

# Tighter Upper Bounds on The Exact Complexity of String Matching\*

Richard Cole<sup>†</sup>      Ramesh Hariharan<sup>‡</sup>

September 18, 1996

## Abstract

This paper considers how many character comparisons are needed to find all occurrences of a pattern of length  $m$  in a text of length  $n$ . The main contribution is to show an upper bound of the form  $n + O(n/m)$  character comparisons, following preprocessing. Specifically, we show an upper bound of  $n + \frac{8}{3(m+1)}(n - m)$  character comparisons. This bound is achieved by an online algorithm which performs  $O(n)$  work in total, requires  $O(m)$  space and  $O(m^2)$  time for preprocessing. The current best lower bound for online algorithms is  $n + \frac{16}{7m+27}(n - m)$  character comparisons for  $m = 16k + 19$ , for any integer  $k \geq 1$ , and for general algorithms is  $n + \frac{2}{m+3}(n - m)$  character comparisons, for  $m = 2k + 1$ , for any integer  $k \geq 1$ .

## 1 Introduction

String matching is the problem of finding all occurrences of a pattern  $p[1 \dots m]$  in a text  $t[1 \dots n]$ . We assume that the characters in the text are drawn from a general (possibly infinite) alphabet unknown to the algorithm. We investigate the time complexity of string matching measuring both the exact number of comparisons and the time complexity counting all operations. As is standard, the time complexity refers to operations performed following preprocessing of the pattern; preprocessing of the text is not allowed. Our goal is to minimize the number of comparisons, while still maintaining a total linear time complexity and a polynomial in  $m$  preprocessing cost.

Note that if the algorithm is permitted to know the alphabet, then there is a finite automaton which performs string matching by reading each text character exactly once (which can be obtained from the failure function of [KMP77]). However, in this case the running time depends on the alphabet size.

Perhaps the most widely known linear time algorithms for string matching are the Knuth-Morris-Pratt [KMP77] and Boyer-Moore [BM77] algorithms. We refer to them as the KMP and BM algorithms, respectively. The KMP algorithm makes at most  $2n - m + 1$  comparisons and this bound is tight. The exact complexity of the BM algorithm was an open question until recently. It was shown in [KMP77] that the BM algorithm makes at most  $6n$  comparisons if

---

\*The work was supported in part by NSF grants CCR-8902221 and CCR-8906949.

<sup>†</sup>Courant Institute of Mathematical Sciences, New York University.

<sup>‡</sup>Max-Planck Institut für Informatik, Germany. This work was done when the author was at Courant Institute of Mathematical Sciences, New York University.

the pattern does not occur in the text. Guibas and Odlyzko [GO80] reduced this to  $4n$  under the same assumption. Cole [Co91] finally proved an essentially tight bound of  $3n - \Omega(n/m)$  comparisons for the BM algorithm, whether or not the pattern occurs in the text. Colussi [Col91] gave a simple variant of the KMP algorithm which makes at most  $\frac{3}{2}n$  comparisons. Apostolico and Giancarlo [AG86] gave a variant of the BM algorithm which makes at most  $2n - m + 1$  comparisons. Crochemore et al. [CCG92] showed that remembering just the most recently matched portion reduces the upper bound of BM from  $3n$  to  $2n$  comparisons.

Recently, Galil and Giancarlo [GG92] gave a string matching algorithm which makes at most  $\frac{4}{3}n$  comparisons. This was the strongest upper bound for string matching known prior to our work. In fact, [GG92] give this bound in a sharper form as a function of the period  $z$  of the pattern; the bound becomes  $n + \min\{\frac{1}{3}, \frac{\min\{z, m-z\}+2}{2m}\}(n-m)$ .

Galil and Giancarlo [GG91] gave a lower bound of  $n(1 + \frac{1}{2^m})$  comparisons. For online algorithms [GG91] showed an additional lower bound of  $n(1 + \frac{2}{m+3})$ . An online algorithm is an algorithm which examines text characters only in a window of size  $m$  sliding monotonically to the right; further the window can slide to the right only when all matching pattern instances to the left of the window or aligned with the window have been discovered. Recently, Zwick and Paterson gave additional lower bounds, including a bound of  $\frac{4n}{3}$  for patterns of length 3 in the general case [ZP92].

Our contribution is a linear time online algorithm for string matching which makes at most  $n(1 + \frac{8}{3(m+1)})$  character comparisons. Our algorithm requires  $O(m)$  space and  $O(m^2)$  preprocessing time and runs in  $O(m+n)$  time overall (exclusive of preprocessing). Independently, Breslauer and Galil discovered a similar algorithm which performs at most  $n + O(\frac{n \log m}{m})$  comparisons [BG92]; this algorithm requires  $O(m)$  preprocessing space and time and runs in linear time. Recently, Hancart [Ha93] and Breslauer et al. [BCT93] have independently shown an upper and lower bound of  $(2 - \frac{1}{m})n$  on the number of comparisons required for string matching when comparisons must involve only text characters in a window of size one sliding monotonically to the right.

Nearly matching lower bounds are given in a companion paper [CHPZ92]. They show the following bounds: for online algorithms, a bound of  $n + \frac{16}{7m+27}(n-m)$  character comparisons for  $m = 16k + 19$ , for any integer  $k \geq 1$ , and for general algorithms a bound of  $n + \frac{2}{m+3}(n-m)$  character comparisons, for  $m = 2k + 1$ , for any integer  $k \geq 1$ .

Even if exponential (in  $m$ ) preprocessing and exponential space are available, it is not clear that the above upper bound can be achieved (assuming a result independent of the alphabet size is sought). The difficulty is that text characters which are mismatched may need to be compared repeatedly. In order to minimize the total number of comparisons this has to be offset by other text characters not needing to be compared. The hardest patterns to handle are those which have proper suffixes which are also prefixes of the pattern. We refer to such substrings as *presufs*<sup>1</sup>. Our algorithm has two parts; a basic algorithm and a presuf handler. The basic algorithm handles primary patterns, i.e., patterns with no presufs; this is also the core of the algorithm for the general case. The presuf handler copes with presufs; its design constituted the main challenge in this work. Understanding the structure of the presufs was a key ingredient in its design. Understanding this structure also led to the new lower bound constructions given in [CHPZ92].

The flavor of the algorithm is as follows. Initially, the pattern is aligned with the left end of the text. Repeatedly, an attempt to match the pattern against the text is made. When a

---

<sup>1</sup>Presufs are also called *borders* in literature. Strings without presufs are called *primary* strings.

mismatch is found or the pattern is fully matched the pattern is shifted to the right. The goal is to maximize this shift without missing any possible matches. The basic algorithm has the property that the length of each shift is at least equal to the number of comparisons since the previous shift (or the start of the algorithm). This results in an algorithm that performs at most  $n$  comparisons if the pattern has no presuf (the algorithms of [GG92] and [CP89] have this property too).

The presuf handler cannot quite match the performance of the basic algorithm (which is not surprising given that the lower bounds for this problem are larger than  $n$  comparisons). Here the approach is to follow the basic algorithm until a suffix which is also a prefix is matched. The only possible matches in which a new instance of the pattern overlaps the current partially (or fully) matched instance arise with an overlap by a presuf. Ignoring, for the moment, problems introduced by periodic patterns, it is the case that at most one of these overlapping pattern instances can result in a match. An elimination is performed to determine which one, if any, of the overlapping pattern instances might result in a match. Following this elimination, a further non-trivial sequence of comparisons is made; this can lead to one of two situations: another match of a suffix which is also a prefix, or a mismatch which causes a return to the basic algorithm. The presuf handler is invoked at most once for every  $\frac{m}{2}$  text characters, and performs a number of comparisons at most 2 greater than the number of characters shifted over (actually, there are two possible scenarios: an invocation after  $\frac{3}{4}m$  text characters and at most 2 excess comparisons, or an invocation after  $\frac{m}{2}$  text characters and at most 1 excess comparison). Periodic patterns have the added difficulty that the presuf handler could be invoked more frequently. In this case, we show the additional fact that if the presuf handler is invoked after fewer than  $\frac{m}{2}$  text characters then the number of comparisons is at most the number of characters shifted over.

This structure of the algorithm of Breslauer and Galil is similar; their analog of the presuf handler works in a completely different way, however.

Section 2 provides several definitions. The basic algorithm is described in Section 3. In Section 4 the presuf handler for non-periodic strings is presented. Section 5 gives a technical construction deferred from Section 4. Finally, in Section 6, the result is extended to periodic patterns.

We remark here that the properties of strings which we develop in Section 4 and later are mostly new and appropriate references are given otherwise.

## 2 Definitions and Preliminaries

A string  $v$  is a *presuf* of  $p$  if it is both a proper suffix and a proper prefix of  $p$ . Let  $x$  be the length of the largest presuf of  $p$ . The *period* of a pattern  $p$  with length  $m$  is defined to be  $m - x$ .  $x$  is called the *s-period* (or shift period) of  $p$ . A string  $p$  is *cyclic* in string  $v$  if it is of the form  $v^k$ ,  $k > 1$ . A *primitive* string is a string which is not cyclic in any string.

A string  $p$  is *periodic* if  $p = wv^k$ , where  $w$  is a (possibly null) proper suffix of  $v$  and  $k > 1$ . The smallest such  $v$  is called the *core* of  $p$  and the corresponding  $w$  is called the *head* of  $p$ . Note that the core is primitive. A *cyclic shift* of  $p$  is any string  $vu$  where  $p = uv$ .  $|v$  and  $|v|$  refer respectively to the leftmost and rightmost characters in string  $v$ ; on occasion, we will call these characters respectively, the left end and right end of  $v$ . Two characters are said to be *distance*  $d$  apart if they are separated by  $d - 1$  other characters.

For the rest of the paper, let  $p$  be a pattern with length  $m$ . Let the text  $t$  have length  $n$ .  $p[i]$

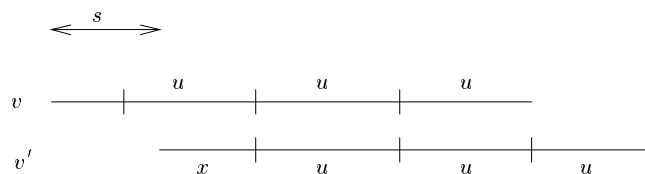


Figure 1: Periodicity

denotes the  $i$ th character of  $p$ , reading from the left end;  $i$  is called the *index* of  $p[i]$  in  $p$ . The same notation and terminology is used for string  $t$ .

The algorithm will be comparing the pattern with substrings of the text with which the pattern is aligned; as the algorithm proceeds, the pattern is shifted to the right across the text. Each possible alignment of the pattern with the text is called an *instance* of the pattern. Note that an instance is not necessarily an occurrence.

For each pair of overlapping instances of the pattern a location at which the two differ, if any, will be precomputed. This location is called the *difference point* of the two instances. Note, however, that for a given pair, a difference point may not exist; but this can happen only if the pattern has a non-empty presuf. Let  $p_1$  and  $p_2$  denote two pattern instances, where  $p_1[i]$  is aligned with  $p_2[1]$ ; then  $diff_i$  is the difference point if any; i.e.,  $p_1[diff_i] \neq p_2[diff_i - i + 1]$ .

Let  $q$  be a pattern instance. Those pattern instances to the right of  $q$ , overlapping  $q$ , but which do not have a difference point with  $q$  are called the *presuf overlaps* of  $q$ .

We quote a few standard results concerning strings.

**Lemma 2.1** *Let  $w$  be a presuf of string  $v$ . If  $|w| > |v|/2$  then  $v$  is periodic.*

**Proof:** See Fig.1. Let  $s = |v| - |w|$ . Let  $v'$  denote string  $v$  shifted distance  $s$  to the right. Then the portion of  $v'$  overlapping  $v$  is presuf  $w$  which matches the corresponding portion of  $v$ . Let  $u$  denote the suffix of  $v'$  of length  $s$ . An easy induction shows that  $v = xu^k$  for some  $k \geq 2$ , where  $x$  is a proper suffix of  $u$ .  $\square$

The following appear in different forms in [Lo82] (see Proposition 1.3.2, 1.3.4, and 1.3.5).

**Lemma 2.2** [LS62, FW65] *If  $x, y$  are two distinct periods of a string  $v$  such that  $x + y \leq m + \gcd\{x, y\}$  then  $\gcd\{x, y\}$  is also a period of  $v$ .*

**Lemma 2.3** *Suppose that  $v = xy$  where both  $x$  and  $y$  are presufs of  $v$ . Then  $v$  is cyclic in some string  $w$  of length  $\gcd\{|x|, |y|\}$ .*

**Lemma 2.4** *If  $v$  is periodic and can be expressed both as  $x_1u_1^{k_1}$  and  $x_2u_2^{k_2}$  where  $x_i$  is a suffix of  $u_i$ ,  $u_1 > u_2$  and  $k_1, k_2 \geq 2$ , then either  $u_1$  is cyclic in  $u_2$  or both  $u_1$  and  $u_2$  are cyclic in some smaller string.*

### 3 The Basic Algorithm

The algorithm in this section also appears in [Col91] and is also exposed in [GG92]. We describe it again for the sake of completeness.

If all the characters in  $p$  are identical then it is easily seen that the KMP algorithm makes at most  $n$  character comparisons. Further, if  $m = 2$  and  $p$  consists of two distinct characters then the Boyer-Moore algorithm makes at most  $n$  character comparisons. Henceforth, we assume that  $m > 2$  and that  $p$  has at least two distinct characters.

The algorithm proceeds by eliminating pattern instances as possible matches. It repeatedly performs the following two steps: first, it attempts to match the leftmost surviving pattern instance with the aligned text substring; then, it shifts to the next leftmost surviving pattern instance.

After a shift occurs the strategy followed depends on the nature of the shift. The order in which pattern characters are compared ensures that all the shifts satisfy one of the following two properties.

1. A shift has size greater than or equal to the number of comparisons made since the previous shift. This is called a *basic shift*.
2. When Property 1 is not true, a proper prefix  $x$  of  $p$  is completely matched with the text after the shift. Moreover,  $x$  is also a suffix of  $p$ . This is called a *presuf shift*.

Following a basic shift, the basic algorithm is continued; a presuf shift results in a transfer to the presuf handler.

The following observation is the key to the basic algorithm. Consider two overlapping instances of the pattern  $p$ . Then comparing either of the two pattern characters at their difference point with the aligned text character is sure to eliminate one of the two pattern instances from being a potential match. So long as the overlap is not a presuf of  $p$ , there will be a difference point. This is exactly the notion of *duelling* introduced by Vishkin [Vi85].

More formally, let  $p_a$  and  $p_b$  be the two leftmost surviving pattern instances, where  $p_b$  is not a presuf overlap of  $p_a$ . Let  $d$  be the difference point of  $p_a$  and  $p_b$ .  $p_a[d]$  is compared with the aligned text character. A match eliminates  $p_b$ ; a mismatch eliminates  $p_a$ .

Next, we give the exact sequence of comparisons made by the above strategy. We precompute the following sequence  $S$ .  $S$  is the sequence of indices  $dif_2, dif_3, \dots, dif_m$  omitting repetitions and undefined indices. Henceforth, where no ambiguity will result, we will use the sequence  $S$  to refer both to the indices it contains and to the corresponding characters in  $p_a$ .

The characters in  $p_a$  are compared with their corresponding text characters in two passes, stopping if a mismatch is found. In Pass 1, those characters in  $p_a$  contained in  $S$  are compared in sequence. If all of these match then the remaining pattern characters are compared from right to left in Pass 2.

**Lemma 3.1** *If a mismatch occurs at the character given by the  $k$ th index in  $S$  then the resulting shift has size at least  $k$ .*

**Proof:** Let the  $k$ th index in  $S$  be  $dif_l$ . Note that  $k < l$ . Recall that  $l \leq dif_l \leq m$  and  $p[dif_l] \neq p[dif_l - l + 1]$ . Suppose for a contradiction that the shift was of length  $j < k$ . Let  $p_a$  and  $p_b$  be the pattern instances as specified in the algorithm above, before this shift. Note that  $p_b$  becomes  $p_a$  after the shift; i.e.,  $p_b$  is  $p_a$  shifted  $j$  units. But then  $p_a$  and  $p_b$  have a difference point and hence  $dif_{j+1}$  is defined.  $dif_{j+1}$  is the  $i$ th index in  $S$ , for some  $i \leq j$ ; hence, as  $j < k$ ,  $dif_{j+1}$  occurs prior to  $dif_l$  in  $S$ . Therefore,  $p_a[dif_{j+1}]$  would have been matched against the text and one of  $p_a$  or  $p_b$  eliminated before  $p_a[dif_l]$  was compared. The contradiction proves the lemma.  $\square$

Consequently, all shifts resulting from mismatches in Pass 1 are basic shifts. When a basic shift is made the basic algorithm is restarted. It is easy to see that if all shifts are basic shifts then the total number of comparisons made is upper bounded by  $n$ .

Next, suppose that all comparisons in Pass 1 result in matches.

**Lemma 3.2** *Suppose Pass 2 results in a mismatch at  $p_a[l]$ . The resulting shift has length at least  $l$ .*

**Proof:** Suppose for a contradiction that the resulting shift has length  $i$ ,  $i < l$ . Let  $p_b$  be  $p_a$  shifted distance  $i$ . Then  $l$  is a difference point for  $p_a$  and  $p_b$ ; hence one of  $p_a$  and  $p_b$  would have been eliminated in Pass 1, a contradiction.  $\square$

Consequently, for each shift resulting from Pass 2 with length less than the number of comparisons made since the previous shift, a proper prefix of  $p$  (which is also a suffix of  $p$ ) is matched with the text, i.e., it is a presuf shift. The main challenge in minimizing the exact number of comparisons is to handle presuf shifts.

**Preprocessing.** The sequence  $S$ , as defined above, is not unique. We show that a particular instance of  $S$  can be precomputed in a manner akin to the computation of the KMP shift function or the BM shift function. The KMP shift function comprises, for each  $j$ ,  $1 < j \leq m$ , a number  $s_j$ .  $s_j$  is the largest  $i$ ,  $i < j$ , such that  $p[1 \dots i - 1] = p[j - i + 1 \dots j - 1]$  and  $p[i] \neq p[j]$ ; note that  $i = \text{dif}_{j-i+1}$ . If no such  $i$  exists then  $s_j$  is defined to be zero. Consider the set of all those values of  $j$  for which  $s_j > 0$ . Furthermore, let this set be ordered by the increasing value of  $j - s_j + 1$ . This provides the sequence  $S$ . For every  $k$ ,  $2 \leq k \leq m$ , if  $\text{dif}_k$  is defined, then, for some  $l \in S$ ,  $k \leq l \leq m$ ,  $p[1 \dots l - k] = p[k \dots l - 1]$  and  $p[l - k + 1] \neq p[l]$ ; hence the value  $\text{dif}_k$  occurs in  $S$  (though not necessarily indexed by  $k$ ). Finally, it is straightforward to compute  $S$  in  $O(m)$  time.

## 4 The Presuf Handler

In this section the presuf handler for non-periodic patterns  $p$  is described. This presuf handler also deals with some presuf shifts for periodic  $p$ , as specified in the next few paragraphs.

