings aa and bb. Therefore every infinite Sturmian string consists either of repeated occurrences of a separated by single occurrences of b, or of repeated b's separated by single a's. We speak then of the *repeating letter* and the *single letter*. Without loss of generality, we shall suppose throughout this paper that the single letter is b, and we shall use the term *block* to denote any occurrence of $a^q b$ in s, $q \geq 0$, that cannot be extended to the left in s. For example, a Sturmian string

$$s = baabaaabaabaabaaab \cdots$$

contains blocks b, aab and $aaab$.

(2) To the right of the leftmost occurrence of the single letter b, every infinite Sturmian string s is a concatenation of *short blocks* ($a^p b$) and *long blocks* ($a^{p+1} b$) for some specific integer $p \geq 1$ — it is easy to see that no other block, except

possibly for the leftmost one, can exist without causing s to be unbalanced in the sense defined above.

Furthermore, after the first block (that is, to the right of the leftmost b), s consists either of repeated occurrences of short blocks separated by single occurrences of long ones, or else repeated occurrences of long blocks separated by single occurrences of short ones. For if only single occurrences of each block occur, s has a repetitive suffix $(a^pba^{p+1}b)^\infty$ and so is aperiodic; while if both a^pb and $a^{p+1}b$ repeat, there necessarily exist two substrings $u = ba^pba^pb$ and $v = a^{p+1}ba^{p+1}$ of length $|u| = |v| = 2p+3$ such that $\phi_a(v) - \phi_a(u) = 2$, an imbalance.

We speak then of the *repeating block* and the *single block*. We call p the *signature* of s; note that $p+1 = |a^pb|$ is the length of a short block.

(3) As suggested in (2), the leftmost block of a (finite or infinite) Sturmian string is to some degree arbitrary: if a Sturmian string s contains *only* short blocks and long blocks, then $s' = a^qbs$ is Sturmian for any integer $q \in 0..p$, since the prefix a^qb cannot disrupt the balance; if moreover the leftmost block of s is short or if long blocks in s are repeating blocks, then $s' = a^{p+1}bs$ must also be Sturmian. To put it another way, if the leftmost block of a Sturmian string is removed, the remaining suffix is still Sturmian.

These observations suggest the idea of a *reduction* of a string; that is, a mapping that takes each repeating block into the repeating letter a and each single block into the single letter b, of course based on a prior determination of which block is single and which repeating. A reduction may then be thought of as the inverse of a morphism, α say, on $\{a,b\}$ such that

$$\alpha : a \to a^qb; \quad b \to a^{q'}b$$

where (q,q') is fixed at either $(p,p+1)$ (repeating block is short) or $(p+1,p)$ (repeating block is long). We therefore, by a slight abuse of notation, use the symbol α^{-1} to denote a reduction. A reduction is defined for any string on $\{a,b\}$ that contains only blocks whose lengths differ by at most one; in particular, by observation (3) above, a reduction is defined for at least that part of every infinite Sturmian string s that occurs to the right of the first occurrence of b. A string x on $\{a,b\}$ is said to be *reducible* to y if and only if $x = u\alpha(y)$ for some choice of the morphism α, where u is either empty or a block. Thus every Sturmian string s is reducible. If in fact α^{-1} is defined for every block in s including the first one, so that $\alpha\big(\alpha^{-1}(s)\big) = s$, we say that the reduction is *exact*.

As an example of a reduction, consider an initial segment of an infinite Sturmian string

$$s = ababab aababababaabababaababababaabababaabababaababab aab\cdots\cdots.$$

Then with signature $p = 1$, the first reduction

$$\alpha^{-1}(s) = aaabaaabaabaaabaaabaaabaab\cdots\cdots,$$

where the short block has been replaced by a, so that with $p = 2$, the second reduction

$$\alpha^{-2}(s) = aabaaab\cdots\cdots,$$

where the short block has been replaced by b. Both of these reductions are exact. Now however observe that, because only the first 59 letters of s have been specified, we cannot with certainty describe any further reduction, since $\alpha^{-2}(s)$ might for example be

$$aabaaabaaaab\cdots \quad \text{or} \quad aabaaabaab\cdots,$$

both Sturmian strings but with differing signatures (3 and 2, respectively). Observe further that, in the first of these strings, the initial block aab is neither short nor long; it is a *partial block* that would have to be excluded from the next reduction. Thus, in this case, any further reduction of $aabaaabaaaab$ could not be exact in the sense defined above.

