A New Approach to Regular & Indeterminate Strings

Felipe A. Louza[a] Neerja Mhaskar[b] W. F. Smyth[b,c,d]

[a] Dept. of Computing and Mathematics, University of Sao Paulo, Brazil

[b] Dept. of Computing and Software, McMaster University, Canada

[c] Dept. of Informatics, King's College London, UK
[d] School of Engineering & Information Technology, Murdoch University, Perth, Australia

LSD & LAW 2019, London, UK

Outline

- Abstract
- Regular and Indeterminate Strings
- Palindromes and Maximal Palindrome Array
- Open Problems

Abstract

We propose a new, more appropriate definition of a regular string; that is, one that is isomorphic to a string whose entries all consist of a single letter. A string that is not regular is said to be indeterminate. We describe an algorithm to determine whether or not a string $\boldsymbol{x}$ is regular and, if so, to replace it by a lexicographically least string string $\boldsymbol{y}$ whose entries are all single letters. We then introduce the idea of a feasible palindrome array MP of a string, and show that every feasible MP corresponds to some (regular or indeterminate) string – perhaps, surprisingly, both! We describe an algorithm that constructs a string $\boldsymbol{x}$ corresponding to given feasible MP, lexicographically least whenever $\boldsymbol{x}$ is regular.

Introduction

The idea of a string as something other than a sequence of single letters has been discussed for almost half a century.

In 1974 Fischer & Paterson [FP74] studied pattern-matching on strings $\boldsymbol{x}$ whose entries could be **don't-care** letters; that is, letters matching any single letter in the alphabet Σ on which the string is defined, hence matching every position in $\boldsymbol{x}$.

In 1987 Abrahamson [Abr87] extended this model by considering pattern-matching on **generalized** strings whose entries could be arbitrary subsets of Σ.

Both of these models have been intensively studied in this century, notably by Blanchet-Sadri ("strings with holes") and Iliopoulos ("degenerate strings").

Regular & Indeterminate Strings

In this paper we redefine an indeterminate string in a context that we believe captures the idea in a more appropriate way — at once more general and more precise.

A **letter** ℓ is a finite list of s distinct **characters** $c_1, c_2, \ldots, c_s$, each drawn from a set Σ of size $\sigma = |\Sigma|$ called the **alphabet**.

In the case that Σ is ordered, ℓ is said to be in **normal form** if its characters occur in the ascending order determined by Σ.

The integer $s = s(\ell)$ is called the **scope** of ℓ. For $s = 1$, ℓ is said to be **regular**, otherwise **indeterminate**.

Two letters ℓ_1, ℓ_2 are said to **match**, written $\ell_1 \approx \ell_2$, if and only if $\ell_1 \cap \ell_2 \neq \emptyset$. In the case that matching ℓ_1 and ℓ_2 are both regular, we may write $\ell_1 = \ell_2$.

For $n \geq 1$, a **string** $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a sequence $\boldsymbol{x}[1], \boldsymbol{x}[2], \ldots, \boldsymbol{x}[n]$ of letters, where $n = |\boldsymbol{x}|$ is the **length** of $\boldsymbol{x}$, and every $i \in 1..n$ is a **position** in $\boldsymbol{x}$.

If every letter in $\boldsymbol{x}$ is in normal form, then $\boldsymbol{x}$ itself is said to be in **normal form**.

A tuple $T = (i, j_1, j_2)$ of distinct positions i, j_1, j_2 in $\boldsymbol{x}$ such that

$$\boldsymbol{x}[j_1] \approx \boldsymbol{x}[i] \approx \boldsymbol{x}[j_2]$$

is said to be a **triple**.

A triple T is **transitive** if $\boldsymbol{x}[j_1] \approx \boldsymbol{x}[j_2]$, otherwise **intransitive**.

If every triple T in $\boldsymbol{x}$ is transitive, then we say that $\boldsymbol{x}$ is **regular**; otherwise, $\boldsymbol{x}$ is **indeterminate**.

The **scope** of $\boldsymbol{x}$ is given by

$$S(\boldsymbol{x}) = \max_{i \in 1..n} s(\boldsymbol{x}[i]).$$

Two strings $\boldsymbol{x}$ and $\boldsymbol{y}$ of equal length n are said to be **isomorphic** if and only if for every $i, j \in 1..n$,

$$\boldsymbol{x}[i] \approx \boldsymbol{x}[j] \Longleftrightarrow \boldsymbol{y}[i] \approx \boldsymbol{y}[j]. \tag{1}$$

Lemma (1)

Every regular string is isomorphic to a string of scope 1.