With each presuf shift, we associate a presuf  $x'_1$  of  $p$ , defined as follows. If  $p$  is not periodic then  $x'_1$  is the longest presuf of  $p$ . Otherwise, suppose  $p = u_p v_p^{i_p}$  is periodic with core  $v_p$  and head  $u_p$ . Then if the presuf of  $p$  matching the text is at least  $|v_p|$  long,  $x'_1 = u_p v_p^{i_p - 1}$ . Otherwise, if the above presuf is shorter than  $|v_p|$ , then  $x'_1$  is defined to be the longest presuf of  $p$  of length less than  $|v_p|$ .

In this section, we give an algorithm for handling presuf shifts for the case  $|x'_1| < \frac{m}{2}$ . The case  $|x'_1| \geq \frac{m}{2}$  is considered in Section 6. Note that  $|x'_1| < \frac{m}{2}$  always holds for non-periodic  $p$  and may hold for periodic  $p$ .

Consider the situation immediately following a presuf shift. Some prefix of  $p$ , which is also a presuf, matches the text substring it is aligned with. It is convenient for the presuf handler to assume that the pattern was shifted by  $m - |x'_1|$  characters and that  $x'_1$  matches the text. Note that this will not be the case if Pass 2 in the basic algorithm mismatches before  $x'_1$  is completely matched. A simple check will prevent the declaration of any incorrect complete match that might result from the above assumption. To facilitate this check, a variable  $t_{last}$  is used. Suppose Pass

2 in the basic algorithm ends in a mismatch. Then  $t_{last}$  is set to the index of the text character where the mismatch occurred. Otherwise, if no mismatch occurs,  $t_{last} \leftarrow \phi$ .

As  $|x'_1| < \frac{m}{2}$ ,  $p = x'_1 u x'_1$ , for some string  $u$ . Let  $t_A$  be the substring of the text aligned with the prefix  $x'_1$  of  $p$  immediately following the presuf shift; note that  $x'_1$  matches  $t_A$ . Order all the presufs of  $x'_1$  by decreasing length and let this order be  $x_1, x_2, x_3, \dots, x_k, x_{k+1}$ , where  $x_k$  is the smallest non-null presuf of  $x'_1$  and  $x_{k+1}$  is the null string and hence a trivial presuf of  $x'_1$ . Note that  $x_1 = x'_1$ . Let the future instances of  $p$  (i.e., potential match instances) before its left end slides beyond  $t_A$ , in left to right order, be  $p_1, p_2, \dots, p_k$ . Let  $p_{k+1}$  be the pattern instance whose left end is to the immediate right of  $t_A$ . Then  $p_i$ ,  $1 \leq i \leq k+1$ , is the pattern instance with the prefix  $x_i$  of  $p$  aligned with the suffix  $x_i$  of  $t_A$ .  $x_i$  is said to be the presuf associated with  $p_i$ .  $p_1, p_2, \dots, p_k, p_{k+1}$  are called the *presuf pattern instances*.

**Lemma 4.1** *If  $|x'_1| < \frac{m}{2}$  then at most one of  $p_1, \dots, p_k, p_{k+1}$  can lead to a complete match.*

**Proof:** By contradiction. Suppose some two of them, say  $p_i$  and  $p_j$ ,  $i < j$ , each result in a complete match. It follows that there is a prefix of  $p$  of size  $m - |x_i| + |x_j|$  that matches a suffix of  $p$ . Since  $|x_i| - |x_j| < m/2$ ,  $m - |x_i| + |x_j| > m/2$ ; also,  $|x_i| - |x_j| \leq |x'_1|$ . This implies that  $p$  is periodic with core of length at most  $|x'_1|$ , contrary to assumption.  $\square$

The presuf handler begins by eliminating all but at most one of  $p_1, p_2, \dots, p_k, p_{k+1}$ . This is carried out by a procedure that performs  $j \leq k$  comparisons; at most two of these comparisons are unsuccessful. We seek to minimize the number of unsuccessful comparisons because while successful comparisons can be remembered, unsuccessful comparisons may lead to repeated comparison of some text characters.

The elimination procedure is described in Section 4.1. The remainder of the presuf handler procedure, for all but two special cases, is given in Section 4.2 and its analysis is presented in Section 4.3. The special cases are handled in Section 4.4. Finally, data structure details are described in Section 4.5.

## 4.1 Elimination Strategy

Before describing the exact sequence of comparisons made by the elimination strategy, we need to understand some structural properties of these overlapping instances of  $p$ .

**Lemma 4.2** *Suppose  $x_i = uv^l$  is the  $i$ th presuf, where  $u$  is a proper suffix of  $v$ ,  $v$  is primitive, and  $l \geq 2$ . Then  $x_{i+1} = uv^{l-1}$ .*

**Proof:** Certainly  $uv^{l-1}$  is a presuf, so the only question is whether there is a presuf  $x$  between  $uv^l$  and  $uv^{l-1}$ . Suppose there is such an  $x$ . As  $|uv^{l-1}| < |x| < |uv^l|$  and as  $x$  is a prefix of  $uv^l$ , the suffix of  $x$  of length  $|v|$  is a cyclic shift of  $v$ . But  $x$  is a suffix of  $uv^l$ , which implies that a proper cyclic shift of  $v$  matches  $v$ . By Lemma 2.3,  $v$  is cyclic, contrary to assumption.  $\square$

**Lemma 4.3** *The presuf pattern instances can be partitioned into  $g = O(\log m)$  groups<sup>2</sup>  $A_1, A_2, \dots, A_g$ . The groups preserve the left to right ordering of the pattern instances; i.e., the pattern instances in group  $A_i$  are all to the left of those in group  $A_{i+1}$ , for  $i = 1, \dots, g-1$ . Let  $B_i$  be the set of presufs associated with the pattern instances in  $A_i$ . Then either  $B_i = \{u_i v_i^{k_i}, \dots, u_i v_i^3, u_i v_i^2\}$  or  $B_i = \{u_i v_i^{k_i}, \dots, u_i v_i\}$  or  $B_i = \{u_i v_i^{k_i}, \dots, u_i v_i, u_i\}$ , where  $k_i \geq 1$  is maximal,  $u_i$  is a proper suffix of  $v_i$ , and  $v_i$  is primitive.*

<sup>2</sup>Actually, a sharper bound of  $\log_\phi m$  groups is known [KMP77, B94], where  $\phi$  is the golden ratio.

**Proof:** The proof is by construction. The groups are constructed in left to right order. Inductively suppose  $A_i$  is being built presently and all presuf pattern instances with associated presufs longer than  $u_i v_i^{k_i}$  have been placed in groups to the left of  $A_i$ .

$\{u_i v_i^{k_i}, \dots, u_i v_i^2\}$  are all added to  $B_i$ .  $u_i v_i$  is also added if and only if it is not periodic; otherwise,  $u_i v_i$  starts set  $B_{i+1}$ . By Lemma 4.2 all presuf pattern instances with associated presufs longer than  $u_i v_i$  are in group  $A_i$  or by induction in a group to its left. In addition, if  $u_i$  is empty and  $v_i$  has no presufs then  $u_i$  is also added.

The maximality of  $k_i$  can be seen as follows. For suppose  $k_i$  is not maximal, i.e., there exists a presuf  $w$  of the form  $u_i v_i^{k_i+1}$ ,  $k_i + 1 \geq 2$ . By the inductive hypothesis describing the construction, this presuf would already be in one of the groups  $B_1, \dots, B_{i-1}$ . By Lemma 4.2, it follows that  $w$  is the smallest presuf in  $B_{i-1}$ .  $w$  is clearly periodic. By construction,  $w = u_{i-1} v_{i-1}^2 = u_i v_i^{k_i+1}$ ,  $k_i + 1 \geq 2$ . Then, by Lemma 2.4,  $v_{i-1}$  must be cyclic, which contradicts the assumption that  $v_{i-1}$  is primitive. Thus  $k_i$  must be maximal.

This shows that the presuf pattern instances are partitioned into groups. It remains to show that there are only  $O(\log m)$  groups. Let  $x_{j_i}$  be the leftmost presuf in  $B_i$ . If  $x_{j_{i+1}} = u_i v_i$ , then  $|x_{j_{i+1}}| \leq \frac{2}{3}|x_{j_i}|$ , and otherwise,  $|x_{j_{i+1}}| \leq \frac{1}{2}|x_{j_i}|$  (the latter claim follows because  $x_{j_{i+1}}$  is both a prefix and a suffix of  $x_{j_i}$  and this prefix and suffix are non-overlapping). The  $O(\log m)$  bound follows immediately.  $\square$

**Lemma 4.4** *The groups satisfy the following properties.*

*Property 1. Consider the presufs  $x_i$  corresponding to the pattern instances  $p_i$  in some group  $A_j$ . For  $j \neq g$ , all these presufs  $x_i$ , except possibly the rightmost one, are periodic with the same core and head. For  $j = g$ , all but the rightmost two presufs are periodic with the same core and head.*

*Property 2. Let  $p_i$  be the rightmost instance in its group. If  $x_i$  is periodic then so is  $x_{i+1}$ .*

*Property 3. Suppose  $p_i$  is the rightmost instance in its group  $A_j$  and  $x_i$  is periodic with head  $u$  and core  $v$ ; then  $|x_{i+2}| < |v|$ . Further, suppose  $x_{i+1} = u'(v')^l$ , where  $v'$  is primitive and  $u'$  is a proper suffix of  $v'$ . Then  $|v'| > |u|$ .*

*Property 4. Suppose  $p_i$  is the rightmost instance in its group  $A_j$ , where  $|A_j| > 1$ ; further suppose that  $x_{i-1}$  is periodic with core  $v$ , and  $x_i$  is not periodic. Then  $|x_{i+1}| < |v|$ .*

*Property 5. Both  $p_k$  and  $p_{k+1}$  are in the group  $A_g$ .*

**Proof:** Let  $p_i$  be in group  $A_j$ .

Property 1 is true by definition. To see Property 2, note that as  $x_i$  is periodic,  $x_i = uv^l$ , where  $u$  is a proper suffix of primitive  $v$  and  $l \geq 2$ . But if  $l > 2$ , then the pattern instance corresponding to either presuf  $uv^2$  or presuf  $uv$  would be the rightmost item in  $A_j$ . Thus  $l = 2$ ; but, by definition, the pattern instance corresponding to  $uv$  is not in  $A_j$  only if  $uv$  is periodic. Finally, by Lemma 4.2,  $x_{i+1} = uv$ .

Property 3 can be seen as follows. As in the previous paragraph,  $x_i = uv^2$  and  $x_{i+1} = uv$ . Again  $uv$  is periodic, that is  $uv = u'(v')^l$  for some  $l \geq 2$ , where  $u'$  is a proper suffix of  $v'$  and  $v'$  is primitive. By Lemma 4.2,  $x_{i+2} = u'(v')^{l-1}$ . Suppose  $|v'| \leq |u|$ . As  $v$  is primitive, there must be a substring  $v'$  of  $u'(v')^l = uv$  which straddles the boundary between  $u$  and  $v$ . Thus the substring of  $u'(v')^l$  aligned with the rightmost  $|v'|$ -sized substring of  $u$  is a proper cyclic shift of



$v'$ . But as  $u$  is a suffix of  $v$  this substring is also identical to  $v'$ . By Lemma 2.3,  $v'$  is cyclic, a contradiction. Thus  $|v'| > |u|$  and hence  $|x_{i+2}| < |v|$ .

Property 4 can be seen as follows. As with the previous properties, it follows that  $x_i = uv$ , where  $u$  is a proper suffix of  $v$  and  $v$  is primitive. If  $|x_{i+1}| \geq |v|$ , then  $|x_{i+1}| > |x_i|/2$ . But  $x_{i+1}$  is a presuf of  $x_i$ ; by Lemma 2.1,  $x_i$  would be periodic, a contradiction.

Property 5 can be seen as follows. Since  $x_k$  is the smallest non-null presuf of  $p$ , no non-null prefix of  $x_k$  matches a suffix of  $x_k$ . Therefore, all strings in  $B_g$  have the form  $u_g v_g^l$ ,  $0 \leq l \leq k_g$ , where  $u_g$  is the null string and  $v_g = x_k$ . Since both  $x_k$  and  $x_{k+1}$  have this form, Property 5 is true.  $\square$

**Remark.** The elimination strategy described below and the algorithm in Section 4.2 which uses this elimination strategy to handle presuf shifts, work for most patterns  $p$ . However, there are some patterns for which presuf shifts must be handled differently. The reason for this is made clear in Section 5, which gives a technical portion of the analysis of the algorithm in Section 4.2. These exception patterns are precisely those in which  $x_k$ , the smallest non-null presuf, is a single character and  $g$ , the number of groups, is one. Presuf shifts for these exception patterns are handled separately in Section 4.4.

**Definition.** A clone set is a set  $Q = \{s_1, s_2, \dots\}$  of strings, with  $s_i = uv^{k_i}$ , where  $u$  is a proper suffix of primitive  $v$  and  $k_i \geq 0$ . A set  $U$  of pattern instances is *half-done* if  $|U| \leq 2$  or the set of associated presufs forms a clone set.

The following lemma is the key to our elimination strategy.

**Lemma 4.5** *Consider three presuf pattern instances  $p_a, p_b, p_c$ ,  $a < b < c$  (the order of the indices corresponds to the left-to-right order of the pattern instances). Suppose the set  $\{x_a, x_b, x_c\}$  is not a clone set. Then there exists an index  $d$  in  $p_1$  with the following properties. The characters in  $p_1, p_2, \dots, p_a$  aligned with  $p_1[d]$  are all equal; however, the character aligned with  $p_1[d]$  in at least one of  $p_b$  and  $p_c$  differs from  $p_1[d]$ . Moreover  $m - |x_a| + 1 \leq d \leq m$ , i.e.,  $p_1[d]$  lies in the suffix  $x_a$  of  $p_1$ .*

**Proof:** The substrings of  $p_1, \dots, p_a$  aligned with the suffix  $x_a$  of  $p_1$  are all identical to the string  $x_a$ . Let the substring of  $p_b$  (respectively  $p_c$ ) aligned with the suffix  $x_a$  of  $p_1$  be  $y_b$  (respectively  $y_c$ ). See Fig.2. It suffices to show that at least one of  $y_b$  or  $y_c$  is not identical to  $x_a$ . Suppose for a contradiction that  $y_b = y_c = x_a$ .

Let  $y_b = z_b x_b$  and  $y_c = z_c x_c$ . Note that  $z_b$  is a suffix of  $z_c$ . As  $y_b = y_c$ , a simple induction shows that  $y_b = uv^l$ , where  $u$  is a proper prefix of  $v$  and  $l \geq 1$ , and  $|v|$  is either  $|x_b| - |x_c|$  or some proper divisor of  $|x_b| - |x_c|$  and  $v$  is primitive. As  $x_a = y_b$ , if  $l \geq 2$ ,  $x_a$  is periodic with core  $v$ .

First suppose  $x_a$  is periodic with head  $u$  and core  $v$ . By Lemma 4.2, if  $|x_b| > |uv|$ ,  $x_b = uv^h$  for some  $h$ ,  $1 \leq h < l$ . If  $|x_b| < |uv|$ , as  $|x_b| - |x_c|$  is a multiple of  $|v|$ ,  $|x_b| = |v| + |x_c|$ , so  $x_b = wv$  for some string  $w$ ,  $|w| < |u|$ . But then  $wv$  is a prefix of  $uv$  which implies that  $v$  is cyclic; this is a contradiction. Thus  $x_b = uv^h$ . As  $|x_b| - |x_c|$  is a multiple of  $|v|$ ,  $x_c = uv^j$  for some  $j$ ,  $0 \leq j < h$ , contradicting the fact that  $\{x_a, x_b, x_c\}$  is not a clone set.

Consequently,  $x_a = uv$ . If  $x_a$  is not periodic then  $|x_b| < |x_a|/2$  and  $|v| \leq |x_b| - |x_c| < |x_a|/2$ . But then  $|x_a| < 2|v| < |x_a|$ , a contradiction. While if  $x_a$  is periodic,  $x_a = u'(v')^k$  for some  $k \geq 2$ ; also  $|v'| > |u|$ , by Property 3 of Lemma 4.4; hence  $|x_b| < |v|$ . But  $|x_c| \leq |x_b| - |v| < 0$ , a contradiction.  $\square$

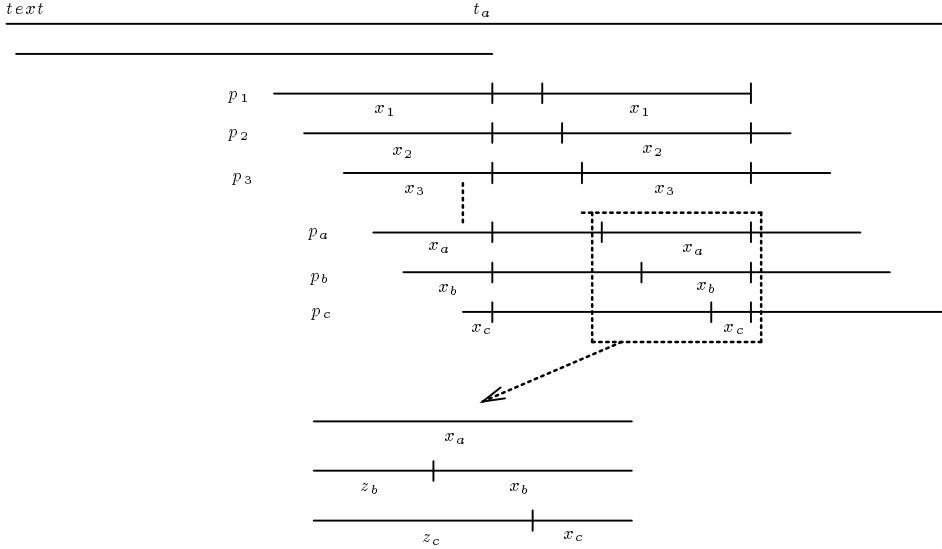


Figure 2: Overlapping pattern instances

Lemma 4.5 implies that a comparison of  $p_1[d]$  with the aligned text character has the following effect: if it is a mismatch, all of  $p_1, \dots, p_a$  are eliminated, while if it is a match, at least one of  $p_b$  and  $p_c$  is eliminated.

Lemma 4.5 enables the elimination of essentially all but one group of pattern instances with at most one mismatch. At each step, for the rightmost  $d$  yielded by Lemma 4.5,  $p_1[d]$  is compared with the aligned text character. If there is a mismatch, the surviving set of pattern instances is half-done, as we show in the following lemma. While if there is no such  $d$ , the surviving set of pattern instances is half-done by Lemma 4.5. This procedure comprises Phase 1 of the elimination procedure.