The following theorem shows that every infinite Sturmian string gives rise to an infinite sequence of reductions.

Theorem 2.1 An infinite string x is Sturmian if and only if it is reducible to a Sturmian string.

Proof See [5] and [7]. □

Note that, in order to fully determine a reduction of s, it suffices to specify a pair (p, λ), where p is the signature of s and λ the letter of $\alpha^{-1}(s)$ that every short block of s maps into. Adopting the notation $s_n = \alpha^{-n}(s)$ with $s_0 = s$ by convention, we may then define a *reduction sequence* of a given infinite Sturmian string s to be an infinite sequence of pairs

$$\big\langle (p_n, \lambda_n) \mid n \geq 1 \big\rangle,$$

where p_n is the signature of s_{n-1} and λ_n is the image in s_n of short blocks in s_{n-1}. Then a reduction sequence of the Sturmian string

$$s = ababaabababaababaabababaabababaababaababaabababaab\cdots$$

given in the example above is

$$\big\langle (1, a), (2, b), \ldots . \big\rangle,$$

corresponding to reductions

$$\begin{aligned} \alpha^{-1}(s) &= aaabaaabaabaaabaaabaaabaab\cdots \\ \alpha^{-2}(s) &= aabaaab\cdots \\ &\vdots \end{aligned}$$

Clearly every infinite Sturmian string s gives rise to exactly one reduction sequence, so that we may speak of *the* reduction sequence of s. In order to describe the relationship between Sturmian strings and reduction sequences more precisely, we need the following definition: if every reduction in the reduction sequence of s is exact, then s is said to be *block-complete*. Thus, informally, block-complete infinite Sturmian strings are those for which no reduction gives rise to an irreducible prefix. As our first formal result shows, each reduction sequence corresponds to one and only one block-complete Sturmian string, and so uniquely identifies it.

Lemma 2.2 No two distinct block-complete infinite Sturmian strings have the same reduction sequence.

Proof Suppose on the contrary that there exist two block-complete Sturmian strings s and t with the same reduction sequence. Let i denote the first position such that $s[i] \neq t[i]$, and observe that since s and t are block-complete, they must therefore have the same first letter, so that $i > 1$. Without loss of generality, suppose that $s[i] = a$, $t[i] = b$, so that s has a long block where t has a short block. Now consider the effect of applying the same reduction to both s and t, assuming, again without loss of generality, that a short block maps into b, a long block into a. Since both s and t are block-complete, for some integer $j > i$ there will be prefixes $\alpha^{-1}\bigl(s[1..j]\bigr) = ua$ and $\alpha^{-1}\bigl(t[1..i]\bigr) = ub$ of $\alpha^{-1}(s)$ and $\alpha^{-1}(t)$ respectively, with $i_1 = |ua| < i$. In other words, the effect of the reduction has been to duplicate at position i_1 in two distinct block-complete Sturmian strings $\alpha^{-1}(s)$ and $\alpha^{-1}(t)$ the condition that initially existed at position i of s and t. Since $i_1 < i$, these reductions can be carried out at most a finite number of times, a contradiction since a reduction sequence is infinitely long. $\Box$

We turn now to a consideration of the morphism α rather than its inverse α^{-1}; in particular, we consider the series of *expansions* resulting from a partial reduction sequence

$$\bigl\langle(p_1, \lambda_1), (p_2, \lambda_2), \ldots, (p_k, \lambda_k)\bigr\rangle,$$

for any integer $k \geq 1$. Since the sequence of k reductions α^{-1} is well-defined by this sequence, it follows that the k expansions, executed in reverse order, are also well defined. Denoting by α_i the morphism defined by (p_i, λ_i), $i = 1, 2, \ldots, k$, we consider the compound morphism

$$\alpha^{(k)}(a) = \alpha_1\Bigl(\alpha_2\bigl(\ldots\alpha_k(a)\ldots\bigr)\Bigr).$$

Since

$$\alpha^{(k+1)}(a) = \alpha_1\Bigl(\alpha_2\bigl(\ldots\alpha_k\bigl(\alpha_{k+1}(a)\bigr)\ldots\bigr)\Bigr)$$

and every $\alpha_i(a)$ has the prefix a, we see that $\alpha^{(k)}(a)$ is necessarily a prefix of $\alpha^{(k+1)}(a)$ for every k. Thus

$$\lim_{k\to\infty} \alpha^{(k)}(a)$$

exists and defines the unique block-complete infinite Sturmian string corresponding to the reduction sequence

$$\bigl\langle(p_n, \lambda_n)\bigr\rangle \mid n \geq 1\bigr\rangle.$$

Thus, in view of Lemma 2.2, we have established a 1-1 correspondence between reduction sequences and block-complete Sturmian strings.