Lemma (2)

Given a regular string $\boldsymbol{x}[1..n]$, then, corresponding to every triple (i, j_1, j_2), we can assign a regular letter to $\boldsymbol{y}[i], \boldsymbol{y}[j_1], \boldsymbol{y}[j_2]$ in such a way that the resulting string $\boldsymbol{y}[1..n]$ is isomorphic to $\boldsymbol{x}[1..n]$.

We propose the algorithm (function $regular$) outlined below to determine whether a given string $\boldsymbol{x}[1..n]$ on alphabet Σ is regular.

If $\boldsymbol{x}$ is regular, on exit the string $\boldsymbol{y}$ is the lex-least regular string of scope 1 on the integer alphabet $\Sigma' = \{1, 2, \ldots, \sigma'\}$ that is isomorphic to $\boldsymbol{x}$.

The runtime complexity of $regular$ is $\mathcal{O}(n^2\sigma^2)$

Function $regular$

Input: String $\boldsymbol{x}[1..n]$

Output: If $\boldsymbol{x}$ is regular, returns $true$; otherwise, $false$.

(And if $\boldsymbol{x}$ is regular, also constructs a lex-least string $\boldsymbol{y}[1..n]$.)

Outline of function $regular$

- Initialize each letter in $\boldsymbol{y}[1..n]$ to 0.
- Scan $\boldsymbol{x}$ from left to right, using $\boldsymbol{y}$ to record previous matches.
- During this scan the following condition holds as long as $\boldsymbol{x}$ is regular:

$$C : \boldsymbol{x}[i] \approx \boldsymbol{x}[j] \Leftrightarrow \boldsymbol{y}[i] = \boldsymbol{y}[j] \wedge \boldsymbol{y}[i] \neq 0.$$

- If at a position $i \in 1..n$, we have $\boldsymbol{y}[i] = 0$ — that is, it was not part of a previous match — we fill it with a new character σ'.
- We then scan the rest of the strings $\boldsymbol{x}[i+1..n]$ and $\boldsymbol{y}[i+1..n]$ to see if condition C continues to hold. If it does not, we mark $\boldsymbol{x}$ as indeterminate and exit; otherwise, whenever $\boldsymbol{x}[j] \approx \boldsymbol{x}[i]$ and $\boldsymbol{y}[j] = 0$, we assign $\boldsymbol{y}[j] \leftarrow \sigma'$.

Palindromes

A **substring** $\boldsymbol{u} = \boldsymbol{x}[i..j]$, $1 \leq i \leq j \leq n$, of length $\ell = j-i+1$ is said to be a **palindrome** if $\boldsymbol{x}[i+h] \approx \boldsymbol{x}[j-h]$ for every $h \in 0..\lfloor \ell/2 \rfloor$.

A palindrome $\boldsymbol{u} = \boldsymbol{x}[i..j]$ is said to be a **maximal palindrome** if one of the following holds: $i = 1$, $j = n$, or $\boldsymbol{x}[i-1] \not\approx \boldsymbol{x}[j+1]$.

The **centre** of a palindrome $\boldsymbol{u}$ is at position $i+\frac{\ell-1}{2}$.

Since this is not an integer for odd ℓ, we form the string $\boldsymbol{x}^*$, where $\# \notin \Sigma$ and $m = 2n+1$.

$$\boldsymbol{x}^*[1..m] = \#x_1\#x_2\#\cdots\#x_n\#,$$

Now every palindrome in $\boldsymbol{x}^*$ has an integer centre c. We call $d = 2\ell+1$ the **diameter** and $r = \lfloor d/2 \rfloor$ the **radius** of a palindrome in $\boldsymbol{x}^*$.

Maximal Palindrome Array

We can now define the **maximal palindrome array** $\mathsf{MP} = \mathsf{MP}_{\boldsymbol{x}^*}$ of $\boldsymbol{x}^*$:

For every $i \in 1..m$, if $\boldsymbol{x}^*[i] = \#$ and $\boldsymbol{x}^*[i{-}1] \not\approx \boldsymbol{x}^*[i{+}1]$, then $\mathsf{MP}[i] = 0$ (radius zero); otherwise, $\mathsf{MP}[i] \geq 1$ is the radius of the maximal palindrome centred at position i.

For example, $\mathsf{MP}_{\boldsymbol{x}^*}$ derived from $\boldsymbol{x} = aabac$ is as follows:

$$\begin{array}{rccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \boldsymbol{x}^* = & \# & a & \# & a & \# & b & \# & a & \# & c & \# \\ \mathsf{MP}_{\boldsymbol{x}^*} = & 0 & 1 & 2 & 1 & 0 & 3 & 0 & 1 & 0 & 1 & 0 \end{array} \tag{2}$$

The most general form of the palindrome array is given by

$$\mathsf{MP} = 0 i_2 i_3 \cdots i_{m-1} 0, \tag{3}$$

where for every $j \in 2..m-1$:

(a) $i_j \in (1-j \bmod 2)..\min(j-1, m-j)$;

(b) i_j is odd if and only if j is even.