**Lemma 4.6** *If there is a mismatch in Phase 1 of the elimination procedure, the set  $X$  of surviving pattern instances is half-done.*

**Proof:** Suppose it was not, i.e., for some subset  $\{p_a, p_b, p_c\}$  of  $X$ ,  $\{x_a, x_b, x_c\}$  is not a clone set. The characters in the  $x_i$  suffix of  $p_1$ , for  $i = a, b, c$ , match the aligned substring of  $p_i$ . Hence the mismatch at  $p_1[d]$ , which created set  $X$ , lies to the left of the suffix  $x_i$  of  $p_1$ , for  $i = a, b, c$ . But, by Lemma 4.5, at least one of  $p_a, p_b, p_c$  could have been eliminated by a comparison made within the suffix of  $p_1$  of size  $\max\{|x_a|, |x_b|, |x_c|\}$ . This contradicts the choice of  $d$  as the rightmost index at which a comparison eliminates some pattern instance.  $\square$

The elimination among the remaining half-done set of pattern instances also requires at most one mismatch.

**Lemma 4.7** *Let  $O = \{p_{i_1}, p_{i_2}, \dots, p_{i_l}\}$ ,  $l \geq 2$ , be an uneliminated half-done set and let  $p_{i_1} \in A_r$ . Then  $x_{i_j} = uv^{h-j}$ , where  $u$  is a proper suffix of primitive  $v$  and  $h \geq l$ . Further, there exists an index  $d$  such that the characters in  $\{p_{i_1}, p_{i_2}, \dots, p_{i_{l-1}}\}$  aligned with  $p_{i_l}[d]$  are all equal, but differ from the character  $p_{i_l}[d]$ .  $p_{i_l}[d]$  is aligned with or to the left of  $p_{i_1}[m]$ . In addition, if  $p_{i_1} \notin A_1$  then  $p_{i_l}[d]$  is to the right of  $p_1[m]$  and within distance  $|x_o| - |x_{i_1}|$  of  $p_1[m]$ , where  $p_o$  is*

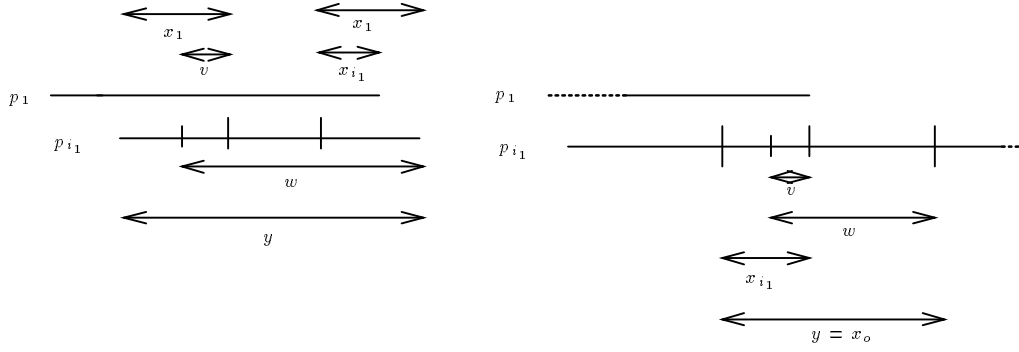


Figure 3: (a)  $p_{i_1} \in A_1$ . (b)  $p_{i_1} \notin A_1$ .

the rightmost pattern instance in  $A_{r-1}$ . If  $p_{i_1} \in A_1$  then  $p_{i_1}[d]$  is to the right of  $t_A$  and aligned with or to the left of  $p_1[m]$ .

**Proof:** Each  $x_{i_j}$ ,  $1 \leq j \leq l$ , is of the form  $uv^{h_j}$ , for some  $h_j \geq 0$ , where  $u$  is a proper suffix of primitive string  $v$ .

See Fig.3. Let  $y$  denote the string  $p_{i_1}$  if  $p_{i_1} \in A_1$  and the string  $x_o$  otherwise. Clearly,  $y$  cannot be periodic with core  $v$ . If  $p_{i_1} \in A_1$  then let  $w$  denote the suffix of  $p_{i_1}$  of length  $m - |x_1| + |v|$ . If  $p_{i_1} \notin A_1$  then let  $w$  denote the substring of  $p_{i_1}$  which has length  $|x_o| - |x_{i_1}| + |v|$  and which overlaps  $p_1$  in exactly  $|v|$  characters. Note that  $w \neq uv^{h'}$ , where  $h' > 0$ ; for otherwise, by Lemma 2.3 and the fact that  $v$  is primitive, the suffix of  $y$  of length  $|w| - |v|$  is cyclic in  $v$  and therefore,  $y$  is periodic with core  $v$ , contrary to assumption.

Let  $w'$  be the smallest suffix of  $w$  which is not of the form  $u'v^{h'}$ , with  $u'$  a suffix of  $v$  and  $h' > 0$ . Define  $d$  to be the index in  $p_{i_1}$  corresponding to  $w'$ . Clearly,  $|w'| > |x_{i_1}| \geq (l-1)|v|$ . Therefore,  $p_{i_1}[d + (l-1)|v|]$  is a character in  $w$ . In addition, if  $p_{i_1} \in A_1$ , then  $|w'| > |x_1|$  and therefore  $p_{i_1}[d + (l-1)|v|]$  is aligned with or to the left of  $p_1[m]$ . The lemma follows if  $p_{i_1}[d]$  is aligned with  $p_{i_1}[d + (l-1)|v|]$  and the characters in  $p_{i_1}, \dots, p_{i_{l-1}}$  which are aligned with  $p_{i_1}[d]$  are all identical and different from  $p_{i_1}[d]$ . We show that these two claims are indeed true.

First, we show that  $|x_{i_j}| - |x_{i_{j+1}}| = |v|$ , for  $1 \leq j < l$ . So suppose for a contradiction that there is a pattern instance  $p_b \notin O$  with  $x_b = uv^{h''}$ ,  $h'' > 0$ , and there are pattern instances  $p_a, p_c \in O$ ,  $p_a$  to the left of  $p_b$  and  $p_c$  to the right of  $p_b$ . See Fig.4. Let  $p_a[e]$  be the rightmost character in  $p_a$  such that the substring of  $p_a$  which starts at  $p_a[e]$  and overlaps  $p_1$  is longer than  $|x_a|$  and not periodic with core  $v$ . Consider the character  $p_1[e']$  aligned with  $p_c[e]$ . The portions of  $p_a, p_b, p_c$  which overlap the suffix of  $p_1$  to the right of  $e'$  are all identical. If Phase 1 stayed to the right of  $e'$  then  $p_a, p_b, p_c$  would all have been eliminated by the mismatch at the end of Phase 1. So  $p_1[e']$  must have been compared in Phase 1. A mismatch at  $e'$  eliminates  $p_a$  and  $p_b$  while a match eliminates  $p_c$ . Either way, a contradiction results.

Finally, note that the character in  $p_{i_j}$ ,  $1 \leq j \leq l-1$ , which is aligned with  $p_{i_1}[d]$  is precisely the character  $p_{i_j}[d + (l-j)|v|]$ . But  $p_{i_1}[d] \neq p_{i_1}[d + |v|] = p_{i_1}[d + 2|v|] = \dots = p_{i_1}[d + (l-1)|v|]$ .  $\square$

**Corollary 4.8** *To eliminate all but one of the pattern instances in any half-done set (in particular, the Phase 1 survivors set)  $\{p_{i_1}, p_{i_2}, \dots, p_{i_j}\}$  it suffices to compare a sequence of characters with the property that any two consecutive characters in the sequence are distance  $|v|$  apart,*

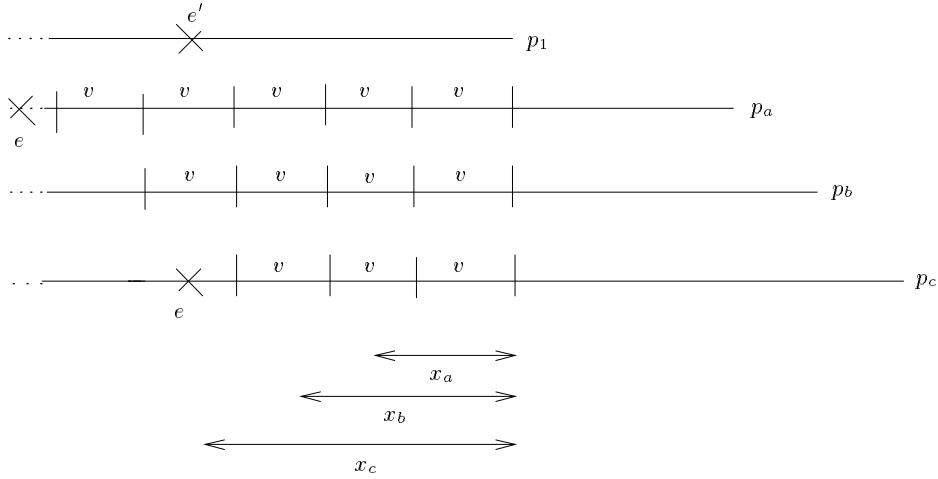


Figure 4: The half-done set is complete.

where  $v$  is the core of  $x_{i_1}$ . Further, the pattern instances in this set are eliminated in right to left order by this comparison sequence (i.e., in decreasing value of  $j$ ).

Let  $p_{i_j}$  be as in Lemma 4.7; the character in  $p_{i_j}$  aligned with  $p_{i_j}[d]$  is compared with the aligned text character. A match eliminates  $p_{i_j}$ ; a mismatch leaves only  $p_{i_j}$  surviving. Iteration of this step ends with one pattern instance surviving after at most one mismatch. This comprises Phase 2 of the elimination procedure.

The sequence of comparisons made in Phase 2 is clearly a right to left sequence. If  $p_1$  is eliminated in Phase 1 then all comparisons in Phase 2 are made to the right of the characters compared in Phase 1. Otherwise, if  $p_1$  is not eliminated in Phase 1, all comparisons in Phase 2 are made to the left of the characters compared in Phase 1.

We recapitulate the elimination strategy now. In Phase 1, characters in  $p_1$  at indices given by a precomputed sequence  $S_1$  are compared in sequence until a mismatch occurs or until the sequence is exhausted. Associated with a mismatch at the  $i$ th comparison given by  $S_1$  is an auxiliary sequence  $S_{2_i}$  of indices. If a mismatch occurs at the  $i$ th comparison in  $S_1$ , Phase 2 begins and comparisons are now made according to the auxiliary sequence  $S_{2_i}$ . A mismatch at any index in the relevant auxiliary sequence completes the elimination process as does the exhaustion of that auxiliary sequence. In either case, only one pattern instance from the set  $\{p_1, \dots, p_k, p_{k+1}\}$  survives.

Let  $|S_1| = j$ . The sequences  $S_1$  and  $S_{2_1}, S_{2_2}, \dots, S_{2_j}$  collectively form a tree  $ET$  (the *elimination tree*).  $ET$  is a binary tree. Each internal node  $x$  of  $ET$  stores an index indicating the comparison to be made. Each internal node has two children. The computation continues at the left child if the comparison at  $x$  is successful, and at the right child otherwise. The computation starts at the root of  $ET$ . Each external node stores the one pattern instance to survive the two phases of comparisons leading to that external node. The external nodes are also called terminal nodes. Note that no pattern instance  $p_i$  can be the survivor at two distinct terminal nodes of  $ET$ . This is because one of the two outcomes of the comparison at the least common ancestor of these two nodes in  $ET$  is bound to eliminate  $p_i$ . It follows that the size of

$ET$  is  $O(k)$ .

The total number of mismatches occurring in the elimination process is at most two because each phase terminates when a mismatch occurs.

**Lemma 4.9** *All but at most one of  $p_1, \dots, p_k, p_{k+1}$  can be eliminated by making up to  $k$  comparisons using the  $O(k)$  sized binary comparison tree  $ET$ . At most two of these comparisons result in mismatches. The sequence of comparisons made by the elimination strategy consists of two left to right sequences. The second sequence is either entirely to the right or entirely to the left of the first one.*

## 4.2 Strategy for Handling Presuf Shifts

Subsequent to the elimination due to tree  $ET$  the presuf handler proceeds in a manner reminiscent of the basic algorithm. That is, there is a current pattern instance,  $p_a$ , which is being matched, and which is the leftmost surviving pattern instance. The next leftmost surviving pattern instance,  $p_b$ , which has a difference point with  $p_a$  is a candidate for elimination. Indeed, a comparison of  $p_a$  with the text is made at the difference point.

The analysis of the presuf handler has the following flavor. Comparisons, with a few exceptions, are charged to distinct text characters. To be precise, for each suffix shift, at most 2 comparisons are charged to the shift rather than to text characters. Even more precisely, if 2 comparisons are charged to the shift, the next presuf shift is at distance at least  $\frac{3(m+1)}{4}$  to the right, and otherwise it is at distance at least  $\frac{m+1}{2}$  to the right. The complexity bound of the algorithm now follows readily.

**Theorem 4.10** *The algorithm performs at most  $n + \frac{8}{3(m+1)}(n - m)$  character comparisons.*

There are three ways in which text characters are charged:

- (i) The character compared is charged.
- (ii) The text character aligned with the left end of the pattern instance eliminated by the comparison is charged.
- (iii) The text character to the immediate right of  $t_A|$  (i.e., aligned with  $|p_{k+1}$ ) is charged.

The three charging methods do not interact readily. To ensure that no text character is charged twice, the switching from one charging method to the other will occur only at carefully selected points in the algorithm. In addition, mismatches are not charged according to rule (i), for the text characters in question may be compared again.

Basically, charging method (i) is used if a pattern instance is successfully matched (at least up to a suffix which is a presuf). Charging method (iii) is used only for the comparison that eliminates the presuf pattern instance which survives the elimination procedure  $ET$ . Charging method (ii) is used otherwise. A partial exception arises for the characters compared by procedure  $ET$ ; this is discussed further below.

Following the use of procedure  $ET$ , the aim is to perform comparisons essentially as in the basic algorithm, that is to compare the character at the difference point of the two leftmost surviving pattern instances which are not prefix overlaps of each other. The analysis ceases to be as straightforward because of the additional  $j \leq k$  comparisons performed by procedure  $ET$ ; indeed, to cope with this, a modified form of the basic algorithm is needed.

There are two objectives.

1. To avoid repeating comparisons at the text characters successfully compared by procedure *ET*.
2. To perform essentially  $j$  fewer comparisons than in the basic algorithm.

Objective 1 is achieved by keeping a record of the successful comparisons in a bit vector of length roughly  $m$ .

The major difficulty, however, is caused by the method used for charging the  $j$  comparisons made by procedure *ET*. It is natural to charge these comparisons to the text characters compared. Unfortunately, this may conflict with the charging using method (ii). To avoid this difficulty, a single additional comparison, with text character  $t_b$ , is performed before using procedure *ET*. The following lemma can then be shown.

**Lemma 4.11** *For each text character  $t_c$  compared by procedure *ET*, with at most  $\alpha \leq 2$  exceptions, there is a distinct previously uncharged text character  $t_{c'}$ ,  $t_{c'}$  aligned with or to the left of  $t_c$  and to the right of  $|p_{k+1}$ , such that the pattern instance  $q_c$  whose left end is aligned with  $t_{c'}$  mismatches either the text character  $t_b$  or some text character matched in procedure *ET*.*

*Let  $\beta$  be the number of mismatches performed by Procedure *ET*. Then, in addition,  $\alpha + \beta \leq 2$ .*

The lemma is proven by specifying a transfer function  $f$ , which associates  $c$  to  $c'$ . The form of  $f$  depends on the sequence of comparisons performed by procedure *ET*. The proof of the lemma is quite non-trivial; it is deferred to Section 5.

Lemma 4.11 is used as follows. Let  $q_e$  be the next pattern instance to match the text (or, at least, to have a suffix, which is also a presuf, matching the text). All pattern instances to the left of  $q_e$ , eliminated by comparisons made after the use of procedure *ET*, are charged using charging method (ii). Using the transfer function, those comparisons to the left of  $|q_e$  made by procedure *ET* are charged to text characters which are not otherwise charged. By contrast, text characters aligned with  $q_e$  are charged using charging method (i). There will be no more than two comparisons performed by the presuf handler that are not thereby charged to a text character; these comparisons are charged to the presuf handler itself.

The algorithm requires a total of five subphases, whose details depend on exactly how  $q_e$  arises.

It is helpful to distinguish three scenarios that may ensue. To this end, let  $p_e$  denote the presuf pattern instance to survive the elimination using tree *ET*. In addition, let  $t_a$  denote the text character  $t_A$ .

The three scenarios follow:

1. All pattern instances overlapping  $p_e$  are eliminated apart from its presuf overlaps, and  $p_e$  or at least a suffix of  $p_e$  is matched.
2.  $p_e$  is eliminated. In addition, there is some pattern instance  $q_c$  overlapping  $p_e$ , such that all pattern instances overlapping  $q_c$  are eliminated apart from its presuf overlaps; further,  $q_c$  or at least a suffix of  $q_c$  is matched.
3.  $p_e$  is eliminated as are all pattern instances overlapping  $p_e$ . Let  $q_d$  denote the leftmost surviving pattern instance in this case.

The first scenario causes no problems from the perspective of the analysis. It suffices to ensure that none of the successful comparisons made by *ET* are repeated. The third scenario is handled by using charging scheme (iii) for the comparison which eliminates  $p_e$  and charging scheme (ii) for the remaining comparisons in the post *ET* phase. Lemma 4.11 ensures that for each comparison made by *ET* (with at most two exceptions) with a text character strictly between  $t_a$  and  $|q_d$  there is a distinct pattern instance whose left end lies strictly between  $|p_{k+1}$  and  $|q_d$ , and which is eliminated by the comparisons made by *ET* plus the one other comparison at text character  $t_b$ . Again, this leads to the desired complexity bound without difficulty.

The second scenario provides the greatest difficulty. In order to avoid unnecessary comparisons, the locations of successful comparisons are recorded. Then, if a difference point occurs at one of these matched text characters the present pattern instance  $p_b$  (see the first paragraph of the subsection) can be removed without further comparisons. However, following a mismatch it is not clear how to maintain this property. For, with only linear storage, it is not clear how to ensure that the current pattern instance, following the mismatch, agrees with the text on a previously matched character, at least if the total work bound is to be linear. (There is no problem if exponential in  $m$  space is available for precomputed structures.) To avoid this difficulty, only the successful comparisons since the last mismatch are recorded.

In fact, this is not quite good enough. It appears necessary to keep track of the characters compared by procedure *ET* regardless of how many characters are compared. This avoids subsequent comparison of these characters. Indeed, any pattern instances mismatching on one or more of these characters are eliminated immediately after the computation with procedure *ET*. This is done with the help of precomputed information.

With this motivation, we proceed with a precise description of the presuf handler procedure. It proceeds in five steps.

**Step 1.** (Before the use of tree *ET*). The characters in  $p_1, \dots, p_k$  aligned with  $p_1[m]$ , the rightmost character in  $p_1$ , are identical. If the character in  $p_{k+1}$  aligned with  $p_1[m]$  is also identical to it, then  $p_1[m]$  is compared with the aligned text character. A mismatch eliminates all of  $p_1, \dots, p_k, p_{k+1}$  and the basic algorithm is restarted with  $|p_a$  placed immediately to the right of  $|p_{k+1}$ . A match is not immediately beneficial as it does not eliminate any of  $p_1, \dots, p_k, p_{k+1}$ . However, it ensures the elimination of sufficiently many appropriate pattern instances for Scenarios 2 and 3 described above.