Referring to the above example, consider the truncated sequence

$$\bigl\langle(1, a), (2, b)\bigr\rangle$$

for $k = 2$: $(2, b)$ defines the mapping

$$\alpha_2(a) = aaab,$$

since a maps into a long block, and then compounding with $(1, a)$ yields

$$\alpha_1\bigl(\alpha_2(a)\bigr) = abababaab,$$

since a now maps into a short block. Observe that $\alpha_1\big(\alpha_2(a)\big)$ is as expected a prefix of the example string s.

3 DECIDING WHETHER x IS STURMIAN

In this section we outline a simple algorithm that, given an arbitrary string x on $\{a,b\}$ of length n, determines in time $O(n)$ whether or not x is Sturmian. Thus we provide a means of identifying a finite Sturmian string without any explicit reference to the infinite Sturmian string of which it is a substring. The algorithm makes use of the fact, established in Theorem 2.1 for infinite strings, that a string can be Sturmian only if it reduces to a Sturmian string.

In observation (3) of Section 1, it was remarked that if x is a Sturmian string, finite or infinite, it must break down into blocks, of which the first may be partial. Observe further that a finite Sturmian string x may also have a partial last block a^j for some $j \in 1..p+1$. For example, $x = abaaabaaaabaa$ is a substring of a Sturmian string, hence by definition Sturmian, but the first block ab and the last block aa are both partial. Given an arbitrary nonempty string x on $\{a,b\}$, we call the prefix ending with the first occurrence of b, if any, the *head* of x, written $h(x)$; if b does not occur in x, then $h(x) = \epsilon$, the empty string. Similarly, we call the suffix beginning after the final occurrence of b, if any, the *tail* of x, written $t(x)$; if b does not occur in x, then $t(x) = x$. The *core* of x, written $c(x)$, is the string that remains after the head and tail are removed. In the above example, $h(x) = ab$, $t(x) = aa$, and $c(x) = aaabaaaab$. Note that each of the head, tail and core can be the empty string; for example, if $x = a^j$ for some integer $j > 0$, $h(x) = \epsilon$, $t(x) = x$ and $c(x) = \epsilon$; while if $x = a^j b$, $h(x) = x$ and $t(x) = c(x) = \epsilon$.

First we describe an algorithm that determines whether or not a given nonempty string x on $\{a,b\}$ is Sturmian, then go on in Section 4 to show that this algorithm can be modified to enable the reduction sequence of a superstring u of x to be computed in the case that x is in fact Sturmian. The algorithm depends on a simple observation: since $h(x)$ and $t(x)$ may be partial blocks, they are of interest only if they contain more than p occurrences of a ($p+1$ occurrences imply a long block, more than $p+1$ imply that x is not Sturmian); otherwise, $h(x)$ and $t(x)$ may be discarded, since they cannot influence the decision on whether or not a later reduction of x is Sturmian. For example, the strings $aaabab\cdots$ and $ababaaa\cdots$ cannot be Sturmian because $p = 1$ for both of them and three consecutive a's occur in head and tail, respectively. On the other hand, $aababaabaa$ can be a Sturmian string only if it is assumed that $h(x) = aab$ is a complete long block and that $t(x) = aa$ is followed by b, thus a prefix of a long block.

Algorithm `STURM(x)` consists of the following four steps:

(1) Compute the signature p of x if possible:
In this step `true` *is returned if the core of x is empty,* `false` *if there is no valid signature. Otherwise, if there is no exit, then after this step p is determined and a reduction can be performed on x.*

(2) Compute λ, the letter that every short block maps into:

The repeating block, if there is one, should map into a. Hence if the repeating block is short, $\lambda \leftarrow a$; otherwise, $\lambda \leftarrow b$.

(3) Adjust the head and tail of x as required:
If the head is not a long block, it is deleted; if the tail has at most p occurrences of a, it is deleted; if the tail is a^{p+1}, b is appended to turn it into a long block.

(4) Recursively apply the algorithm to the (p, λ) reduction of x:
Here we make use of the fact that x can be Sturmian only if its reduction is Sturmian.

Step (1) may be expanded for clarity:

(1.1) Compute p_{min}, the smallest number of adjacent occurrences of a within the core:
For this calculation, any shorter run of a's in $h(x)$ or $t(x)$ is ignored. Note that $p_{min} = 0$ if bb occurs in x or if there are fewer than two occurrences of b in x.