Any array satisfying (3) is said to be **feasible**.

Lemma (3)

There exists a string corresponding to every feasible palindrome array.

To prove the above lemma, we construct a string $\boldsymbol{x}^*$ corresponding to the input feasible array $\text{MP} = \text{MP}_{\boldsymbol{x}^*}[1..m]$ as follows:

- First, for every odd $c \in 1..m$, we assign $\boldsymbol{x}^*[c] \leftarrow \#$, while every even position remains empty.
- For every $c \in 3..m{-}2$ such that $\text{MP}[i] = r \geq 2$, we add a unique character to each pair of positions $c{-}k, c{+}k$ in $\boldsymbol{x}^*$, where
  - (c even, r odd) $k = 2, 4, \ldots, r{-}1$;
  - (c odd, r even) $k = 1, 3, \ldots, r{-}1$.
- Finally, we assign a unique character to each position that remains empty.

Example of construction for the previous Lemma:

$$\begin{array}{rccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \boldsymbol{x}^* = & \# & a & \# & \{a,b\} & \# & c & \# & b & \# & d & \# \\ \mathsf{MP}_{\boldsymbol{x}^*} = & 0 & 1 & 2 & 1 & 0 & 3 & 0 & 1 & 0 & 1 & 0 \end{array} \quad (4)$$

However, as we have seen in (2), the regular string $\#a\#a\#b\#a\#c\#$ has the same palindrome array: a palindrome array can correspond to both a regular and an indeterminate string!

Forbidden pair: To each position $c \in 1..m$ of a feasible MP array we associate a pair of integers (i, j) such that

$$i = c - \text{MP}[c] - 1, \text{ and } j = c + \text{MP}[c] + 1.$$

Then, provided $0 < i, j < m + 1$, we must have $\boldsymbol{x}^*[i] \not\approx \boldsymbol{x}^*[j]$. We call (i, j) the **forbidden pair** with respect to c.

Assuming that $\boldsymbol{x}^*[0] = \boldsymbol{x}^*[m+1] = \emptyset$, we denote by FP the set of all forbidden pairs with respect to each centre $c \in 1..m$.

Example of forbidden pair for each centre c:

$$
\begin{array}{cccccccc}
c & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
MP & 0 & 1 & 0 & 3 & 2 & 1 & 0 \\
\boldsymbol{x}^* & \# & 1 & \# & \{2,3\} & \# & \{1,3\} & \# \\
\text{FP} & (0,2) & (0,4) & (2,4) & (0,8) & (2,8) & (4,8) & (6,8)
\end{array} \tag{5}
$$

To characterize MP arrays we use Manacher's condition [Man75], restated in [IIBT10], and restated again below:

In MP $=$ MP$_{\boldsymbol{x}^*}$, we consider each centre c of a palindrome of radius $r =$ MP$[c]$, where for c even, $r \geq 3$, and for c odd, $r \geq 2$.

Then we must have $\boldsymbol{x}^*[c{-}k] \approx \boldsymbol{x}^*[c{+}k]$, where $1 \leq k \leq r$. For each k, let $r_\ell =$ MP$[c{-}k]$, $r_r =$ MP$[c{+}k]$. We then have

Definition (Manacher's condition)

Every position c in MP $=$ MP$_{\boldsymbol{x}^*}$ satisfies the following:

(a) **if** $r_\ell \neq rk$ **then** $r_r = \min(r_\ell, rk)$ **else** $r_r \geq r_\ell$;

(b) **if** $r_r \neq rk$ **then** $r_\ell = \min(r_r, rk)$ **else** $r_\ell \geq r_r$.

In [Man75] it is shown that, for every palindrome in string $\boldsymbol{x}^*$ such that $S(\boldsymbol{x}^*) = 1$, Manacher's condition must hold.

Thus, by Lemma (1), Manacher's condition holds for every regular string; that is, for every string whose triples are all transitive.

We propose procedure $construct$ which on an input MP produces a lex-least regular string if MP is regular; otherwise produces an indeterminate string.

Procedure *construct*

Input: Feasible maximal palindrome array MP$[1..m]$.

Main Data Structure: FS$[1..m]$ — an array giving forbidden pairs (i, j) that have been identified w.r.t centres c: for every (even) i and $\forall j \in$ FS$[i]$, $\boldsymbol{x}^*[i] \not\approx \boldsymbol{x}^*[j]$.