**Step 2.** The elimination strategy using tree *ET* is applied to the pattern instances  $p_1, \dots, p_k, p_{k+1}$ .

Following Step 2, at most one presuf pattern instance survives. Call it  $p_e$ . Let  $Q$  denote the set of pattern instances which overlap  $p_e$  and have their left end to the right of  $|p_{k+1}$ . In the elimination process, some elements of  $Q$  may also have been eliminated from being potential matches. They need not be reconsidered. Indeed, as the characters successfully matched in Step 2 must not be compared anew, it appears that these pattern instances must not be considered anew. To this end, a subset  $Q_x$  of  $Q$  is associated with each terminal node  $x$  in *ET*.

Let  $T_x$  denote the indices of the text characters successfully compared in Steps 1 and 2.  $Q_x$  contains those pattern instances in  $Q$  which match at all the text indices in  $T_x$ , except possibly the last. This seemingly odd exception is necessary in order to store  $Q_x$  efficiently. Actually,  $Q_x$  satisfies further constraints, but they are not needed for this section. The complete definition of  $Q_x$  and the method for computing it are described in Section 4.5. Here, it suffices to work with the following property: all but at most two of the comparisons in Steps 1 and 2 are successful and are remembered by pattern instances in  $Q_x$ .

Suppose that the elimination process terminates at terminal node  $x$ . Let  $Q' = \{p_\epsilon\} \cup Q_x$ . The elimination procedure of Step 3 is applied to the pattern instances in  $Q'$ .

**Step 3.** This step eliminates among the elements of  $Q'$ .  $q_c$  will denote the leftmost pattern instance to survive Step 3. If  $q_c = p_\epsilon$  then every surviving pattern instance overlapping  $q_c$  will be a presuf overlap of  $q_c$ .

The strategy used here is similar to the one for the basic algorithm. One of two overlapping patterns instances is eliminated by comparing at the difference point of the two instances.

To prevent repeated comparisons of text characters to the right of  $t_a$ , two additional data structures are used. The first is a bit vector  $BV[1 \dots 2m]$ .  $BV[i] = 1$  if the  $i$ th text character to the right of  $t_a$  has been successfully compared in Steps 1 and 2 or in Step 3 since the last mismatch. The second is a list  $LBV$ ; it stores the indices of the bits in  $BV$  set to one in Step 3 since the last mismatch. Initially,  $LBV$  is empty.

The elimination procedure for Step 3 follows. Let  $q_a$  and  $q_b$  denote the two leftmost uneliminated pattern instances in  $Q'$ . Suppose that  $q_b$  lie  $i$  units to the right of  $q_a$ . The reader is advised to refer to Section 2 to review the definition of  $diff_{i+1}$ . If  $diff_{i+1}$  is undefined then  $q_b$  is removed from  $Q'$ .

If  $diff_{i+1}$  is defined then the bit in  $BV$  corresponding to the text character aligned with  $q_a[diff_{i+1}]$  is read. If this bit is 1 then  $q_b$  is removed from  $Q'$  ( $q_b$  can be eliminated as it does not match an already compared text character). Otherwise,  $q_a[diff_{i+1}]$  is compared with the aligned text character. If the two characters are equal, the corresponding bit in  $BV$  is set, the bit's index is added to  $LBV$ , and  $q_b$  is eliminated. If they are not equal, then the bits in  $BV$  at all indices currently in  $LBV$  are reset to 0,  $LBV$  is reset to empty, and  $q_a$  is eliminated.

The elimination procedure is iterated until only one pattern instance remains in  $Q'$ . Let  $q_c$  denote this remaining pattern instance.

**Step 4.** In this step, either all pattern instances overlapping  $q_c$ , apart from presuf overlaps, are eliminated, or  $q_c$  is eliminated.

Let  $Q''$  be the set of pattern instances whose left end lies to the right of  $p_\epsilon$  but not to the right of  $q_c$ . The following step is repeated until either  $q_c$  is eliminated or  $Q'' = \phi$ . Let  $q_d$  be the leftmost pattern instance in  $Q''$ . Suppose  $q_d$  lies  $i$  units to the right of  $q_c$ . If  $diff_{i+1}$  does not exist then  $q_d$  is removed from  $Q''$ . Otherwise, the following bit in  $BV$  is read: the bit corresponding to the text character aligned with  $q_c[diff_{i+1}]$ . If this bit is 1,  $q_d$  is eliminated. If it is 0,  $q_c[i]$  and the aligned text character are compared. If they match then the corresponding bit in  $BV$  is set, its index is added to  $LBV$ , and  $q_d$  is eliminated. Otherwise,  $q_c$  is eliminated and Step 4 comes to an end.

If  $q_c$  is eliminated then  $LBV$  is reset to be empty,  $BV$  is reset to 0, and the basic algorithm is restarted with  $p_a = q_d$ . Otherwise, Step 5 is performed.

**Step 5.** This step seeks to complete the match of  $q_c$ . If at least a presuf of  $q_c$  is matched, the complete match, or the partial match results in a new presuf shift. Otherwise, the basic algorithm is resumed with  $p_a$  immediately to the right of  $q_c$ .

Step 5 compares the characters in  $q_c$  to the right of  $t_a$  apart from those matched in Steps 1 and 2, and those matched in Steps 3 and 4 following the most recent mismatch (incidentally, there was no mismatch in Step 4, as  $q_c$  survived Step 4 if Step 5 is performed). These characters are identified with the help of bit vector  $BV$ . They are matched in right to left order until either a mismatch occurs or they are all matched.

If they all match  $q_c$  is declared a complete match if either  $t_{last} = \phi$  or  $t_{last}$  lies to the left of



$|q_c$ . Recall that  $t_{last}$  is the index of the text character mismatched, if any, immediately prior to the most recent presuf shift (if there was no mismatch,  $t_{last} = \phi$ ).

Next,  $BV$  is reset to zero,  $LBV$  is reset to be empty, and  $t_{last}$  is updated as follows. If the above right to left pass results in a mismatch then  $t_{last}$  is set to the index of the text character at which the mismatch occurs. Otherwise,  $t_{last}$  retains its value unless  $|q_c$  is to its right. In the latter case,  $t_{last} := \phi$ .

The present situation is identical to that preceding a presuf shift in the basic algorithm. This resulting shift is treated in the same way; it too is called a presuf shift.

### 4.3 The Analysis

The comparison complexity of the algorithm of Section 4.2 is given by the following lemma.

**Lemma 4.12** *If  $p$  is not a special case pattern and  $|x'_1| < \frac{m}{2}$  for each presuf shift, then the comparison complexity of the algorithm is bounded by  $n(1 + \frac{8}{3(m+1)})$ .*

**Proof:** We give a charging scheme to account for the comparisons made by the algorithm. This scheme charges almost every comparison to a distinct text character. The only exceptions are a few of the comparisons made by the presuf shift handler. For each presuf shift, depending on the distance between this presuf shift and the next one, the charging scheme fails to charge for up to two of the comparisons made by the presuf shift handler. We refer to the number of comparisons which the charging scheme fails to charge to distinct text characters as the *overhead* of the presuf shift. If a presuf shift has an overhead of two, we show that the next presuf shift must occur at least distance  $\frac{3(m+1)}{4}$  to the right of the current presuf shift. The comparison complexity of our algorithm now follows from the fact that any two consecutive presuf shifts must occur at least distance  $\frac{m+1}{2}$  apart.

**Charging Scheme.** The charging scheme charges in phases. The phases begin and end at shifts and at reversions to the basic algorithm. There are four types of phases; for each phase type a different charging scheme is used.

1. A phase beginning and ending with a basic shift.
2. A phase beginning in the basic algorithm and ending with a presuf shift.
3. A phase beginning with a presuf shift and ending with a reversion to the basic algorithm.
4. A phase beginning and ending with a presuf shift.

Consider any phase and let  $q_1$  and  $q_2$  refer to the leftmost surviving pattern instances at the beginning and end of the phase, respectively. Note that for Type 3 and Type 4 phases,  $q_1$  is a presuf overlap of the pattern instance  $q'$ , the leftmost uneliminated pattern instance prior to the presuf shift which initiated this phase. Specifically, the prefix  $x'_1$  of  $q_1$  is aligned with the suffix  $x'_1$  of  $q'$  (Recall from the start of Section 4 that on a presuf shift, we assume that the suffix  $x'_1$  of  $q_1$  matches the text).

The charging scheme obeys the following properties.

1. At the start of a Type 1 or Type 2 phase, only text characters to the left of  $q_1$  have been charged.

2. At the start of a Type 3 or Type 4 phase, only text characters aligned with or to the left of the prefix  $x'_1$  of  $q_1$  have been charged.

**Type 1 Phase.** Suppose  $i$  comparisons were made in this phase. These  $i$  comparisons are charged to text characters which are aligned with  $q_1$  but to the left of  $|q_2$ . By Lemma 3.1,  $|q_2$  lies at least  $i$  characters to the right of  $|q_1$ . Thus, each text character aligned with  $q_1$  and to the left of  $|q_2$  is charged at most once in this process. Clearly, Property 1 holds at the start of the next phase.

**Type 2 Phase.** In each comparison, a distinct character in  $q_1$  is compared with the aligned text character. Each of these comparisons is charged to the text character compared. Thus, each text character aligned with  $q_1$  is charged at most once in this process. Clearly, Property 2 holds at the start of the next phase.

The charging scheme for Type 3 and Type 4 phases is more involved. Before describing the scheme, we mention the ranges of the text characters charged in each case.

**Type 3 Phase.** The text characters charged lie to the right of the right end of the prefix  $x'_1$  of  $q_1$  and to the left of  $|q_2$ . Each text character in this range is charged at most once. Clearly, Property 1 holds at the start of the next phase.

**Type 4 Phase.** The text characters charged lie to the right of the right end of the prefix  $x'_1$  of  $q_1$  and are aligned with or to the left of the rightmost character in the prefix  $x'_1$  of  $q_2$ . Each text character in this range is charged at most once. Clearly, Property 2 holds at the start of the next phase.

Clearly, the ranges of the text characters charged for different phases are disjoint. Next, we specify the charging scheme for Type 3 and Type 4 phases and justify the claims regarding the overhead.

Consider a presuf shift which initiates a new Type 3 or Type 4 phase. Let  $q'$  be the leftmost uneliminated pattern instance immediately before the presuf shift. Recall that  $t_a$  is the text character aligned with  $q'$ . Consider the comparisons made by the current use of the presuf shift handler. If a mismatch occurs in Step 1, the current phase ends immediately and the basic algorithm is resumed. The presuf shift in this case has overhead 1 and the next presuf shift occurs at least distance  $m + 1$  to the right. Next, suppose that the comparison in Step 1 is successful. Let  $p_e$  be the presuf pattern instance to survive the elimination using tree  $ET$  in Step 2. After the presuf shift handler finishes, one of the three scenarios mentioned Section 4.2 ensues. We consider each in turn.

1. All pattern instances overlapping  $p_e$  are eliminated apart from its presuf overlaps, and  $p_e$  or at least a suffix of  $p_e$  is matched. This is a Type 4 phase.

All comparisons made by the presuf shift handler, except the unsuccessful comparisons in Step 2, are charged to the text characters compared. The bit vector  $BV$  ensures that each of these comparisons involves a different text character. Thus each text character which lies to the right of  $t_a$  and is aligned with or to the left of  $p_e$  is charged at most once. At most two comparisons in Step 2 are unsuccessful, so this shift has overhead at most two.

Consider the situation when there are exactly two mismatches in Step 2.  $p_1$  is clearly eliminated in this case. In addition, we show in the next paragraph that if  $x_1$  is periodic, with core  $v$  and head  $u$  say, then all pattern instances whose associated presufs have the form  $uv^o$ ,  $o \geq 1$ , are also eliminated. Let  $x_e$  be the presuf associated with  $p_e$ . It follows that  $x_1 = x_e w x_e$

for some non-empty string  $w$ . Since  $p = x_1 z x_1$ , for some non-empty string  $z$ ,  $|x_e| \leq \frac{m-3}{4}$ . This guarantees that the next presuf shift occurs at least distance  $\frac{3(m+1)}{4}$  to the right. If there is just one mismatch in Step 2 then, as  $|x'_1| < \frac{m}{2}$ , the next presuf shift occurs at least distance  $\frac{m+1}{2}$  to the right.

To see that two mismatches in Step 2 eliminate all presuf pattern instances with associated presufs of the form  $uv^o$ ,  $o \geq 1$ , it suffices to show that at most one such pattern instance survives the first mismatch; the second mismatch will surely eliminate this pattern instance. Suppose two pattern instances  $p_{i_1}, p_{i_2}$ ,  $i_1 < i_2$ ,  $i_1, i_2 \neq 1$ ,  $x_{i_1} = uv^{o_1}$ ,  $x_{i_2} = uv^{o_2}$ ,  $o_1, o_2 \geq 1$ , survive the first mismatch, which occurs at text character  $t_x$ , say. The portions of  $p_1$  and  $p_{i_1}$  to the right of  $t_x$  match each other while the characters in  $p_1$  and  $p_{i_1}$  aligned with  $t_x$  are different. This implies that  $p_1$  and  $p_{i_2}$  have a difference point strictly between  $t_x$  and  $t_b$ ; more precisely, the character in  $p_1$  which is distance  $(o_2 - o_1)|v|$  to the right of  $t_x$  is a difference point. Therefore, either  $p_1$  or  $p_{i_2}$  would have been eliminated before the first mismatch, which is a contradiction.

2.  $p_e$  is eliminated. In addition, there is some pattern instance  $q_c$  overlapping  $p_e$ , such that all pattern instances overlapping  $q_c$  are eliminated apart from its presuf overlaps; further,  $q_c$  or at least a suffix of  $q_c$  is matched. This is also a Type 4 phase.

Each comparison in Steps 1 and 2 with a text character to the left of  $|q_c$  for which function  $f$  is defined is charged to the text character specified by the function  $f$ , called its  $f$  value;  $f$  values are distinct by definition. Comparisons in Step 3 fall into one of three categories (see Lemma 4.11 and the following paragraph):

1. Comparisons which eliminate patterns instances whose left ends lie to the right of  $|p_{k+1}$  and to the left of  $|q_c$ .
2. Comparisons which eliminate pattern instances whose left ends are aligned with or to the right of  $|q_c$ .
3. The comparison which eliminates  $p_e$ .

Each comparison in the first category is charged to the text character aligned with the left end of the pattern instance eliminated. By the definition of the function  $f$ , these text characters do not occur in the range of  $f$  values. Comparisons in the second category, along with the comparisons made in Steps 4 and 5 and those successful comparisons in Steps 1 and 2 that involve text characters overlapping  $q_c$ , are charged to the text characters compared.  $BV$  ensures that each of these comparisons involves a distinct text character. Thus each text character which lies to the right of  $|p_{k+1}$  and is aligned with or to the left of  $|q_c|$  is charged at most once. The comparison that eliminates  $p_e$  is charged to the text character aligned with  $|p_{k+1}$ . Since all  $f$  values lie to the right of  $|p_{k+1}$  and all pattern instances eliminated by comparisons in the first category are with left ends to the right of  $|p_{k+1}$ , this text character is charged exactly once. The two comparisons in Step 2 lacking  $f$  values constitute the overhead of this presuf shift. Since  $p_e$  is eliminated, the next presuf shift occurs at least distance  $m + 1$  to the right of the current presuf shift.

3.  $p_e$  is eliminated as are all pattern instances overlapping  $p_e$ . This is a Type 3 phase.

Let  $q_d$  denote the leftmost surviving pattern instance. All comparisons in Steps 1 and 2 for which function  $f$  is defined are charged to their  $f$  values.  $f$  values are distinct by definition. Excluding the comparison which eliminates  $p_e$ , each comparison in Steps 3 and 4 eliminates some pattern instance whose left end lies to the right of  $|p_{k+1}$  and to the left of  $|q_d$ . Each such

comparison is charged to the text character aligned with the left end of the pattern instance eliminated. These text characters cannot occur in the range of the function  $f$  and hence are charged only once. Thus each text character which lies to the right of  $|p_{k+1}$  and to the left of  $|q_d$  is charged at most once. The comparison that eliminates  $p_e$  is charged to the text character aligned with  $|p_{k+1}$ . The two comparisons in Step 2 lacking  $f$  values constitute the overhead of this presuf shift. Since  $p_e$  is eliminated, the next presuf shift occurs at least distance  $m + 1$  to the right of the current presuf shift.  $\square$

The following lemma is shown in Section 4.5.

**Lemma 4.13** *The total space used by the algorithm for the case when  $|x'_1| < \frac{m}{2}$  for all presuf shifts is  $O(m)$ . Further, for any terminal node  $x$  of  $ET$ ,  $Q_x$  can be obtained in  $O(m)$  time. The preprocessing required by the algorithm can be accomplished in  $O(m^2)$  time.*

**Lemma 4.14** *Suppose that  $p$  is not a special case pattern and  $|x'_1| < \frac{m}{2}$  for all presuf shifts. Then the total time taken by the algorithm is  $O(n + m)$ , following preprocessing of the pattern which takes  $O(m^2)$  time.*

**Proof:** By Lemma 4.12, the number of character comparisons made is  $O(n)$ . It remains to count the time spent in all other operations. The basic algorithm makes only character comparisons. Next, consider the presuf handler of Section 4.2. Steps 1 and 2 make only character comparisons. Following Step 2, computing  $Q_x$  takes  $O(m)$  time by Lemma 4.13. Steps 3 and 4 take  $O(m)$  time because  $|Q'|, |Q''| = O(m)$  and each of the operations in these steps, except the operations used for resetting  $BV$ , leads to the removal of a pattern instance from one of  $Q''$  or  $Q'$ . Further, the total time spent by Steps 3 and 4 in resetting  $BV$  is bounded by the time taken by these steps to set bits in  $BV$ , which is  $O(m)$ . Clearly, Step 5 takes  $O(m)$  time. Thus, the total time taken by the the presuf handler of Section 4.2 is  $O(m)$ . Since any two presuf shifts occur at least  $m - |x'_1| > m/2$  distance apart, the total time taken by the algorithm is  $O(n + m)$ .  $\square$

#### 4.4 Handling Presuf Shifts for Special Case Patterns

As mentioned in Section 4.1, a different algorithm is needed to handle presuf shifts for patterns for which  $|x_k| = 1$  and  $g = 1$ . We give an algorithm which handles presuf shifts for such patterns when  $|x'_1| < \frac{m}{2}$  (recall that  $x'_1$  for a presuf shift was defined towards the start of Section 4). The case  $|x'_1| \geq \frac{m}{2}$  is handled in Section 6.

The goal of this algorithm is to reach one of the following two situations:

1. The identification of a pattern instance  $q_c$  satisfying the following property: no pattern instance  $q_d$  which precedes  $q_c$  survives and a pattern instance overlapping  $q_c$  survives only if it is a presuf overlap of  $q_c$ .
2. A return to the basic algorithm.

Further, this is achieved with at most two mismatches.

