Two-pattern strings

F.Franek, J. Jiang, W. Lu, and W.F. Smyth

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

Motivation: Investigate periodicities in strings, in particular tandem repetitions in binary strings that can be computed in linear time.

1997 *Iliopoulos, Moore, Smyth* Squares in (a finite fragment of) Fibonacci string can be computed in linear time.

Fibonacci (infinite) string:

$$F_0 = b, F_1 = a, F_n = F_{n-1}F_{n-2}$$
$$abaababaabaab.........$$

Fibonacci string is a special case of infinite Sturmian strings: balanced and not ultimately periodic $\equiv$ consisting of short blocks $a^i b$ and long blocks $a^{i+1}b$.

2000 *Franek, Karaman, Smyth* Repetitions in finite fragments of Sturmian strings can be computed in linear time.

A characteristics of Sturmian strings: replace each short block by a and each long block by b,

you again get a Sturmian string.

This RECURSIVE nature of Sturmian strings was used for linear time recognition of finite Sturmian strings and repetitions algorithms.

Our research focused on as wide generalization of Sturmian strings as possible while still preserving linear time recognition and computation of repetitions.

Meanwhile: 1997 *Farach* $\Theta(n)$ computation of suffix tree
1977-78 *Ziv, Lampel* $\Theta(n)$ computation of s-factorization
1989 *Main* $\Theta(n)$ computation of

runs
2000 *Kolpakov, Kucherov*
at most $O(n)$ runs in a string based on s-factorization.

Thus, in essence there is a linear way of computing all the repetitions. But the actual algorithm never given and never investigated as to its practicality (especially the space complexity of Farach's algorithm is very large). Also does not give any insight into the periodicity.

Therefore, understanding strings that have a "natural" linear time repetitions algorithm is still useful. This was an additional mo-

tivation for our research.

Finite Sturmian strings are characterized by REDUCTION PROPERTY: replace each short block by a and each long block by b and you get a shorter Sturmian string. Repeating (recursively) this, it can be reduced to a single letter.

We generalized it to *two-pattern* strings: a string consists of short blocks $p^i q$ and long blocks $p^j q$ ($i < j$) for some strings p and q (p and q must be sufficiently distinct). Reducing it by replacing each short block by a and each long block by b, we again get

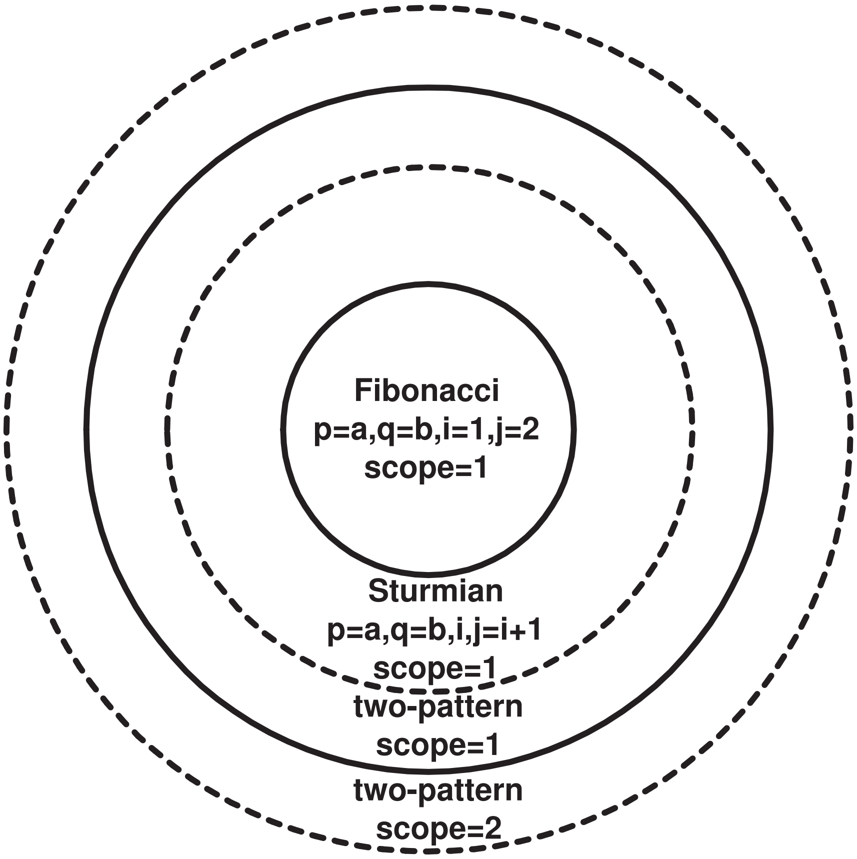
a two-pattern string or a single letter string.

This recursive definition is in the heart of the recognition algorithm for the input string x:

1. If $|x| = 1$, output TRUE.
2. Find (canonical) p, q, i, j so that the string x is a concatenation of short blocks $p^i q$ and long blocks $p^j q$. If found, output $[p, q, i, j]$, otherwise output FAIL.
3. Reduce x to y by replacing each $p^i q$ by a and each $p^j q$ by b.
4. Repeat recursively with the new string y as input.

Since each reduction shortens the input string by at least a half, it is a linear time algorithm. When a two-pattern string is recognized, its *reduction sequence* is output as well.