(1.2) Compute p_{max}, the longest run of adjacent a's in x:
This calculation includes runs of a's in $h(x)$ and $t(x)$.

(1.3) If $c(x) = \epsilon$, then return `true`:
This is the case in which $x = a^j$ or $a^{j_1}ba^{j_2}$, both Sturmian strings. Therefore the original given string must by Theorem 2.1 have been Sturmian.

(1.4) If $p_{min} = 0$ or $p_{max} - p_{min} > 1$, then return `false`:
x is not Sturmian if it contains bb or if there exists a block longer by more than one letter than a short block.

(1.5) $p \leftarrow p_{min}$.

Theorem 3.1 Algorithm `STURM(x)` correctly determines in time $O(|x|)$ whether or not a given nonempty finite string x on $\{a, b\}$ is Sturmian.

Proof It is clear from the expanded version of Step (1) that the signature of x is computed correctly. Similarly in Step (2), λ is correct when a repeating block exists and otherwise is arbitrarily set to b. Step (3) correctly adjusts the head and tail of x, and Step (4) is the recursive application of `STURM` to the reduction of x. To see that the algorithm performs this reduction correctly, observe that x may take only the following forms:

(a) $x = a^j$ for some $j > 0$;
(b) $x = a^{j_1}ba^{j_2}$ for some $j_1 \geq 0$, $j_2 \geq 0$;
(c) $x = a^{p+1}bva^j$ for some $j \in 0..p+1$ and some nonempty string v of signature $p \geq 1$;
(d) $x = a^{j_1}bva^{j_2}$ for some $j_1 \in 0..p$, $j_2 \in 0..p+1$, and some nonempty string v of signature $p \geq 1$;
(e) x has an undefined signature.

Cases (a) and (b) are handled by Step (1.3), case (e) by Step (1.4). Cases (c) and (d) are processed by Step (3), and it is straightforward to verify that, in each of these cases, the reduction α^{-1} yields

$$0 < \left|\alpha^{-1}(x)\right| < |x|/2,$$

so that STURM(x) must terminate with one of the forms (a), (b) or (e). We conclude that STURM(x) is correct.

Now consider Steps (1)-(3). Step (1) requires two counts to be maintained: one for the minimum run of a's (within the core), the other for the maximum run of a's (in the entire string). Step (2) requires that in the core the repeating block, if it exists, be identified as short or long; while Step (3) requires only that counts be kept of the a's in the head and tail, a task already included in Step (1). Thus Steps (1)-(3) altogether can be implemented to require at most a single scan of the current reduction of x. Since as we have seen a reduction of x decreases the length of x by at least a factor of 2, it follows that the total length of string scanned by all recursive calls of STURM(x) is less than $2|x|$. Therefore STURM(x) executes in time $O\big(|x|\big)$. $\square$

Suppose now that x is in fact a finite Sturmian string. In this case, STURM(x) will at each recursive step compute a signature p and corresponding λ; thus it reduces x by a sequence of well-defined reductions (p, λ), eventually yielding one of the trivial forms described in Step (1.3) that correspond to reductions (j, a) and $\big(\max(j_1, j_2), a\big)$, respectively. Taking into account the possibility that the head and tail may be discarded at each stage, we see that for some integer $k \geq 1$, STURM(x) determines a reduction sequence

$$\big\langle (p_1, \lambda_1), (p_2, \lambda_2), \ldots, (p_k, \lambda_k) \big\rangle$$

corresponding to a prefix u of a block-complete Sturmian string. It is tempting to suppose (as we did at first) that u must be a substring of x. But as the following example shows, this is not necessarily the case.

Consider the string

$$x = aaab\ aab\ aaab\ aab\ aaab\ aab\ aaa.$$

STURM(x) would in this case return true, determining in the process a reduction sequence

$$\big\langle (2, a), (1, a), (3, a) \big\rangle$$

that in fact reduces x to the letter a. However, if this reduction sequence is applied to a, the result is

$$u = aab\ aaab\ aab\ aaab\ aab\ aaab\ aab\ aab\ aaab,$$

a block-complete prefix of an infinite Sturmian string but neither a substring nor a superstring of x! This phenomenon results from adjustments to the head and tail of x as specified in Step (3) of the algorithm, and Step (3) can in fact be modified to ensure that the reduction sequence at least specifies a substring of x. But even with such a change, STURM(x) provides no basis for determining what we really want: the reduction sequence of a block-complete superstring of x. To achieve this objective, somewhat more sophisticated modifications are required, as explained in the next section.