Let  $x_k = b$ . Any character other than  $b$  is called a non- $b$  character. Since we assume that the pattern contains at least two different characters, it contains a non- $b$  character. Let  $p[j]$  be the leftmost non- $b$  character in  $p$  and let  $t_c$  denote the text character aligned with  $|p_{k+1}$ . Let  $t_d$  be the leftmost non- $b$  text character, if any, to the right of, and including,  $t_c$ .

By the definition of special cases patterns, all presufs consist solely of  $b$ 's. Therefore,  $p_1[j]$  lies to the right of  $t_a$ . Note that no complete match can occur with one of  $p[1 \dots j - 1]$  aligned

with  $t_d$ . Thus, if  $t_d$  lies to the left of  $p_1[j]$  then the next potential match instance of  $p$  would have its left end to the right of  $t_d$ . Otherwise, the next potential match instance of  $p$  has  $p[j]$  aligned with  $t_d$ . Also notice that if  $t_d$  does not exist then there are no more complete matches. These observations lead to the following three step procedure.

**Step 1.** This step locates  $t_d$  and then eliminates all but at most one pattern instance  $q_c$  overlapping  $t_d$ . Starting at  $t_c$ , a left to right scan of the text is performed to locate  $t_d$  (i.e., each text character is compared to  $b$ ;  $t_d$  is the character at which the first mismatch occurs). If  $t_d$  does not exist the algorithm halts. If  $t_d$  exists and lies to the left of  $p_1[j]$  then the basic algorithm is restarted with  $|p$  placed to the immediate right of  $t_d$ . Otherwise,  $q_c$  is chosen to be the pattern instance such that  $q_c[j]$  is aligned with  $t_d$ .  $q_c$  is the next potential match instance to be considered.

**Step 2.** In this step, either  $q_c$  is eliminated or all pattern instances overlapping  $q_c$ , except for presuf overlaps of  $q_c$ , are eliminated. This is done using the basic algorithm, slightly modified to account for the matched prefix. Suppose the leftmost difference points are used in the sequence  $S$  in the basic algorithm, as against any arbitrary difference points. Then  $dif_2, \dots, dif_j$  are all equal to  $j$  and  $dif_{j+1}, \dots, dif_m$  are all greater than  $j$ , whenever defined. In Step 2, the characters in  $q_c$  to the right of  $q_c[j]$  which are at the indices given by  $S$  are compared with the aligned text characters in the order in which they appear in  $S$ . This continues until either a mismatch occurs or the sequence is exhausted. A mismatch leads to the basic algorithm with  $|p$  shifted to the right of  $q_c$  by distance at least  $j - 1$  plus the number of comparisons made in this step. If no mismatch occurs then Step 3 follows.

**Step 3.** Characters in  $q_c$  which are not yet matched are compared from right to left with their aligned text characters until a mismatch occurs or  $q_c$  is fully matched. The present situation is now identical to the situation at the beginning of a presuf shift and is handled in the same way.

The comparison complexity of the above algorithm is determined by the following lemma.

**Lemma 4.15** *If  $p$  is a special case pattern and  $|x'_1| < \frac{m}{2}$  for each presuf shift, then the comparison complexity of the algorithm is  $n(1 + \frac{2}{m+1})$ .*

**Proof:** We give a charging scheme to account for the comparisons made by the algorithm for handling special case patterns. The definition of a phase, the charging scheme for Type 1 and Type 2 phases, and the ranges of text characters charged in each phase type remain the same as in Lemma 4.12. Only the charging scheme for Type 3 and Type 4 phases needs to be modified in accordance with the presuf shift handler for special case patterns.

Consider a presuf shift which initiates a new Type 3 or Type 4 phase. We show that it has an overhead of at most one. The comparison complexity of the algorithm now follows from the fact that  $|x'_1| < \frac{m}{2}$  and therefore, any two consecutive presuf shifts must occur at least  $\frac{m+1}{2}$  characters apart.

**Charging Scheme for the Presuf Shift Handler.** Let  $q'$  and  $q_1$  be the leftmost uneliminated pattern instances immediately before and after the presuf shift, respectively. Recall that  $t_a$  is the text character aligned with  $q'$ .

We show that presuf shifts have overhead at most one for these patterns. Let  $q_c$  be the leftmost pattern instance which survives Step 1. All successful comparisons in Step 1 are charged to the text characters compared. These text characters lie to the left of  $q_c[j]$ , where  $j$  is the least index such that  $p[j]$  differs from  $p[m]$ . The lone unsuccessful comparison in Step 1 constitutes the overhead of this shift. Consider two cases now.

1. Suppose  $q_c$  survives Step 2. All comparisons made in Steps 2 and 3 are charged to the text characters compared. Thus, each text character which lies to the right of  $t_a$  and is aligned with or to the left of  $q_c$  is charged at most once. All future comparisons will be charged to text characters to the right of  $q_c$ .
2. Suppose  $q_c$  does not survive Step 2. Each successful comparison in Step 2 eliminates some pattern instance lying entirely to the right of  $q_c[j]$  and is charged to the text character aligned with the left end of that pattern instance. The unsuccessful comparison which eliminates  $q_c$  in Step 2 is charged to the text character aligned with  $q_c[j]$ . Thus, each text character lying strictly between  $t_a$  and  $|q_d$  is charged at most once, where  $q_d$  is the leftmost surviving pattern instance at the end of Step 2. All future comparisons will be charged to text characters aligned with or to the right of  $|q_d$ .

□

**Lemma 4.16** *Suppose that  $p$  is a special case pattern and  $|x'_1| < \frac{m}{2}$  for all presuf shifts. Then the total time taken by the algorithm is  $O(n + m)$ , following preprocessing of the pattern which takes  $O(m^2)$  time. The total space used by the algorithm is  $O(m)$ .*

**Proof:** The lemma, except for the preprocessing time, is obvious from the above description. Since no extra preprocessing is required for special case patterns, the lemma follows from Lemma 4.14. □

**Theorem 4.17** *Suppose for all presuf shifts,  $|x'_1| < \frac{m}{2}$ . Then the total space used by the algorithm is  $O(m)$  and the total time taken by the algorithm, after preprocessing, is  $O(n + m)$ . The preprocessing required by the algorithm takes  $O(m^2)$  time.*

**Proof:** Follows from Lemmas 4.14 and 4.16. □

## 4.5 Data Structure Details

We prove Lemma 4.13 in this section. The following data structures are used by the algorithm.

1. The array  $S$  used in the basic algorithm.
2. An array, indexed by  $i$ , storing  $dif_i$ ,  $2 \leq i \leq m$ , used by the presuf shift handler.
3.  $BV$  and  $LBV$ , the bit vector and its associated list.
4.  $ET$ , the elimination tree.
5.  $Q_x$ , for each terminal node  $x$  of  $ET$ , as defined after Step 2 in Section 4.2.

Of these, the first three have size  $O(m)$  by definition. By Theorem 4.9,  $ET$  has size  $O(m)$  too.

It remains to show how to represent  $Q_x$ , for each terminal node  $x$  of  $ET$ , using  $O(m)$  space overall. The following definitions are helpful. Let  $t_b$  be the text character aligned with  $p_1$ . Let  $Q$  refer to the set of pattern instances which overlap  $p_{k+1}$ , have left ends to the right of  $|p_{k+1}$  and either match or do not overlap  $t_b$ .

Before showing how to maintain  $Q_x$ , it is helpful to recapitulate some structural properties of  $ET$ .  $ET$  is a binary tree with each internal node having two children. At each internal

node  $y$ , a character  $c_y$  in  $p$  is potentially compared with the text character  $tc_y$ . A successful comparison leads to the left child of  $y$  while a mismatch leads to the right child. Comparisons are made starting at the root of  $ET$  and continuing until a terminal node (a leaf) is reached. A node in  $ET$  lies in the right subtree of at most two of its ancestors.

Node  $x$  is said to be a *failing descendant* of node  $y$  if  $x$  is a proper descendant of  $y$  and lies in the right subtree of  $y$ . A terminal node  $x$  can be a failing descendant of at most two nodes in  $ET$ . Let  $p(x)$  denote the parent of  $x$ . For each terminal node  $x$ , let  $Anc(x)$  be defined as follows. If both children of  $p(x)$  are terminal nodes and  $p(x)$  is the right child of  $p(p(x))$  then  $Anc(x)$  is the set of proper ancestors of  $p(x)$ . Otherwise,  $Anc(x)$  is the set of proper ancestors of  $x$ .

$q \in Q$  is said to *occur* at terminal node  $x$  of  $ET$  if  $q \in Q_x$ . In Section 4.2, we tentatively defined  $Q_x$  to be the set of pattern instances in  $Q$  which match at all text characters compared successfully at nodes in  $Anc(x)$  (actually, the definition was not this precise). Now, we refine this definition by letting  $Q_x$  satisfy some additional constraints. Informally,  $q$  should occur at  $x$  if it is consistent with all comparisons made at nodes in  $Anc(x)$ . This motivates the following characterization of  $Q_x$ . Let  $Y \subset Anc(x)$  consist of those nodes with respect to which  $x$  is a failing descendant. Then  $Q_x$  is the maximal subset of  $Q$  such that each  $q \in Q_x$  satisfies the following properties:

1.  $\forall y \in Anc(x) - Y$ , the character in  $q$  aligned with  $tc_y$ , if any, matches the character  $c_y$ .
2.  $\forall y \in Y$ , the character in  $q$  aligned with  $tc_y$ , if any, is different from  $c_y$ .

$ET$  may have  $\theta(m)$  terminal nodes. Even though  $|Q_x| < m$  for each terminal node  $x$ , storing  $Q_x$  explicitly for each terminal node  $x$  could require  $\Omega(m^2)$  space overall. We show how to store the sets  $Q_x$  so that  $O(m)$  space is used in total and any particular  $Q_x$  can be retrieved in  $O(m)$  time.

Let  $l_1, l_2, \dots, l_h$ , in that order, be the nodes along the leftmost path in  $ET$  starting at the root and ending at the terminal node  $l_h$ . Define the right subtree of  $l_i$  to be the subtree rooted at the right child of  $l_i$ . Note that  $tc_{l_1}, \dots, tc_{l_{h-1}}$  form a right to left sequence. We show how to maintain  $Q_x$ , for all terminal nodes  $x$  in the right subtrees of  $l_1, \dots, l_{h-1}$ , in  $O(m)$  space altogether. Only the terminal node  $l_h$  remains and  $Q_{l_h}$  can be stored explicitly in  $O(m)$  space.

We mark some of the nodes  $l_1, \dots, l_{h-1}$ . Node  $l_i$  is marked if its right child is neither a terminal node nor the parent of two terminal nodes. Thus, node  $l_i$  is marked if Phase 2 could make at least two comparisons following a mismatch at  $tc_{l_i}$ . Let  $l'_1, \dots, l'_s$ , in that order, be the nodes marked.

The following lemmas are helpful.

**Lemma 4.18** *Consider terminal nodes  $x_1$  and  $x_2$  of  $ET$  and let their least common ancestor be  $y$ . Suppose at most one of the following is true: first,  $y$  is the parent of both  $x_1$  and  $x_2$  and second,  $y$  is the right child of  $p(y)$ . If  $q$  occurs at  $x_1$  and at  $x_2$  then  $q$  does not overlap  $tc_y$ .*

**Proof:** Clearly,  $y \in Anc(x_1)$  and  $y \in Anc(x_2)$ . Suppose  $q$  overlaps  $tc_y$ . Let  $c$  be the character in  $q$  aligned with  $tc_y$ . Without loss of generality, assume that  $x_1$  is a failing descendant of  $y$ . Then  $x_2$  is not a failing descendant of  $y$ . By the definition of  $Q_{x_1}$ ,  $c \neq c_y$ . By the definition of  $Q_{x_2}$ ,  $c = c_y$ , a contradiction.  $\square$

**Corollary 4.19** *Let  $i \geq 1$  be the smallest number such that  $q \in Q$  does not overlap  $tc_{l_i}$ .  $q$  can occur at terminal nodes in the right subtrees of at most one of  $l_1, \dots, l_{i-1}$ . Further, if  $q$  occurs*

at some terminal node in the subtree rooted at  $l_i$ , it cannot occur at terminal nodes in the right subtrees of any of  $l_1, \dots, l_{i-1}$ .

**Lemma 4.20** *Let  $i \geq 1$  be the smallest number such that  $q \in Q$  does not overlap  $tc_{l_i}$ . Suppose  $q$  occurs at a terminal node in the subtree rooted at  $l_i$ . Then  $q$  occurs at all terminal nodes in the right subtrees of each of those nodes among  $l_i, \dots, l_{h-1}$  which are unmarked. Further,  $q$  occurs at  $l_h$ .*

**Proof:** Clearly, the characters in  $q$  which overlap  $tc_{l_1}, \dots, tc_{l_{i-1}}$  match the characters  $c_{l_1}, \dots, c_{l_{i-1}}$ , respectively. Further,  $q$  does not overlap  $tc_{l_i}, \dots, tc_{l_{h-1}}$ . Therefore,  $q$  occurs at  $l_h$ . In addition, if a terminal node  $x$  is in the right subtree of an unmarked node  $l_j$ ,  $j \geq i$ , then either  $l_j = p(x)$  or  $l_j = p(p(x))$  and  $p(x) \notin \text{Anc}(x)$ . From the definition of  $Q_x$ ,  $q$  must occur at  $x$ .  $\square$

**Lemma 4.21** *Consider marked node  $l'_i$ ,  $1 \leq i \leq s$ , and let  $j$  be the smallest number such that  $tc_{l'_i}$  is to the left of the suffix  $x_j$  of  $p_1$ .  $p_{j-1}$  must be the rightmost pattern instance in its group. In addition,  $p_j$  is the leftmost presuf pattern instance to survive a mismatch at  $tc_{l'_i}$ .*

**Proof:** Since  $l'_i$  is marked, at least three presuf pattern instances must survive a mismatch at  $tc_{l'_i}$ . Let the leftmost three such pattern instances be  $p_a, p_b, p_c$  (listed in left to right order). Let  $A_w$  be the group containing  $p_j$ . Write  $x_j$  as  $uv^e$ , where  $e \geq 1$ ,  $u$  is a proper suffix of primitive  $v$ , and all presufs associated with  $A_w$  have the form  $uv^{e'}$ ,  $e' \geq 1$ .

By Lemma 4.5, successful comparisons within the suffix  $x_j$  of  $p_1$  suffice to eliminate all but at most two of the pattern instances in the groups  $A_{w+1}, \dots, A_g$  (at most two pattern instances in  $A_{w+1}, \dots, A_g$  can form a half-done set with  $p_j$ ). Therefore,  $p_a \in A_w$  and  $x_a = uv^{e_a}$ ,  $e_a \geq 1$ . Since  $p_a, p_b, p_c$  all survive the mismatch at  $tc_{l'_i}$ ,  $\{p_a, p_b, p_c\}$  is a half-done set and therefore  $e_a \geq 2$ . It follows that  $x_b = uv^{e_b}$ ,  $e_b \geq 0$ , and  $x_c = uv^{e_c}$ ,  $e_c \geq 0$ .

Next, suppose  $p_{j-1}$  is not the rightmost pattern instance in its group. Then  $p_{j-1} \in A_w$  and  $x_{j-1}$  has the form  $uv^{e+1}$ ,  $e+1 \geq 2$ . We show that  $p_b$  would have been eliminated by a comparison to the right of  $tc_{l'_i}$ , which is a contradiction. Note that  $p_{j-1}$  and  $p_a$  have a difference point, which is aligned with or to the right of  $tc_{l'_i}$  and aligned with  $p_1$ . Let  $p_{j-1}[d]$  be the rightmost such difference point. Clearly,  $p_{j-1}[d]$  is to the left of the suffix  $x_a$  of  $p_1$ .  $p_{j-1}[d + (e_a - e_b)|v]$  is a difference point of  $p_{j-1}$  and  $p_b$  which is aligned with  $p_1$ . A match at this difference point would have eliminated  $p_b$ .

Finally, suppose  $p_a \neq p_j$ . Then  $p_j$  and  $p_a$  have a difference point, which is aligned with or to the right of  $tc_{l'_i}$  and aligned with  $p_1$ . Let  $p_j[d]$  be the rightmost such difference point. An argument similar to the one in the previous paragraph shows that  $p_j$  and  $p_b$  have a difference point to the right of  $p_j[d]$  and aligned with  $p_1$ ; a match at this difference point would have eliminated  $p_b$ , which is a contradiction.  $\square$

**Corollary 4.22** *Consider marked nodes  $l'_{i_1}$  and  $l'_{i_2}$ ,  $1 \leq i_1 < i_2 \leq s$ . Let  $x_{i_1}$  and  $x_{i_2}$  be the smallest suffixes (which are also presufs) of  $p_1$  which overlap  $tc_{l'_{i_1}}$  and  $tc_{l'_{i_2}}$  respectively. Then  $i_1 \neq i_2$ .*

**Proof:** If  $i_1 = i_2$  then by Lemma 4.21,  $p_j$  is the leftmost presuf pattern instance to survive the mismatches at both  $tc_{l'_{i_1}}$  and  $tc_{l'_{i_2}}$ . But since a  $p_j$  survives a mismatch at  $tc_{l'_{i_1}}$ , it cannot survive a match at  $tc_{l'_{i_1}}$  and therefore, it cannot survive a mismatch at  $tc_{l'_{i_2}}$ .  $\square$



**Lemma 4.23** *The size of the presuf corresponding to the rightmost pattern instance in  $A_j$ ,  $1 \leq j \leq g$ , is at most  $\frac{m}{(\frac{3}{2})^j}$ .*

**Proof:** For  $j = 1$ , the claim is clearly true. Assume that the claim is true for  $A_{j-1}$ , i.e., the size of the presuf corresponding to rightmost pattern instance  $p_e$  in  $A_{j-1}$  is less than  $\frac{m}{(\frac{3}{2})^{j-1}}$ .  $x_e$  has either the form  $uvv$  or the form  $uv$ , where  $u$  a proper suffix of  $v$ . In the former case,  $x_{e+1} = uv$  and in the latter case,  $x_e = x_{e+1}zx_{e+1}$  for some non-empty string  $z$  (as  $uv$  is not periodic). Thus  $|x_{e+1}| < \frac{2|x_e|}{3}$ . The claim follows.  $\square$

**Lemma 4.24** *The number of pattern instances in  $Q$  which overlap  $t_b$  and are entirely to the right of  $tc_{l'_i}$  is less than  $\frac{m}{(\frac{3}{2})^{s-i+1}}$ , for all  $i$ ,  $1 \leq i \leq s$ .*

**Proof:** From Lemma 4.21 and Corollary 4.22, the rightmost presuf pattern instance  $p_j$  such that the suffix  $x_j$  of  $p_1$  overlaps  $tc_{l'_i}$  must be the rightmost pattern instance in some group  $A_{j'}$ ,  $j' \geq s - i + 1$ . The lemma follows from Lemma 4.23.  $\square$

Consider the right subtrees of  $l'_1, \dots, l'_s$ . Note that the comparisons made in each of these subtrees are aimed at eliminating half-done sets whose leftmost pattern instances are in distinct groups. Each of these comparisons is made to the right of  $p_1[m]$ , as described in Lemma 4.7 and Corollary 4.8.