Of course, searching for possible p and q may be a problem, in order to keep it linear, we introduce an outside parameter, λ, the *scope* and we require that all p's and all q's are of size $\leq \lambda$. If x is a two-pattern string with scope $\lambda \leq \delta$, then x is a two-pattern string with scope δ.



Consider $\boldsymbol{x=}$ $\boldsymbol{(abb)aa(abb)^4aa(abb)aa(abb)^4 aa(abb)^4aa(abb)aa.}$

Set $\boldsymbol{p = abb, q = aa, i = 1, j = 4.}$

Reduce, and we get $\boldsymbol{(ab)^2ba.}$

Set $\boldsymbol{p = ab, q = ba, i = 2, j = 3.}$

Reduce, and we get a.

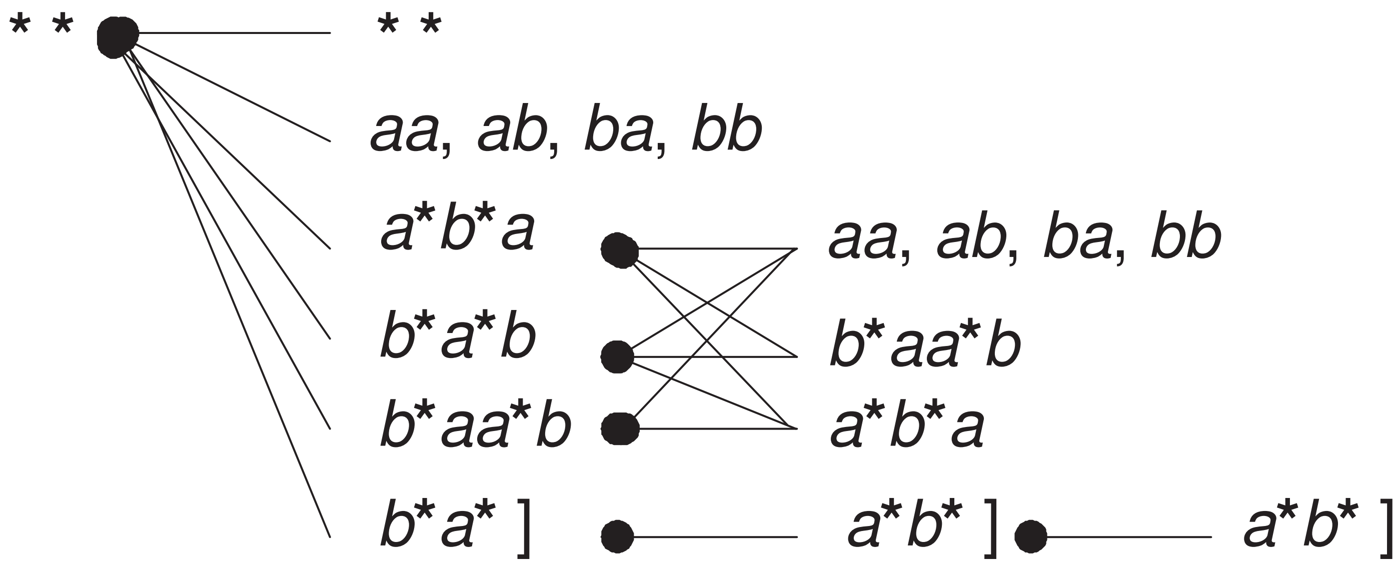
Thus, $\{[ab, ba, 2, 3], [abb, aa, 1, 4]\}$ is the reduction sequence of x (may be considered a compressed form of x) and x is two-pattern string with scope 3.

When searching for suitable p and q of size $\leq \lambda$, we may find more than one solution. It is conceivable that some will not lead to a single letter reduction and some will. This stumbling block is alleviated by the following theorem:

x is a two-pattern string iff there is a sequence of canonical reductions.

Thus, the recognition algorithm only searches for canonical p's and q's (in a sense leading to the "shortest" sequence of reductions).

To compute repetitions, we fully utilize the reduction sequence as the primary information of the structure of the string. All repetitions (runs) in x arise from certain linear configurations or runs on lower level of reduction y. So they can be computed and tallied recursively along the "construction" of the string (as given by the reduction sequence) and propagated from the lower level to the higher level.


* *
* *
aa, ab, ba, bb
a*b*a
b*a*b
b*aa*b
b*a*]
aa, ab, ba, bb
b*aa*b
a*b*a
a*b*]
a*b*]

An illustration of expansion of runs

...b a a b...

p=abba, q=aba
i = 1, j = 2

...*abba abba aba* abba aba abba aba *abba abba aba*...

(baabaa)^4 and 4 right shifts (right rotations)

Complicated? Yes, but there is a formula!

Conclusion and future work

The project on generalization of Sturmian strings proved interesting and fruitful. For two graduate students (*W. Lu* and *J. Jiang*) who co-authored this paper, this research constituted a significant part of their graduate work.

At the moment, only complete two-pattern strings have been dealt with. It is our aim to extend the results reported here to incomplete two-pattern strings, similarly as we did for Sturmian

strings.

Papers related to this topic, including these slides, can be viewed at the web site

www.cas.mcmaster.ca/~franek