4 COMPUTING THE REDUCTION SEQUENCE

The problem with the algorithm `STURM(x)` described in Section 3 is *not* that it yields an incorrect reduction sequence. In fact, the reduction sequence, so far as it goes, corresponds exactly to some superstring of the given string x: each reduction (p, λ) is correct for the current core and consistent with the current head and tail. Thus the mismatch between x and the reconstituted string u derives from the fact that at the lowest level of recursion, no account is taken of the number of times that the head and tail have previously been deleted by Step (3) of the algorithm. If there are sufficient deletions, it may happen, as in the example of Section 3, that u will not be long enough and so may only overlap with x rather than being a superstring of it.

Indeed, in the example

$$x = aaab\ aab\ aaab\ aab\ aaab\ aab\ aaa$$

of Section 3, the slightly altered reduction sequence

$$\big\langle (2, a), (1, a), (4, a) \big\rangle$$

yields a superstring

$$u = aab\ aaab\ aab\ aaab\ aab\ aaab\ aab\ aaab\ aab\ aab\ aaab$$

of x. This new reduction sequence merely adds one a to the first expansion, taking account of one deleted head and thus ensuring that u is a superstring.

In order to record the number of times that the head $h(x)$ and the tail $t(x)$ are deleted, we introduce two counters Δ_h and Δ_t, respectively. These counters will be updated in Step (3) and then used in Step (1.3) (when the core $c(x) = \epsilon$) to adjust the final reductions. Thus we can derive a new algorithm `REDUCE(x)` from `STURM(x)` by making the following changes:

Step (0) Initialize $\Delta_h \leftarrow 0$, $\Delta_t \leftarrow 0$.

Step (1.3) If $c(x) = \epsilon$, then return `true` and output the final reductions with sufficiently large signature:

if $h(x) = \epsilon$ **then** $\{x = a^j\}$
  $j \leftarrow j + \Delta_h + \Delta_t$
  output reduction (j, a)
else $\{x = a^{j_1} b a^{j_2}\}$
  $j_1 \leftarrow j_1 + \Delta_h$
  $j_2 \leftarrow j_2 + \Delta_t$
  if $j_1 > j_2$ **then**
    output reductions $(j_1, b), (1, a)$
  else
    output reductions $(j_2, a), (1, a)$.

Step (2) After λ has been computed, output the current reduction (p, λ).

Step (3) Adjust the head $h(x)$ and tail $t(x)$ of x, incrementing the counters Δ_h and Δ_t as deletions occur of head and tail, respectively:

```
if h(x) = a^j b for some 0 ≤ j ≤ p then
    if the adjacent block is a single block then
        h(x) ← repeating block
    else
        delete h(x)
        Δ_h ← Δ_h + 1
if t(x) = a^{p+1} then
    t(x) ← a^{p+1} b
elseif t(x) ≠ ϵ then
    if the adjacent block is a single block then
        t(x) ← repeating block
    else
        delete t(x)
        Δ_t ← Δ_t + 1.
```

Thus the new algorithm `REDUCE(x)` uses the counters to ensure that the reconstituted string u is long enough to always be a superstring of x. Hence

Theorem 4.1 Given a finite Sturmian string x, Algorithm `REDUCE(x)` correctly computes in time $\Theta(|x|)$ a reduction sequence of a superstring u of x that is a prefix of a block-complete Sturmian string.

Proof The asymptotic time requirement of `REDUCE(x)` is exactly the same as that of `STURM(x)`, and so is linear in $|x|$. The string u corresponding to the reduction sequence must by definition be block-complete. As we have seen above, the reduction sequence computed by `REDUCE(x)` corresponds to a superstring of x. $\square$

Note that `REDUCE(x)` does not compute u, only the reduction sequence of u; thus even though the algorithm executes in time $\Theta(|x|)$, it remains possible that $|u|$ is actually supralinear in $|x|$. In fact, if we imagine a string x whose every reduction has signature p with $h(x) = b$ and $t(x) = a$ at each step, then the final reduction computed by Step (1.3) could have as many as

$$\Delta_h + \Delta_t = 2\log_p n$$

additional a's that after $\log_p n$ expansions would yield a string u with

$$p^{2\log_p n} = n^2$$

additional letters.

5 COMPUTING THE REPETITIONS

In this section we describe another simple algorithm that, given the reduction sequence of a finite block-complete prefix u of an infinite Sturmian string, computes all the repetitions in u in time $\Theta(|u|)$. We begin with some useful definitions.