**Lemma 4.25** *The number of pattern instances in  $Q$  which are entirely to the right of  $t_b$  and overlap some text character compared in the right subtree of  $l'_i$  is at most  $\frac{m}{(\frac{3}{2})^{s-i+1}}$ , for all  $i$ ,  $1 \leq i \leq s$ .*

**Proof:** Recall from Lemma 4.7 that a half-done set whose leftmost pattern instance is in group  $A_j$ ,  $j > 1$ , is eliminated in Phase 2 of the elimination strategy by making comparisons at text characters which are at most distance  $|x_{j'-1}| - |x_{j'}|$  to the right of  $t_b$ , where  $p_{j'}$  is the leftmost presuf pattern instance in  $A_j$ . From Lemma 4.23,  $|x_{j'-1}| < \frac{m}{(\frac{3}{2})^{j-1}}$ , the lemma follows.  $\square$

**Lemma 4.26** *Consider a marked node  $l'_i$  and the set of terminal nodes in its right subtree. If a pattern instance  $q$  occurs at two of these terminal nodes, say  $w$  and  $y$ , then  $q$  occurs at all terminal nodes in the subtree rooted at the least common ancestor  $z$  of  $w$  and  $y$ .*

**Proof:** By Lemma 4.18,  $q$  does not overlap the character  $tc_z$ . Since comparisons made in the right subtree of  $l'_i$  constitute a right to left sequence,  $q$  does not overlap  $tc_{z'}$ , where  $z'$  is any descendant of  $z$ . The lemma now follows immediately from the definition of the sets  $Q_x$ .  $\square$

We are now ready to describe the data structure for storing the  $Q_x$ 's. The following subsets of  $Q$  are required:  $Z_1, \dots, Z_{h-1}$ ,  $Y_1, \dots, Y_{h-1}$  and  $W_1, \dots, W_s$ . The  $Z$  and the  $Y$  sets are used for terminal nodes which lie in the right subtrees of the unmarked nodes among  $l_1, \dots, l_{h-1}$ . The  $W$  sets are used for terminal nodes which lie in the right subtrees of marked nodes.

The  $Z$  sets are defined first. For each  $i$ ,  $1 \leq i \leq h - 1$ , where  $l_i$  is unmarked, define  $Z_i$  to be the set of pattern instances in  $Q$  which overlap  $tc(l_i)$  and occur only at terminal nodes in the right subtree of  $l_i$ . Clearly,  $\sum_{i=1}^{h-1} Z_i = O(m)$ .

The  $Y$  sets are defined next. For each  $i$ ,  $1 \leq i \leq h - 1$ , define  $Y_i$  to be the set of pattern instances  $q \in Q$  with the following properties.

1.  $q$  does not overlap  $tc_{l_i}$ .
2. If  $i > 1$ ,  $q$  overlaps  $tc_{l_{i-1}}$ .
3.  $q$  occurs at a terminal node in the subtree rooted at  $l_i$ .

Clearly,  $\sum_{i=1}^{h-1} Y_i = O(m)$ . Further, each pair of  $Y$  sets is disjoint and  $Z_i$  is disjoint from  $Y_1, \dots, Y_i$ . The following lemma explains the significance of the  $Y$  and  $Z$  sets.

**Lemma 4.27** *If terminal node  $x$  is in the right subtree of unmarked node  $l_i$ ,  $Q_x = Y_1 \cup Y_2 \cup \dots \cup Y_i \cup Z_i$ .*

**Proof:** Suppose  $q \in Q_x$ . If  $q$  overlaps  $tc_{l_i}$  then, by Corollary 4.19,  $q \in Z_i$ . If  $q$  does not overlap  $tc_{l_i}$  then clearly  $q$  must be in some  $Y_j$ ,  $j \leq i$ .

Next, suppose  $q \in Y_j$ ,  $j \leq i$ . By Lemma 4.20,  $q \in Q_x$ . Finally, if  $q \in Z_i$  then  $q \in Q_x$  as the only internal node (if any) in the right subtree of  $l_i$  is not in  $\text{Anc}(x)$ .  $\square$

Finally, the  $W$  sets are defined. For each  $i$ ,  $1 \leq i \leq s$ ,  $W_i$  consists of those pattern instances which occur at some terminal node in the right subtree of marked node  $l'_i$ . Let  $W'_i$  denote the set obtained from  $W_i$  by removing those pattern instances which do not overlap any of the text characters compared in the right subtree of  $l'_i$ .

**Lemma 4.28**  $\sum_{i=1}^s |W'_i| = O(m)$ .

**Proof:** Split  $W'_i$  into two disjoint subsets,  $W_i^1$  and  $W_i^2$ .  $W_i^1$  consists of those pattern instances which overlap  $tc_{l'_i}$  and  $W_i^2$  consists of pattern instances which do not overlap  $tc_{l'_i}$ .

By Lemma 4.18, pattern instances in  $W_i^1$  occur only at terminal nodes in the right subtree of  $l'_i$ . Therefore, it suffices to show that  $\sum_{i=1}^s |W_i^2| = O(m)$ . From Lemmas 4.24 and 4.25, it follows that  $\sum_{i=1}^s |W_i^2| = \sum_{i=1}^s (2 \frac{m}{(\frac{3}{2})^{s-i+1}}) = O(m)$ .  $\square$

Consider some  $i$ ,  $1 \leq i \leq s$ . The manner in which  $W_i$  is maintained so as to facilitate the recovery of  $Q_x$  for each terminal node  $x$  in the right subtree of marked node  $l'_i$  remains to be shown. Clearly, pattern instances in  $W_i - W'_i$  occur at all such nodes  $x$  and can be stored implicitly in constant space by just storing the rightmost text position compared in the right subtree of  $l'_i$ . For the terminal nodes  $x$  in  $l'_i$ 's right subtree, we show how to store the pattern instances in  $Q_x \cap W'_i$  using a total of  $O(|W'_i|)$  space (summing over all  $x$ ). The linear space bound then follows from Lemma 4.28.

At each internal node  $y$  in the right subtree  $T$  of  $l'_i$ , a set  $Com_y$  is stored. At each terminal node  $x$  in  $T$ , a set  $Spec_x$  is stored. For each  $q \in W'_i$ , if  $q$  occurs only at terminal node  $x$  then it is added to  $Spec_x$ . Otherwise, if  $q$  occurs at more than one terminal node in  $T$  then  $q$  is added to the set  $Com_y$ , where  $y$  is the least common ancestor of those terminal nodes at which  $q$  occurs. Clearly, all  $Com$  and  $Spec$  sets are disjoint and therefore, the total space taken by them is  $O(|W'_i|)$ . The following lemma shows how  $Q_x$  can be retrieved from the  $Com$  and  $Spec$  sets, for each terminal node  $x$  in  $T$ .

**Lemma 4.29** *For each terminal node  $x \in T$ ,  $Q_x = (W_i - W'_i) \cup Com_{y_1} \cup Com_{y_2} \cup \dots \cup Com_{y_j} \cup Spec_x$ , where  $y_1, \dots, y_j$  are the proper ancestors of  $x$  in  $T$ .*

**Proof:** Follows immediately from Lemma 4.26.  $\square$ .

To compute  $Q_x$  as an ordered list, it suffices to maintain each of the  $Y$ ,  $Z$ ,  $Com$  and  $Spec$  sets as ordered lists which are then appended together in  $O(m)$  time according to either Lemma 4.27 or Lemma 4.29, as the case may be.

This concludes the data structure description. We remark that all the data structures mentioned at the beginning of this section can be computed using naive algorithms in  $O(m^2)$  time.

## 5 The Transfer function $f$

Before giving the definition of the function  $f$ , we prove a number of preliminary lemmas.

### 5.1 Preliminary Lemmas

These lemmas describe some properties of periodic strings and the distribution of text characters compared in Step 2 (the elimination tree phase) of the presuf handler described in Section 4.2.

Let  $V = \{p_1, p_2, \dots, p_k, p_{k+1}\}$ . Consider the set of pattern instances in  $V$  which are right-most in their respective groups. Let  $p_i$  be a pattern instance in this set. We introduce a function  $h(x_i)$  which is central to the analysis.

**Definition.** If  $i < k$  then  $h(x_i)$  is defined by one of the following three cases.

1.  $x_i$  is periodic. Then  $x_{i+1}$  is periodic too. Let  $u$  and  $v$  be the head and core, respectively, of  $x_i$ . Let  $w$  be the core of  $x_{i+1}$ .  $h(x_i)$  is defined to be the suffix of  $p_1$  of length  $|v| + |w|$ .
2.  $x_i = uvu$  is not periodic, where  $|u|$  is its s-period. Further,  $x_{i+1}$  is periodic with core  $w$ .  $h(x_i)$  is defined to be the suffix of  $p_1$  of length  $|v| + |u| + |w|$ .
3.  $x_i = uvu$  is not periodic, where  $|u|$  is its s-period. Further,  $x_{i+1}$  is not periodic.  $h(x_i)$  is defined to be the suffix of  $p_1$  of length  $|u|$ .

If  $i = k + 1$  then  $h(x_i)$  is defined to be the empty string. Note that  $i \neq k$  as  $p_k$  and  $p_{k+1}$  are both in the same group.

The first two lemmas consider the case when  $i < k$  and  $x_{i+1}$  is periodic with core  $w$ . They show that  $h(x_i)$  cannot be periodic with core  $w$ .

**Lemma 5.1** *Suppose  $x_i = uv^2$ , where  $v$  is the core of  $x_i$ . Further, suppose  $x_{i+1} = uv = w'w^{k_1}$  is periodic with core  $w$ ,  $|w| < |v|$ . Then  $h(x_i)$  is not periodic with core  $w$ .*

**Proof:**  $w$  is a suffix of  $v$ . Since  $v$  is primitive,  $|v|$  is not a multiple of  $|w|$ . If  $h(x_i)$  were periodic with core  $w$  then the prefix of  $h(x_i)$  of size  $|w|$  would have the form  $xy$ ,  $x$  a proper suffix of  $w$  and  $y$  a proper prefix of  $w$ . But this prefix of  $h(x_i)$  is a suffix of  $v$  and hence is the string  $w$ . This implies that  $w$  is cyclic and cannot be the core of  $x_{i+1}$ , a contradiction.  $\square$

**Lemma 5.2** *Suppose  $x_i = uvu$  is not periodic, where  $|u|$  is the s-period of  $x_i$ . Suppose the string  $x_{i+1} = u = w'w^{k_1}$  is periodic with core  $w$ ,  $|w| < |u|$ . Then  $h(x_i)$  is not periodic with core  $w$ .*

**Proof:**  $w$  is a suffix of  $u$ .  $vu$  is primitive otherwise  $x_i$  would be periodic. Suppose  $h(x_i)$  is periodic with core  $w$ . Then  $|vu|$  is not a multiple of  $|w|$ . Therefore, the prefix of  $h(x_i)$  of size  $|w|$  is of the form  $xy$ ,  $x$  a proper suffix and  $y$  a proper prefix of  $w$ . But this prefix of  $h(x_i)$  is a suffix of  $u$  and hence is the string  $w$ . This implies that  $w$  is cyclic and cannot be the core of  $x_{i+1}$ , a contradiction.  $\square$

The next lemma describes the order in which pattern instances in a half-done set are eliminated in Step 2 of the presuf shift handler.

**Lemma 5.3** *Let  $p_{i_1}, \dots, p_{i_r}$ ,  $r \geq 3$ , be pattern instances in  $V$  comprising a half-done set. For any  $l$ ,  $3 \leq l \leq r$ , if  $p_{i_1}$  and  $p_{i_l}$  both survive at any instant in Step 2 then  $p_{i_1}, \dots, p_{i_{l-1}}$  also survive at that instant.*

**Proof:** We show that the lemma is true for any instant in Phase 1 and at the end of Phase 1. For Phase 2, the lemma follows from Corollary 4.8.

Consider the rightmost position  $e$  such that  $p_{i_1}[e]$  is to the left of  $t_b$  (recall that  $t_b$  is the text character aligned with  $p_1[m]$ ) and different from the character in  $p_{i_l}$  aligned with it. The portions of  $p_{i_1}, \dots, p_{i_l}$  whose left and right ends are aligned with  $p_{i_1}[e+1]$  and  $t_b$ , respectively, are identical and periodic with core  $v$ , where  $v$  is the core of  $x_{i_1}$ . The portions of  $p_{i_1}, \dots, p_{i_{l-1}}$  whose left and right ends are aligned with  $p_{i_1}[e]$  and  $t_b$ , respectively, are identical. Therefore, a comparison to the right of  $p_{i_1}[e]$  eliminates none or all of  $p_{i_1}, \dots, p_{i_l}$  depending upon whether it succeeds or fails. A comparison at  $p_{i_1}[e]$  eliminates either  $p_{i_l}$  or all of  $p_{i_1}, \dots, p_{i_{l-1}}$ . Thus, if  $p_{i_1}$  and  $p_{i_l}$  survive at any instant in Phase 1 or at the end of Phase 1 then all comparisons made till that instant are to the right of  $p_{i_1}[e]$ . Each of these comparisons eliminates none or all of  $p_{i_1}, \dots, p_{i_l}$ .  $\square$

The next lemma establishes that if all comparisons in the suffix  $x_i$  of  $p_1$  are successful then at most two pattern instances to the right of  $p_i$  survive.

**Lemma 5.4** *Suppose all comparisons made by  $S_1$  within the suffix  $x_i$  of  $p_1$  result in matches. Then at most two instances in  $V$  among those lying to the right of  $p_i$  survive. Further, if two instances  $p_y$  and  $p_z$  survive, then  $\{x_i, x_y, x_z\}$  is a clone set.*

**Proof:** First, suppose  $x_i$  is periodic. Then, by the manner in which groups were defined,  $x_i$  has the form  $uv^2$ . Let  $p_a, p_b \in V$ ,  $a, b > i$ . If  $\{x_i, x_a, x_b\}$  is not a clone set then, by Lemma 4.5, successful comparisons in the suffix  $x_i$  of  $p_1$  suffice to eliminate one of  $p_a, p_b$ . Thus, two or more pattern instances in  $V$  to the right of  $p_i$  can survive only if their presufs form a clone set with  $x_i$ . But the only candidates are the pattern instances  $p_y$  and  $p_z$  whose presufs are  $uv$  and  $u$ , respectively.

Second, suppose  $x_i$  is not periodic. Then it is of the form  $uvu$ , where  $u$  is its s-period. For no two pattern instances  $p_y, p_z$ ,  $y, z > i$ , can  $\{x_i, x_y, x_z\}$  be a clone set. By Lemma 4.5, one of  $p_y, p_z$ , for every such  $y$  and  $z$ , can be eliminated by a successful comparison made within the suffix  $x_i$  of  $p_1$ . Thus in this case, at most one pattern instance in  $V$  to the right of  $p_i$  survives.  $\square$

The next two lemmas establish that if all comparisons within  $h(x_i)$  are successful then at most two pattern instances in  $V$  to the right of  $p_i$  survive.

**Lemma 5.5** *Suppose  $x_{i+1}$  is periodic with core  $w$  and all comparisons made by  $S_1$  within  $h(x_i)$  result in matches. Then at most two instances in  $V$  among those to the right of  $x_i$  survive.*

**Proof:** Since the case  $i = k + 1$  is vacuous, we assume that  $i < k$ .

The proof is based on Lemmas 5.1, 5.2, 5.3 and 5.4. Let  $A_s$  be the group containing  $p_i$ . Let  $p_{i+1}, \dots, p_j$  be the pattern instances in group  $A_{s+1}$ . Consider the set  $V'$  of pattern instances in  $V$  which are to the right of  $p_i$  and which survive successful comparisons in the suffix  $x_j$  of  $p_1$ . By Lemma 5.4, with at most one exception (call it  $p_o$ ), the pattern instances in  $V'$  form a half-done set. By Lemma 5.3, the presufs corresponding to the pattern instances in this half-done set comprise the set  $\{w'w^{k2}, \dots, w'w^{k3+1}, w'w^{k3}\}$ , where  $k3$  equals 0, 1 or 2. Let  $V' = \{p_{i_1}, p_{i_2}, \dots, p_{i_j}, p_o\}$ . We show that successful comparisons in  $h(x_i)$  eliminate all but at most one of  $\{p_{i_1}, p_{i_2}, \dots, p_{i_j}\}$ .

Note that  $\{p_{i_1}, p_{i_2}, \dots, p_{i_j}\}$  is a half-done set. By Lemmas 5.1 and 5.2, the suffix  $h(x_i)$  of  $p_1$  (and of  $x_i$ ) is not periodic with core  $w$ . Let the rightmost suffix of  $p_1$  which is longer than  $|x_{i+1}|$  and not periodic with core  $w$  begin at  $p_1[e]$ ;  $p_1[e]$  lies in  $h(x_i)$ . Consider the largest  $h$ ,  $1 \leq h \leq j$ , such that  $p_{i_h}$  survives all comparisons made to the right of  $p_1[e]$ . Then, by Lemma 5.3,  $p_{i_1}, \dots, p_{i_{h-1}}$  also survive these comparisons while  $p_{i_{h+1}}, \dots, p_{i_j}$  are eliminated. If  $h \leq 1$  then we are done. Otherwise, as shown in the next paragraph, the characters in  $p_{i_1}, \dots, p_{i_{h-1}}$  aligned with  $p_1[e]$  are identical to each other yet different from  $p_1[e]$ . Hence there will be a comparison involving  $p_1[e]$ , which by assumption is a match; this leaves only  $p_{i_h}$  and  $p_o$  uneliminated.

Since the rightmost eligible character is always chosen by the elimination strategy for comparison, the portions of  $p_{i_1}, \dots, p_{i_h}$  aligned with the suffix of  $p_1$  which lies to right of  $p_1[e]$  match that suffix. Suppose for some  $r$ ,  $1 \leq r < h$ ,  $a = p_{i_r}[c] \neq p_{i_r}[c + |w|] = b$ , where  $p_{i_r}[c]$  is aligned with  $p_1[e]$ . Since  $p_{i_{r+1}}$  is  $|w|$  units to the right of  $p_{i_r}$ , the character in  $p_{i_{r+1}}$  aligned with  $p_{i_r}[c + |w|] = b$  is an  $a$ , a contradiction. Therefore, the characters in  $p_{i_1}, \dots, p_{i_{h-1}}$  aligned with  $p_1[e]$  are all equal to the character  $p_1[e + |w|]$ . But, from the definition of  $e$ ,  $p_1[e + |w|] \neq p_1[e]$ . This proves the lemma.  $\square$

**Lemma 5.6** *Suppose  $x_{i+1}$  is not periodic and all comparisons made by  $S_1$  within the suffix  $h(x_i)$  of  $p_1$  result in matches. Then at most two instances in  $V$  among those to the right of  $p_i$  survive.*

**Proof:** Since the case  $i = k + 1$  is vacuous, we assume that  $i < k$ .

By the manner in which groups were defined,  $x_i$  is not periodic. Since  $x_{i+1}$  is not periodic,  $p_{i+1}$  is the rightmost instance in its group. Thus,  $x_{i+1}$  cannot form a clone set with any two of its presufs. By Lemma 5.4, at most one pattern instance to the right of  $p_{i+1}$  survives successful comparisons in the suffix  $x_{i+1} = h(x_i)$  of  $p_1$ .  $\square$

The following lemma relates the length of the presufs  $x_i$  and  $x_{i+2}$  with the suffix  $h(x_i)$  of  $p_1$  for  $i \leq k - 1$ .