Outline of procedure *construct*

- For every odd $i \in 1..m$, assign $\boldsymbol{x}^*[i] \leftarrow \#$; for every even $j \in 1..m$, assign $\boldsymbol{x}^*[j] \leftarrow 0$. Set $\boldsymbol{x}^*[2] \leftarrow 1$.
- For each centre $c \in 3..m-1$, compute its forbidden pair (i, j) and update the FS$[i]$ and FS$[j]$ sets accordingly.

- If the string $\boldsymbol{x}^*[1..c-1]$ constructed so far is regular and the centre c and range k satisfy Manacher's condition, we continue to construct the regular string by copying the previously filled letter $\boldsymbol{x}^*[c-k]$ to its corresponding matching position $c+k$.
- Whenever there is a choice of filling an empty position — that is, when $\boldsymbol{x}^*[c] = 0$ — the lex-least character which is not in the forbidden set of characters FS$[c]$ is chosen.
- If a given centre c and range k do not satisfy Manacher's condition, we mark the string as indeterminate, and every subsequent letter match including the current one is achieved by adding a new character to $\boldsymbol{x}^*[c-k]$ and $\boldsymbol{x}^*[c+k]$, where $c-k$ and $c+k$ are even.

Example 1 ($MP[12] = 3$):

$$\begin{array}{rccccccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ MP = & 0 & 1 & 0 & 3 & 0 & 1 & 0 & 7 & 0 & 1 & 0 & 3 & 0 & 1 & 0 \\ \boldsymbol{x}^* = & \# & 1 & \# & 2 & \# & 1 & \# & 3 & \# & 1 & \# & 2 & \# & 1 & \# \end{array} \quad (6)$$

$\boldsymbol{x}^*$ is the string produced by $construct$.

The input MP array is regular, therefore the resulting string $\boldsymbol{x}^*$ is regular.

Example 2 ($MP[12] = 1$):

$$\begin{array}{lccccccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & \mathbf{12} & 13 & 14 & 15 \\ MP = & 0 & 1 & 0 & 3 & 0 & 1 & 0 & 7 & 0 & 1 & 0 & \mathbf{1} & 0 & 1 & 0 \\ \boldsymbol{x}^* = & \# & 1 & \# & \{2,3\} & \# & \{1,4\} & \# & 5 & \# & 4 & \# & 3 & \# & 1 & \# \end{array} \quad (7)$$

$\boldsymbol{x}^*$ is the string produced by $construct$.

The input MP array is not regular, therefore the resulting string $\boldsymbol{x}^*$ is indeterminate.

Results

Lemma (4)

Let $\boldsymbol{x}^*$ be the string produced by procedure $construct$. Then $S(\boldsymbol{x}^*) = 1 \Leftrightarrow \boldsymbol{x}^*$ is regular.

Theorem (1)

Let $\boldsymbol{x}^*$ be the string produced by the procedure $construct$ on an input MP. Then $\boldsymbol{x}^*$ is regular $\Leftrightarrow$ MP is regular.

Theorem (2)

Given an MP array of length $m = 2n+1$, procedure $construct$ executes in $\mathcal{O}(n^2\sigma)$ time, where σ is the size of the generated alphabet.

Open Problems:

1. Procedures $regular$ and $construct$ both have worst-case time complexity $\mathcal{O}(n^2)$. It should be possible to reduce this.
2. Given the new definition of regularity in strings, what is its effect on the computation of data structures such as border array, cover array, prefix array, suffix array, Lyndon array?
3. Is it possible to compute the above mentioned data structures without first using the $\mathcal{O}(n^2)$ time function $regular$?
4. What effect does this new definition have on the "reverse engineering" problem related to these arrays?
5. There exist algorithms to reverse engineer the other arrays noted above — can they be modified/extended to yield equivalent results?

References I

[Abr87] K. Abrahamson.
Generalized string matching.
SIAM Journal of Computing, 16(6):1039–1051, 1987.

[FP74] M.J. Fischer and M.S. Paterson.
String matching and other products.
In R.M. Karp, editor, *Complexity of Computation,*, pages 113–125. American Mathematical Society, 1974.

[IIBT10] Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda.
Counting and verifying maximal palindromes.
Proc. 17th Symposium on String Proocessing & Information Retrieval (SPIRE), LNCS 6393, pages 135–146, 2010.

References II

[Man75] G. Manacher.

Anew Linear– Time “ On– Line” Algorithm for Finding the Smallest Initial Palindrome of a String.

J. Assoc. Comput. Mach., 22:346–351, 1975.