Following Crochemore [4], we define a *repetition* in a given string x of length n to be a triple (i, q, r) of positive integers with the following properties, where $u \equiv x[i..i+q-1]$:

* $u^r = x[i..i+rq-1]$;
* $r \geq 2$;
* u is *primitive* (not itself of the form v^r, v nonempty, $r \geq 2$).

u is said to be the *generator* of the repetition, q its *period*, and r its *exponent*. If in addition

* either $i \leq q$ or else $u \neq x[i-q..i-1]$, and
* either $n < i+(r+1)q-1$ or else $u \neq x[i+rq..i+(r+1)q-1]$,

then the repetition is said to be *maximal*. Observe that for $r > 2$, the repetition (i, q, r) implies repetitions of *rotations* of u; that is, of substrings $u_j = x[i+j..i+q+j-1]$ for every integer $j \in 1..q-1$. Specifically, the implied repetitions are $(i+j, q, r-1)$. This remark suggests the following definition: a *run* in x is a 4-tuple (i, q, r, t) satisfying the following properties:

* for every integer $j \in i..i+t-1$, (j, q, r) is a maximal repetition;
* either $i = 1$ or else $x[i-1] \neq x[i+q-2]$;
* either $n = i+t+rq-2$ or else $x[i+t] \neq x[i+t+rq-1]$.

The second and third of these properties ensure that a run is *nonextendible*; that is, it cannot be extended either to left or right to yield runs $(i-1, q, r, t+1)$ or $(i, q, r, t+1)$, respectively.

Now for a Sturmian string s we define a special kind of substring called an *r-kernel*; that is, for an integer $r \geq 1$ and a (possibly empty) substring w of s, either one of the following two forms: $b(wa)^r wb$ or $a(wb)^r wa$. As we shall see, these forms arise when we try to perform a reduction on a run in a Sturmian string. We show first then that these forms can exist only in very special cases:

Lemma 5.1 An r-kernel can exist in a Sturmian string s only in one of the following three forms:

(a) $ba^p b$ or $ba^{p+1}b$, a p-kernel or $(p+1)$-kernel with $w = \epsilon$;
(b) $a^q ba^q$ for some $1 \leq q \leq p+1$, a 1-kernel with $w = a^{q-1}$;
(c) $a(a^p b)^r (a^p)a$ with $w = a^p$;

where p is the signature of s. Further, a reduction performed on an r-kernel of form (c) yields an r-kernel of form (a).

Proof Suppose $u = b(wa)^r wb$ is a substring of s with $w \neq \epsilon$. Since w is both preceded and followed by b, it must have a^p as both prefix and suffix. But then, since waw occurs in u, it follows that a^{2p+1} occurs, an impossibility in a Sturmian string. Thus w must be empty, and so the only possibilities are those stated in (a).

Next suppose that $u = a(wb)^r wa$. Observe that for $r = 1$, w may be any one of $\epsilon, a, \ldots, a^p$, so that u takes the form (b). Observe further that for $r > 1$, $w \neq \epsilon$. Consider then the case in which, for arbitrary $r \geq 1$, w contains at least one occurrence of b. We argue as above that therefore w has a^p as both prefix and suffix, so that u has a^{p+1} as both prefix and suffix. Hence we may consider $u' = ub$, a substring of s formed from full blocks, that we see contains the substring $a^p ba^p b$; thus $a^p b$ is the repeating block in

u' that under a reduction would map into a. Applying the reduction α^{-1} to u' then yields the substring

$$\alpha^{-1}(u') = b\big(\alpha^{-1}(w')a\big)^r \alpha^{-1}(w')b,$$

for some w'. As we have just seen, this form is possible only if $\alpha^{-1}(w')$ is empty, hence only if w' itself is empty, so that $w = a^p$, as required for case (c). $\square$

We say that a repetition or a run is *nontrivial* if it is not of the form a^p or a^{p+1}. The following theorem shows that nontrivial repetitions in Sturmian strings derive ultimately from expansions of kernels.

Theorem 5.2 Every nontrivial run (i,q,r,t) in a block-complete Sturmian string s is an expansion of one of the following:
(a) a run of $\alpha^{-1}(s)$ of exponent r;
(b) an $(r-1)$-kernel of $\alpha^{-1}(s)$.