**Lemma 5.7**  $|x_{i+2}| + |h(x_i)| \leq |x_i|$ .

**Proof:** First suppose  $x_i = uvu$  is not periodic, where  $u$  is its s-period. Then  $|x_{i+2}| < |u|$ . If  $x_{i+1}$  is not periodic then  $h(x_i) = u$  and  $|x_{i+2}| + |h(x_i)| < 2|u| < |x_i|$ . If  $x_{i+1}$  is periodic with core  $w$  then  $|h(x_i)| = |u| + |v| + |w|$  and  $x_{i+2} = |u| - |w|$ . This implies that  $|x_{i+2}| + |h(x_i)| = |x_i|$ .

Next, suppose  $x_i$  is periodic with core  $v$  and head  $u$ . Then  $x_{i+1}$  is also periodic, say with core  $w$ . Thus  $|h(x_i)| = |v| + |w|$  and  $|x_{i+2}| = |v| + |u| - |w|$ . Then  $|x_{i+2}| + |h(x_i)| = 2|v| + |u| = |x_i|$ .  $\square$

**Definitions.** Let the term *misfit* refer to any character that differs from the rightmost character of  $p$ . If  $|x_k| > 1$ , let  $r_i$  be the number of pattern instances in  $V$  which lie to the right of  $p_i$ . Otherwise, if  $|x_k| = 1$ , let  $r_i$  be one more than the number of pattern instances in  $V$  which lie to the right of  $p_i$  and do not belong to the rightmost group. For convenience, we define  $r_i$  to be 0 if  $|x_k| = 1$  and  $p_i$  belongs to the rightmost group.

We provide some lower bounds on the number of occurrences of misfit characters in the presuffixes of  $p$  and in the cores of periodic presuffixes.

**Lemma 5.8** *Let  $|x_k| > 1$ . Let  $p_j$ ,  $j \leq k$ , be any pattern instance in  $V$ . Then  $x_j$  contains at least  $r_j$  instances of the string  $x_k$  and hence  $r_j$  misfit characters.*

**Proof:** Since  $x_k$  is the smallest non-null suffix of  $p$  that matches a prefix of  $p$ , no non-null suffix of  $x_k$  matches a prefix of  $x_k$ . Hence all instances of  $x_k$  in any string are disjoint. Since  $x_k$  itself contains  $x_k$  and  $r_k = 1$ , the lemma is true for  $j = k$ . Next, suppose  $j < k$  and assume inductively that  $x_{j+1}$  contains at least  $r_{j+1}$  instances of  $x_k$ . Then since  $x_{j+1}$  is a proper prefix and a proper suffix of  $x_j$ ,  $x_j$  must contain at least  $r_{j+1} + 1 = r_j$  instances of  $x_k$ . Since the first character of  $x_k$  differs from its last character,  $x_j$  has at least  $r_j$  misfit characters.  $\square$

**Lemma 5.9** *Suppose  $|x_k| = 1$ . Let  $p_j$  be any pattern instance in  $V$ . Then  $x_j$  has at least  $r_j$  misfit characters.*

**Proof:** If  $p_j$  belongs to the rightmost group then  $r_j = 0$  and the lemma holds trivially. So suppose  $p_j$  is not in the rightmost group. Let  $p_y$  be the rightmost pattern instance not in the rightmost group.  $x_y$  contains at least one misfit character, otherwise it would be in the rightmost group. Since  $r_y = 1$ , the lemma is true for  $j = y$ . Next, assume that  $j < y$  and assume inductively that  $x_{j+1}$  contains at least  $r_{j+1}$  misfit characters. Then since  $x_{j+1}$  is a proper prefix and a proper suffix of  $x_j$ ,  $x_j$  must have at least  $r_{j+1} + 1 = r_j$  misfit characters.  $\square$

**Lemma 5.10** *Let  $p_j$  be any instance in  $V$  and suppose  $x_j$  is periodic with head  $u$  and core  $v$ ,  $|v| > 1$ . Then  $v$  contains a misfit character.*

**Proof:** If  $v$  does not contain a misfit character then  $x_j$  does not contain a misfit character either. This implies that all the characters in  $x_j$  are identical. This contradicts the assumption that  $|v| > 1$ .  $\square$

We conclude this section of lemmas with two key lemmas, the *h-Suffix Mapping Lemma* and the *Half-Done Set Mapping Lemma*. In the *h-Suffix Mapping Lemma*, a set  $R_1(i)$  of text characters is defined for each  $i$ ,  $1 \leq i \leq k - 1$ , such that  $p_i$  is the rightmost instance in its group. In the *Half-Done Set Mapping Lemma*, a set  $R_2(O)$  of text characters is defined for a half-done set  $O$  consisting of pattern instances from  $V$ . These two sets are used as ranges for the  $f$  function.

Recall that  $V = \{p_1, \dots, p_k\}$  and  $t_b$  is the text character aligned with the rightmost character of  $p_1$ .

Let  $i \leq k - 1$  and  $p_i$  be the rightmost instance in its group. Let  $h'_i$  be the suffix of length  $|x_{i+2}|$  of the prefix  $x_i$  of  $p$ . Let  $R_1(i)$  be the set of text characters with which  $|p$  is aligned when some misfit character in  $h'_i$  is aligned with  $t_b$ .

**Lemma 5.11** [The *h-Suffix Mapping Lemma*].  $|R_1(i)| \geq r_i - 2$ . *All text characters in  $R_1(i)$  lie strictly to the left of  $|h(x_i)|$  but within the suffix  $x_i$  of  $p_1$ .*

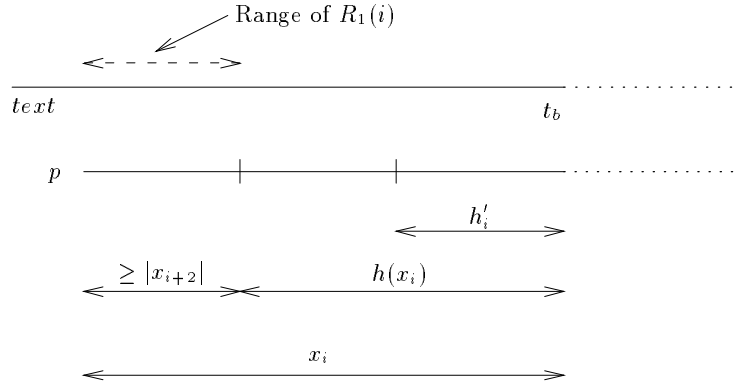


Figure 5: The  $h$ -Suffix Mapping Lemma

**Proof:** (See Fig.5.) By Lemmas 5.8 and 5.9,  $x_{i+2}$  and hence  $h'_i$  contain at least  $r_{i-2} = \max\{0, r_i - 2\}$  misfit characters. Therefore,  $|R_1(i)| \geq r_i - 2$ . By Lemma 5.7, the left end of any pattern instance in which  $h'_i$  overlaps  $t_b$  is strictly to the left of  $|h(x_i)|$  and within the suffix  $x_i$  of  $p_1$ .  $\square$

**Definition.** Let  $O \subset V$  be a half-done set consisting of the pattern instances  $\{p_{h_1}, \dots, p_{h_j}\}$ ,  $j \geq 3$ ,  $p_{h_1} = p_1$ . Let the head and core of  $x_{h_1}$  be denoted  $u$  and  $v$ , respectively. Let  $v = u'u$ . Suppose  $|v| > 1$ . Further, suppose  $p_{h_i}$  is  $p_{h_{i-1}}$  shifted distance  $|v|$  to the right, for  $1 < i \leq j$ . Let  $i_c$ ,  $2 \leq c \leq j$ , be the largest index such that  $p_{h_1}[i_c]$  is different from the character in  $p_{h_c}$  aligned with it (such an index exists by Lemma 4.7). Note that  $i_c - i_{c-1} = |v|$ , for all  $c$ ,  $3 \leq c \leq j$ , and that  $p_{h_1}[i_2] = p_{h_1}[i_3] = \dots = p_{h_1}[i_j]$ . The text character  $t_{i_c}$  aligned with  $p_{h_1}[i_c]$  is called the *characteristic character* of  $p_{h_c}$ .

Let  $d$  be the leftmost character in the prefix  $uu'$  of  $p$  which differs from  $p_{h_1}[i_2]$ . Define  $R_2(t_{i_c})$ ,  $3 \leq c \leq j$ , to be the text character with which  $|p$  is aligned when  $d$  is aligned with  $t_{i_c}$ . In addition, define  $R_2(O_e)$  to be the set of text characters  $R_2(t_{i_c})$ ,  $3 \leq c \leq e \leq j$ . For convenience, let  $R_2(O)$  denote  $R_2(O_j)$ .

**Lemma 5.12** [The Half-Done Set Mapping Lemma] *All text characters in  $R_2(O_j)$  are distinct.  $R_2(t_{i_c})$  is aligned with or to the left of  $t_{i_c}$  and strictly to the right of  $t_{i_{c-1}}$ , for  $3 \leq c \leq j$ . All characters in  $R_2(O_c)$  are aligned with or to the left of the characteristic character of  $p_{h_c}$ ; in addition, they are strictly to the right of  $|p_{k+1}|$ , for  $3 \leq c \leq j$ .*

**Proof:** By construction,  $R_2(t_{i_c})$  is aligned with or to the left of  $t_{i_c}$ ,  $3 \leq c \leq j$ . In addition,  $R_2(t_{i_c})$  is at most distance  $|v| - 1$  to the left of  $t_{i_c}$ . As  $i_c - i_{c-1} = |v|$ ,  $R_2(t_{i_c})$  is strictly to the right of  $t_{i_{c-1}}$ .

The only part of the lemma still unproven is the claim that all characters in  $R_2(O_j)$  are strictly to the right of  $|p_{k+1}|$ . Note that  $t_{i_3}$  is distance  $|v|$  to the right of  $t_{i_2}$ ,  $R_2(t_{i_3})$  is at most distance  $|v| - 1$  to the left of  $t_{i_3}$ , and all characters in  $R_2(O_j)$  are aligned with or to the right of  $R_2(t_{i_3})$ . Since  $t_{i_2}$  is aligned with or to the right of  $|p_{k+1}|$ , the lemma follows.  $\square$

Note that  $p_{h_c}$  is eliminated by the time a comparison is made strictly to the left of its characteristic character, for  $3 \leq c \leq j$ . Further, if a successful comparison eliminates  $p_{h_c}$  then this comparison must involve its characteristic character.

## 5.2 The Transfer Function $f$

Let  $C$  be the set of text characters involved in comparisons in Steps 1 and 2 of the presuf shift handler of Section 4.2. For each character  $t_c \in C$ , with at most two exceptions, we define  $f(t_c)$  to be a text character  $t_d$  satisfying the following properties.

1.  $t_d$  is to the right of  $|p_{k+1}$ .
2.  $t_d$  either coincides with  $t_c$  or lies to the left of  $t_c$ .
3. The pattern instance whose left end is aligned with  $t_d$  is eliminated as a result of comparisons in Steps 1 and 2 of the presuf shift handler.
4. For every distinct  $t_{c_1}, t_{c_2} \in C$ ,  $f(t_{c_1}) \neq f(t_{c_2})$ .

Furthermore, the mismatches, if any, are always included among the exceptions. We refer to the above properties as Properties 1, 2, 3 and 4, respectively.

Since patterns with  $g = 1$  and  $|x_k| = 1$  are special case patterns, we assume that  $g > 1$  if  $|x_k| = 1$ . Further, if  $p_1[m]$  does not match the text, then Steps 1 and 2 of the presuf shift handler together make at most one comparison. Therefore, we also assume that  $p_1[m]$  matches the text. Let  $p_l$  be the rightmost pattern instance in  $A_1$ . Let  $p_r$  be the rightmost pattern instance in  $V$  outside  $A_g$ , if any.

We split the sequence  $C'$  of comparisons made in Steps 1 and 2 of the presuf shift handler into three disjoint classes as follows.

1. Class 1 consists of the comparison in Step 1. In addition, if  $|x_k| = 1$ , then Class 1 contains the comparisons which comprise the smallest prefix of  $C'$  having the following property: either the last comparison in this prefix is unsuccessful or following that comparison, exactly one pattern instance in  $A_g$  survives.
2. Class 2 consists of the comparisons in  $C'$  which follow all Class 1 comparisons and are made in the suffix  $h(x_l)$  of  $p_1$ .
3. Class 3 consists of comparisons in  $C'$  which follow all Class 2 comparisons.

Note that if Class 1 contains an unsuccessful comparison then Class 2 is empty because no further comparisons are made in the suffix  $h(x_l)$  of  $p_1$ . Thus Class 1 and Class 2 together have at most one unsuccessful comparison. The only other possibly unsuccessful comparison is the last comparison in Class 3. We do not define an  $f$  value for the last comparison in Class 3. In addition, one other comparison may not receive an  $f$  value. If Class 1 and Class 2 contain an unsuccessful comparison then this comparison does not receive an  $f$  value. If all comparisons in Classes 1 and 2 are successful then one successful comparison in one of the three classes may not receive an  $f$  value. All other comparisons receive  $f$  values. Thus,  $f$  values are never defined for mismatches and at most two comparisons in  $C'$  do not receive  $f$  values.

We define  $f$  values for each class in turn.  $f$  values for Class 2 comparisons are always defined using the set  $R_1(l)$ . These  $f$  values are aligned with the suffix  $x_l$  of  $p_1$  and to the left of  $h(x_l)$ .  $f$  values for Class 3 comparisons are defined in one of three ways. If all comparisons in Classes 1 and 2 are successful then these  $f$  values are to the left of the suffix  $x_l$  of  $p_1$ . If Class 2 contains a mismatch then these  $f$  values are defined using the set  $R_1(l)$ . If Class 2 is empty and Class 1 contains a mismatch then these  $f$  values are aligned with or to the right of



the suffix  $x_{r+1}$  of  $p_1$ .  $f$  values for Classes 2 and 3 are easily seen to be distinct.  $f$  values for Class 1 comparisons are aligned either with the suffix  $x_r$  of  $p_1$  or with the suffix  $x_{r-1}$  of  $p_1$ ; in Lemma 5.17, we show that these  $f$  values do not clash with the  $f$  values for Classes 2 and 3.

**Classes 2 and 3.** We consider three cases.

**Case 1.** Class 2 contains an unsuccessful comparison.

Class 2 and Class 3 together contain at most  $r_l - 1$  comparisons in addition to this unsuccessful comparison. To see this, note that  $r_l - 1$  comparisons in addition to the comparisons in Class 1 suffice to eliminate all but one of the pattern instances in  $V$  to the right of  $p_l$ . Further, excluding the unsuccessful Class 2 comparison and the last comparison in Class 3, all other comparisons in Classes 2 and 3 are successful.  $f$  is defined to map the text characters involved in these  $r_l - 2$  successful comparisons to the text characters in  $R_1(l)$  in some arbitrary order. By the  $h$ -Suffix Mapping Lemma and the fact that all Class 3 comparisons are to the right of  $p_1[m]$  in this case, all text characters in  $R_1(l)$  lie to the left of all the text characters involved in Class 2 and Class 3 comparisons. Clearly, Properties 2, 3 and 4 are true for these  $f$  values. Property 1 follows from the fact that  $|x_l| \leq |x_1| < \frac{m}{2}$ , and hence  $|p_{k+1}|$  is to the left of the suffix  $x_l$  of  $p_1$ .

**Case 2.** All comparisons in Classes 1 and 2 are successful.

There are at most  $r_l$  comparisons in Class 2, all of which are successful.  $f$  is defined to map the text characters involved in  $r_l - 2$  of these  $r_l$  comparisons to the text characters in  $R_1(l)$  in some arbitrary order. As in Case 1, Properties 1, 2, 3 and 4 are satisfied by these  $f$  values. This leaves at most  $s$  Class 2 comparisons for some  $s \leq 2$ ,

Next, we define  $f$  values for Class 3 comparisons and  $s$  Class 2 comparisons. These  $f$  values will be defined for all comparisons in Class 3 plus the  $s$  comparisons in Class 2, with at most two exceptions. These  $f$  values will be to the left of the suffix  $x_l$  of  $p_1$ , and thus clearly distinct from  $f$  values for Class 2 comparisons.

Following Class 2 comparisons, at most  $\min\{r_l, 2\} - s$  of the pattern instances to the right of  $p_l$  survive along with pattern instances in  $A_1$ . Let  $O'$  denote the following set of  $\min\{r_l, 2\}$  pattern instances: those pattern instances to the right of  $p_l$  which survive Class 1 and Class 2 comparisons and those pattern instances which are eliminated by one of the  $s$  Class 2 comparisons under consideration. Let  $O$  refer to the largest half-done set consisting of pattern instances in  $A_1$  and  $O'$ . Redefine  $O'$  by removing pattern instances in it which are also in  $O$ . Considering comparisons which eliminate pattern instances in  $O$  and  $O'$  is equivalent to considering Class 3 comparisons plus the  $s$  Class 2 comparisons.

Let  $O = \{p_{h_1}, \dots, p_{h_e}\}$ . If  $l = 1$ , then the number of comparisons in Class 3 plus  $s$  is at most  $2 - s + s = 2$ . In this case, we do not define  $f$  values for the comparisons in Class 3 and the  $s$  comparisons in Class 2. So suppose that  $l > 1$ . Each successful comparison which eliminates a pattern instance in  $O$  involves the characteristic character of the pattern instance eliminated. Let  $v$  and  $u$  be the core and head, respectively, of  $x_{h_1}$  and let  $v = u'u$ .  $|v| > 1$  because either  $|x_k| > 1$  or  $|x_k| = 1$  and  $g > 1$ . By Lemma 5.10,  $v$  contains a misfit character.

First, consider successful comparisons which eliminate pattern instances in  $O$ . If  $|O| \leq 2$ , there is at most one such comparison in Class 3 and we do not define an  $f$  value for it. So suppose  $|O| > 2$ . There are two subcases depending on the location of the characteristic character  $t_{i_e}$  of  $p_{h_e}$ .

**Subcase 2a.** Either  $O'$  is not empty and  $t_{i_e}$  is strictly to the left of the left end of the suffix  $x_{h_{e-1}}$  of  $p_1$ ; or  $O'$  is empty and  $t_{i_e}$  is strictly to the left of the left end of the suffix  $x_{h_{e-2}}$  of  $p_1$ .