Proof Let u denote the generator of the first repetition (i,q,r) in the run $R = (i,q,r,t)$. Observe that since it is both nontrivial and nonextendible, R must have prefix a^pb or, if $i=1$, possibly $a^{p+1}b$. There are two main cases:

(1) u has suffix b.

(1.1) If $i=1$, u is an integral number of full blocks and so R must be an expansion of a run of exponent r.

(1.2) If $i>1$, nonextendibility implies that $s[i-1]=a$. Hence

$$aR = a(a^pb\cdots b)^r a^p v$$

for some substring v. If $v=\epsilon$, aR is an expansion of an $(r-1)$-kernel, either $b(wa)^{r-1}wb$ or $a(wb)^{r-1}wa$ depending on whether b maps into long blocks or short blocks, respectively. On the other hand, if $v \neq \epsilon$, aR must be an expansion of one of the runs $bR' = b(wa)^r\cdots$ or $aR' = a(wb)^r\cdots$.

Thus in case (1) the theorem holds.

(2) u has suffix a.

A similar, slightly simpler argument establishes that the theorem holds in this case also. $\square$

In view of Lemma 5.1, Theorem 5.2 tells us that runs in Sturmian strings ultimately reduce to trivial repetitions or to the special form a^qba^q. To gain an understanding of how these runs are formed and overlap each other, consider the fragment *baaabaab* of a Sturmian string. This fragment contains four kernels: *baaab*, *aabaa*, *aba*, and *baab*. Suppose now that the fragment is expanded using signature $p=1$. There are two possibilities:

$$\begin{aligned} baaabaab &\to a\underline{ababa\underline{b\underline{abaaba}bab}a}baab \qquad (1,a)\\ &\to \underline{abaaba\underline{aba\underline{ababa}aba}aba}b \qquad (1,b)\end{aligned}$$

Here in the first line the runs $(aba)^2$, $(ababa)^2$, $(ab)^3a$ and $(ab)^2a$ are underlined, expansions of the kernels aba, a^2ba^2, $baaab$ and $baab$, respectively. In the second line the underlined runs $(ab)^2a$, $(abaab)^2a$, $(aba)^4$ and $(aba)^3$ are expansions of the same kernels.

We see then that all the runs in a block-complete Sturmian string can be computed from its reduction sequence by applying successive expansions to the initial string a while at the same time tracking the expansions of the kernels a^qba^q, ba^pb and $ba^{p+1}b$. Since the number of these expansions is linear in the length of the string, we immediately have

Theorem 5.3 The number of runs in a finite Sturmian string is linear in the length of the string. $\square$

Furthermore, since all of the forms a^qba^q, ba^pb and $ba^{p+1}b$ can be located in a linear scan of each expansion, and since as we have seen the reduction sequence can be computed in linear time, it follows that all the runs can also be computed in linear time. Hence

Theorem 5.4 The runs in a block-complete prefix u of a Sturmian string can be computed in time $\Theta(|u|)$. $\square$

The preceding two theorems generalize results given for Fibonacci strings in [6], and so greatly extend the class of strings for which a linear-time all-repetitions algorithm exists. We omit here further details of the implementation of this algorithm.

We conclude by stating a final result that is also an easy consequence of Lemma 5.1 and Theorem 5.2, and that generalizes the well-known fact that Fibonacci strings contain squares and cubes, but not fourth powers.

Theorem 5.5 Corresponding to every signature p that occurs in its reduction sequence, a block-complete infinite Sturmian string s contains maximal repetitions of exponents 2, $p \geq 2$, $p+1$ and $p+2$, but no maximal repetitions of any other exponent.

Proof Every expansion of the 1-kernel aba yields either

$$a^pba^{p+1}ba^pb \quad \text{or} \quad a^{p+1}ba^pba^{p+1}b,$$

containing maximal squares $(a^pba)^2$ or $(a^pb)^2$, respectively. Analogous maximal squares are also produced by every expansion of every 1-kernel a^qba^q, $q > 1$, that exists in s.

Expansions of a and b yield maximal repetitions of exponents p and $p+1$. Subsequent expansions of these maximal repetitions yield further maximal repetitions of the same exponents p and $p+1$, in accordance with Theorem 5.2(a).

Every expansion of the $(p+1)$-kernel $ba^{p+1}b$ yields either

$$b\,\underbrace{a^qba^qb\cdots a^qb}_{p+1\text{ times}}\,a^{q+1}b \quad \text{or} \quad ab\,\underbrace{a^{q+1}ba^{q+1}b\cdots a^{q+1}b}_{p+1\text{ times}}\,a^qb,$$

giving rise to maximal repetitions $(ba^q)^{p+2}$ or $(aba^q)^{p+2}$, respectively, both of exponent $p+2$.

Thus maximal repetitions of exponents 2, p, $p+1$ and $p+2$ exist in s as claimed, and these exhaust the cases allowed by Lemma 5.1 and Theorem 5.2. $\square$

A classical string problem is the (α, r)-*avoidance problem* [15]: construct an infinite string on an alphabet of size α that contains no repetitions of exponent r (but that does contain repetitions of exponents $2, 3, \ldots, r-1$). From Theorem 5.5 it is clear that the $(2, r)$ problem can be solved for every $r \geq 4$ using Sturmian strings and selecting appropriate terms in the reduction sequence. For example, the block-complete Fibonacci string has reduction sequence

$$\{(1, b), (1, b), \ldots .\}$$

with signature $p = 1$ for every reduction; by Theorem 5.5 it therefore contains maximal repetitions only of exponents 2 and 3.

6 FUTURE WORK

As mentioned in Section 4, it may happen that the superstring u of x computed by Algorithm `REDUCE(x)` has length supralinear in $|x|$. Thus this paper leaves unresolved the question of whether there exists a linear time algorithm to compute a *minimum-length* block-complete superstring u of a given finite Sturmian string x. We conjecture that there does exist such an algorithm, and we conjecture further that $|u| < 4|x|$.

Finally, we remark that many of the results of this paper should be extendible to much more general classes of strings: those on which recursive reductions can be defined.

REFERENCES

[**1**] J. Berstel, **Recent results in sturmian words**, *Developments in Language Theory*, World Scientific (1996) 13-24.

[**2**] J. Berstel & P. Séébold, **A characterization of Sturmian morphisms**, *The Mathematical Foundations of Computer Science*, A. Borzyszkowski & S. Sokolowski (eds.), Springer-Verlag (1993) 281-290.

[**3**] E. B. Christoffel, **Observatio Arithmetica**, *Annali di Mathematica Pura ed Applicata* 6 (1875) 145-152.

[**4**] Maxime Crochemore, **An optimal algorithm for computing all the repetitions in a word**, *IPL* 12-5 (1981) 244-248.

[**5**] S. Dulucq & D. Gouyou-Beauchamps, **Sur les facteurs des suites de Sturm**, *TCS* 71 (1990) 381-400.

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

[**7**] Ayşe Karaman & W. F. Smyth, *A Representation of Sturmian Strings*, Tech. Rep. No. CAS 98-04, Department of Computing & Software, McMaster University (1998): `http://www.cas.mcmaster.ca/cas/research/reports.html`.

[**8**] Aldo de Luca & Filippo Mignosi, **Some combinatorial properties of Sturmian words**, *TCS* 136 (1994) 361-385.

[**9**] W. F. Lunnon & P. A. B. Pleasants, **Characterization of two-distance sequences**, *J. Austral. Math. Soc. (Series A)* 53 (1992) 198-218.

[**10**] Filippo Mignosi, **On the number of factors of Sturmian words**, *TCS* 82 (1989) 221-242.

[**11**] Filippo Mignosi & P. Séébold **Morphismes sturmiens et règles de Rauzy**, *J. Théorie des Nombres de Bordeaux* 5 (1993) 221-233.

[**12**] Marston Morse & Gustav A. Hedlund, **Symbolic dynamics II: Sturmian trajectories** *Amer. J. Math.* 62 (1940) 1-42.

[**13**] G. Rauzy, **Mots infinis en arithmétique**, *Automata on Infinite Words*, M. Nivat & D. Perrin, eds., *Lecture Notes in Computer Science* 192, Springer-Verlag (1984) 165-171.

[**14**] Kenneth B. Stolarsky, **Beatty sequences, continued fractions, and certain shift operators**, *Canad. Math. Bull.* 19-4 (1976) 473-481.

[**15**] Axel Thue, **Über unendliche zeichenreihen**, *Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana* 7 (1906) 1-22.

ACKNOWLEDGEMENTS

The work of the first author was supported in part by Grant No. OGP0025112 of the Natural Sciences & Engineering Research Council of Canada (NSERC), that of the third by NSERC Grant No. A8180 and by Grant No. GO-12278 of the Canadian Genome Analysis & Technology agency. We would like to thank both of the anonymous referees for many helpful comments; in particular, one for drawing to our attention the 1875 paper by Christoffel, the other the 1990 paper by Dulucq & Gouyou-Beauchamp.