For  $3 \leq c \leq e$ , if a successful comparison is made at  $t_{i_c}$ ,  $f(t_{i_c})$  is defined to be  $R_2(t_{i_c})$ . By the Half-Done Set Mapping Lemma, these  $f$  values are strictly to the left of the suffix  $x_{h_{e-1}}$  of  $p_1$  if  $O'$  is not empty and strictly to the left of the suffix  $x_{h_{e-2}}$  of  $p_1$  if  $O'$  is empty. A simple case analysis ( $O'$  equals 0, 1, 2) shows that these  $f$  values are strictly to the left of the left end of the suffix  $x_l$  of  $p_1$ , as claimed. Properties 1, 2 and 4 for these  $f$  values follow easily from the Half-Done Set Mapping Lemma while Property 3 follows from the definition of the set  $R_2(O)$ . At most one successful comparison eliminating pattern instances in  $O$  does not have an  $f$  value: the one eliminating  $p_{h_2}$ .

**Subcase 2b.** Either  $O'$  is not empty and  $t_{i_e}$  is aligned with some character in the suffix  $x_{h_{e-1}}$  of  $p_1$ ; or  $O'$  is empty and the characteristic character of  $p_{h_e}$  is aligned with some character in the suffix  $x_{h_{e-2}}$  of  $p_1$ .

In the first case,  $t_{i_c}$ , the characteristic character of  $p_{h_c}$ , is aligned with some character in the suffix  $x_{h_{c-1}}$  of  $p_1$ , for  $2 \leq c \leq e$ . Consider the set  $R'_2$  of  $e - 2$  text characters with which  $|p$  is aligned when the rightmost misfit character in the prefix  $x_{h_c}$  of  $p$ ,  $1 \leq c \leq e - 2$ , is aligned with  $t_b$ . Since  $v$  contains a misfit character, the  $c$ th leftmost text character in  $R'_2$  is aligned with some character in the suffix  $x_{h_c}$  of  $p_1$  and is strictly to the left of the left end of the suffix  $x_{h_{c+1}}$  of  $p_1$ . A successful comparison at  $t_{i_c}$ ,  $3 \leq c \leq e$ , is mapped by  $f$  to the  $(c - 2)$ nd leftmost character in  $R'_2$ . Clearly, all characters in  $R'_2$  are distinct and  $f(t_{i_c})$  is strictly to the left of  $t_{i_c}$ . All characters in  $R'_2$  are aligned with some character in the suffix  $x_1$  of  $p_1$ . Thus, Properties 1, 2, 3 and 4 are satisfied by these  $f$  values. These  $f$  values are strictly to the left of the left end of the suffix  $x_{h_{e-1}}$  of  $p_1$ . Since  $O'$  is not empty,  $h_{e-1} \leq l$ . Therefore, these  $f$  values are strictly to the left of the left end of the suffix  $x_l$  of  $p_1$ . Again, the only successful comparison without an  $f$  value, if any, is the one eliminating  $p_{h_2}$ .

In the second case,  $t_{i_c}$ ,  $4 \leq c \leq e$ , is aligned with some character in the suffix  $x_{h_{c-2}}$  of  $p_1$ .  $f$  values are not defined for the two leftmost comparisons under consideration. The remaining comparisons involve text characters aligned with some character in the suffix  $x_{h_2}$  of  $p_1$ . A successful comparison at  $t_{i_c}$ ,  $4 \leq c \leq e$ , is mapped by  $f$  to the  $(c - 3)$ rd leftmost character in  $R'_2$ . Clearly,  $f(t_{i_c})$  is strictly to the left of  $t_{i_c}$ . As in the first case, Properties 1, 2, 3 and 4 are satisfied by these  $f$  values. All these  $f$  values are to the left of the left end of the suffix  $x_{h_{e-2}}$  of  $p_1$ . Since  $h_{e-2} \leq l$  for this case, these  $f$  values are strictly to the left of the left end of the suffix  $x_l$  of  $p_1$ . The only successful comparisons without  $f$  values, if any, are those eliminating  $p_{h_3}$  and  $p_{h_2}$ . This ends Subcase 2.1b.

Before, we define  $f$  values for comparisons which eliminate pattern instances in  $O'$ , we need a lemma which will be used later when Class 3 comparisons are defined. This lemma can be verified easily from the above description.

**Lemma 5.13** *If  $O \subset A_1$  and  $|O'| \leq 1$  then the  $f$  values defined in Subcases 2a and 2b are to the left of the suffix  $x_{l-1}$  of  $p_1$ .*

Next, consider successful comparisons which eliminate pattern instances in  $O'$ . If  $|O'| < 2$  or no successful comparison eliminates a pattern instance in  $O'$  then no further  $f$  values are defined. So suppose  $|O'| = 2$  and a successful comparison is made to eliminate one of the pattern instances in  $O'$ . By Lemma 4.5, this comparison involves a text character  $t_c$  which is aligned with some character in the suffix  $x_{h_e}$  of  $p_1$ .  $f(t_c)$  is defined to be the text character with which  $|p$  is aligned when the rightmost misfit character in the prefix  $x_{h_{e-1}}$  of  $p_1$  is aligned with  $t_b$ . Since  $v$  contains a misfit character,  $f(t_c)$  is aligned with some character in the suffix  $x_{h_{e-1}}$  of  $p_1$

and is strictly to the left of the suffix  $x_{h_e}$  of  $p_1$ .  $f(t_c)$  is thus to the right of and distinct from all  $f$  values defined previously for comparisons which eliminate pattern instances in  $O$ . Further,  $f(t_c)$  is to the left of  $t_c$  because  $t_c$  is aligned with the suffix  $x_{h_e}$  of  $p_1$ . Since  $|O'| = 2$ ,  $l = h_e$ , and therefore  $f(t_c)$  is strictly to the left of the suffix  $x_l$  of  $p_1$ , as claimed. Properties 1, 2, 3 and 4 are easily seen to be true for all Class 2 and 3 comparisons now.

**Lemma 5.14** *For Case 2,  $f$  values have been defined for all but two of the comparisons in Class 2 and 3. Further, the omitted comparisons include mismatches, if any.*

**Proof:** We just need to show that at most two of the comparisons amongst those which eliminate pattern instances in  $O$  and  $O'$  do not receive  $f$  values, for the mismatches never receive  $f$  values.

If  $|O| < 2$  then there are at most 2 comparisons which eliminate pattern instances in  $O$  and  $O'$ . If  $O'$  is empty, then  $f$  values are defined for all but the last two comparisons which eliminate pattern instances in  $O$ . So suppose  $O'$  is not empty and  $|O| \geq 2$ . The only possible successful comparisons for which an  $f$  value might not be defined are those which eliminate  $p_{h_2}$  or one of the pattern instances in  $O'$ . There are two cases.

First, suppose  $|O'| = 1$ . Let  $O' = \{p_z\}$ . The only possible successful comparisons for which an  $f$  value is not defined are those which eliminate  $p_{h_2}$  or  $p_z$ . We show that if one of these successful comparisons actually occurs, then there can be at most one mismatch and if both these successful comparisons occur then there are no mismatches (recall that all comparisons in Classes 1 and 2 are successful). Suppose  $p_{h_2}$  is eliminated by a successful comparison.  $p_{h_1}$  must be alive immediately before this comparison and  $p_{h_3} \dots p_{h_e}$  must have been eliminated prior to this comparison. This implies that no mismatch could have occurred before this comparison and only the pattern instances  $p_{h_1}$  and  $p_z$  survive this comparison. Therefore, if  $p_{h_2}$  is eliminated by a successful comparison then there is at most one unsuccessful comparison and if both  $p_{h_2}$  and  $p_z$  are eliminated by successful comparisons then there are no unsuccessful comparisons. Next, suppose  $p_z$  is eliminated by a successful comparison but no successful comparison eliminates  $p_{h_2}$ . Each of the other comparisons in Class 3 eliminates some pattern instance in  $O$  and the first such unsuccessful comparison eliminates all but one of the instances in  $O$ . Therefore, there is at most one unsuccessful comparison in this case.

Second, suppose,  $|O| \geq 2$  and  $|O'| = 2$ . If one of the pattern instances in  $O'$  is eliminated by a successful Class 2 or Class 3 comparison then an  $f$  value is defined for this comparison. From this point onwards,  $|O| \geq 2$  and  $|O'| = 1$ . Therefore, the argument in the previous paragraph applies. On the other hand, if no successful Class 2 or 3 comparison eliminates a pattern instance in  $O'$  then the first comparison in Class 3 must be unsuccessful. This comparison leaves at most two pattern instances uneliminated and thus there are at most two comparisons which eliminate pattern instances in  $O$  and  $O'$ .  $\square$

**Case 3.** Class 1 contains an unsuccessful comparison.

In this case,  $|x_k| = 1$  and Class 2 is empty as mentioned before. We define  $f$  values for all but the last of the comparisons in Class 3. These  $f$  values are aligned with or to the right of the suffix  $x_{r+1}$  of  $p_1$ . All Class 3 comparisons are made to the right of  $p_1[m]$  in this case. Each such successful comparison matches an instance of the character  $x_k$  in  $p_{r+1}$  against a text character  $t_c$ ;  $t_c$  is aligned with a non- $x_k$  character in some pattern instance  $p_s$ ,  $s > r$ .  $f$  is defined to map a text character  $t_c$  matched successfully by a Class 3 comparison to the text character with which  $|p$  is aligned when the leftmost non- $x_k$  character in  $p$  is aligned with  $t_c$ . Clearly, these  $f$  values are aligned with or to the right of the suffix  $x_{r+1}$  of  $p_1$  and Properties 1, 2, 3 and 4 are satisfied by these  $f$  values.

This finishes the description of the  $f$  function for Classes 2 and 3. The following lemma is obvious from the above description.

**Lemma 5.15**  *$f$  values for Class 2 and 3 comparisons belongs to one of the following sets of text characters:*

- (i) *The set  $R_1(l)$ .*
- (ii) *The set of text characters to the left of the suffix  $x_l$  of  $p_1$  and to the right of  $|p_{k+1}$ .*
- (iii) *The set of text characters aligned with or to the right of the suffix  $x_{r+1}$  of  $p_1$ .*

*Further, an  $f$  value can be in set (i) only if  $r_l - 2 > 0$  and in set (iii) only if Class 1 contains an unsuccessful comparison.*

**Class 1.** We consider two cases,  $|x_k| > 1$  and  $|x_k| = 1$ .

**Case 1.**  $|x_k| > 1$ .

The only comparison in Class 1 matches  $t_b$  with  $p_1[m]$ .  $f(t_b)$  is defined to be  $t_b$ . This mapping satisfies Property 3 because the leftmost character in  $p$  is a misfit character in this case. If  $g = 1$  then all other comparisons in  $C'$  are made to the left of  $t_b$  and therefore, all other  $f$  values are to the left of  $t_b$ . If  $g > 1$ , then all other  $f$  values are either to the left of the suffix  $x_l$  of  $p_1$  or to the left of the suffix  $h(x_l)$  of  $p_1$ . Since  $|h(x_l)| \geq 1$  if  $g > 1$ , these  $f$  values are to the left of  $t_b$ . Therefore, Property 4 is satisfied by all  $f$  values. Property 1 and 2 are obvious for  $f(t_b)$ .

**Case 2.**  $|x_k| = 1$ .

$g > 1$ , by assumption.  $f$  values are defined for all comparisons in Class 1 unless either Class 1 contains an unsuccessful comparison or  $r = l = 1$ . If Class 1 contains an unsuccessful comparison or if  $r = l = 1$  then one comparison in Class 1 does not receive an  $f$  value. But in these cases, there is at most one comparison in Classes 2 and 3 for which an  $f$  value was not defined earlier. To see this, note that if the last Class 1 comparison is successful and  $r = l = 1$  then Classes 2 and 3 together can have at most one comparison and if the last Class 1 comparison is unsuccessful then Class 2 is empty and Case 3 must hold for Class 3 comparisons.

All  $f$  values defined for Class 1 comparisons will be to the left of the suffix  $x_{r+1}$  of  $p_1$  and aligned with either the suffix  $x_{r-1}$  of  $p_1$  or the suffix  $x_r$  of  $p_1$ . Clearly, if  $p_l \neq p_r, p_{r-1}$  then these  $f$  values are distinct from all  $f$  values defined earlier for Class 2 and 3 comparisons. If  $p_l = p_{r-1}$  then  $r_l - 2 = 0$  and all  $f$  values for Class 2 and 3 comparisons are either to the left of the suffix  $x_{r-1}$  of  $p_1$  or aligned with or to the right of the suffix  $x_{r+1}$  of  $p_1$ . Therefore,  $f$  values for comparisons in Class 1 are distinct from  $f$  values for comparisons in Classes 2 and 3 in this case. If  $p_l = p_r$  then  $r_l - 2 < 0$  and, by Lemma 5.15, all  $f$  values for Class 2 and 3 comparisons are either to the left of the suffix  $x_r$  of  $p_1$  or aligned with or to the right of the suffix  $x_{r+1}$  of  $p_1$ . In this case, we show that if an  $f$  value for some Class 1 comparison is aligned with the suffix  $x_{r-1}$  of  $p_1$  and to the left of the suffix  $x_r$  of  $p_1$  then all  $f$  values for Class 2 and Class 3 comparisons are to the left of the suffix  $x_{r-1}$  of  $p_1$ . Thus, all  $f$  values are distinct.

The following lemma describes the distribution of Class 1 comparisons.

**Lemma 5.16** *Let  $d$  be the rightmost misfit character in  $p_1$ . Each Class 1 comparison involves a text character which is aligned with or to the right of  $d$ .*

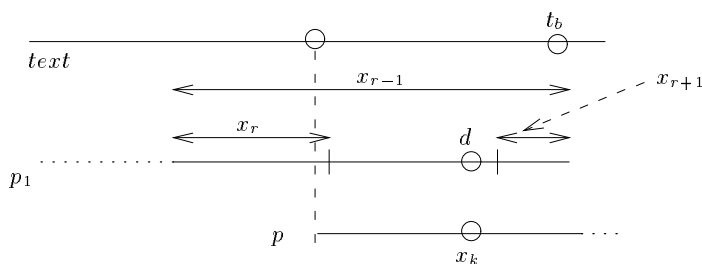


Figure 6: The Set  $R_3$ ,  $r > 1$

**Proof:** Suppose two pattern instances  $p_{j_1}, p_{j_2} \in A_g$ ,  $j_1 < j_2$ , are left uneliminated by comparisons made at or to the right of  $d$ . Let  $c_{j_1}$  and  $c_{j_2}$  be the portions of  $p_{j_1}$  and  $p_{j_2}$ , respectively, which overlap the suffix  $z$  of  $p_1$  starting at  $d$ . Then  $c_{j_1} = c_{j_2} = z$ . Consider the characters in  $p_{j_1}$  and  $p_{j_2}$  aligned with the  $(j_2 - j_1)$ th character to the right of  $d$  in  $p_1$ . Clearly, the first of these matches  $x_k$  while the second is a misfit character. This is a contradiction.  $\square$

Note that  $x_r$  contains a misfit character. Further, its suffix and prefix  $x_{r+1}$  are disjoint. Each contains at least  $k - r$  instances of  $x_k$ . We define a set  $R_3$  of text characters which serves as the range of  $f$  values for Class 1 comparisons. The definition has the following property. All characters in  $R_3$  are to the left of the suffix  $x_{r+1}$  of  $p_1$ . If all successful Class 1 comparisons are made to the right of  $d$  or if  $r = 1$  then all characters in  $R_3$  are aligned with the suffix  $x_r$  of  $p_1$ . If a successful Class 1 comparison is made at  $d$  and  $r > 1$  then characters in  $R_3$  are aligned with the suffix  $x_{r-1}$  of  $p_1$ .

First suppose all successful Class 1 comparisons are made to the right of  $d$ . Each successful comparison matches an occurrence of  $x_k$  to the right of  $d$  against the text.  $R_3$  is defined to be the set of text characters with which  $|p$  is aligned when  $d'$ , the leftmost misfit character in  $p$ , is aligned with one of the text characters matched by a Class 1 comparison. Clearly, all characters in  $R_3$  are aligned with the suffix  $x_r$  of  $p_1$ .  $f$  is defined to map the text characters compared by Class 1 comparisons to the text characters in  $R_3$  in some arbitrary order. Properties 2, 3 and 4 readily follow for these  $f$  values. Property 1 follows from the fact that  $|x_r| \leq |x_1| < \frac{m}{2}$ .

Next, suppose a successful Class 1 comparison is made at  $d$ . In this case, all comparisons in Class 1 are successful and there are at most  $k + 1 - r$  comparisons in Class 1.  $R_3$  is defined differently depending upon whether  $r = 1$  or  $r > 1$ .

First, suppose  $r = 1$ .  $R_3$  is defined to contain the  $k - r$  text characters with which  $|p$  is aligned when one of  $k - r$  instances of  $x_k$  to the left of  $d'$  (recall  $d'$  is the leftmost misfit character in  $p$ ) is aligned with  $d$ . The text characters in  $R_3$  are clearly aligned with the suffix  $x_r$  of  $p_1$ .  $f$  is defined to map up to  $k - r$  of the text characters compared by Class 1 comparisons to the  $k - r$  text characters in  $R_3$  in some arbitrary order. All these  $f$  values are distinct and are aligned with or to the left of  $d$ . Properties 2, 3 and 4 immediately follow for these  $f$  values. Property 1 follows from the fact that  $|x_r| \leq |x_1| < \frac{m}{2}$ .

Next, suppose  $r > 1$ . See Fig.6. When  $p$  is placed with  $|p$  aligned with the left end of the suffix  $x_{r-1}$  of  $p_1$ , there exist at least  $2(k - r) \geq k - r + 1$  instances of  $x_k$  to the left of  $d$  in  $p$ .  $R_3$  is defined to be the set of  $2(k - r)$  text characters with which  $|p$  is aligned when one of these  $2(k - r)$  instances of  $x_k$  is aligned with  $d$ . Clearly, characters in  $R_3$  are aligned with the suffix

$x_{r-1}$  of  $p_1$ .  $f$  is defined to map the text characters compared by Class 1 comparisons to some  $k - r + 1$  of the  $2(k - r)$  text characters in  $R_3$  is some arbitrary order. Properties 2 and 3 readily follow for these  $f$  values. Property 1 follows from the fact that  $|x_{r-1}| \leq |x_1| < \frac{m}{2}$ . Distinctness of these  $f$  values from the  $f$  values for Classes 2 and 3 follows from the following lemma.

**Lemma 5.17** *If  $r > 1$ ,  $p_l = p_r$  and a successful Class 1 comparison is made at  $d$  then  $f$  values for Class 2 and 3 comparisons are to the left of the suffix  $x_{r-1}$  of  $p_1$ .*

**Proof:** At most one pattern instance  $p_{r'} \in A_g$  survives a successful comparison at  $d$ . Further, Class 1 does not contain an unsuccessful comparison. Since  $r_l = 1$ , Class 2 contains at most one comparison. If this comparison is unsuccessful then only  $p_{r'}$  survives; no  $f$  values are defined for Class 2 or 3 comparisons in this case. If the sole Class 2 comparison is a successful one, then Case 2 must hold for all Class 2 and 3 comparisons. Since  $r_l = 1$ ,  $r_l - 2 < 0$  and, by Lemma 5.15, no  $f$  values are defined using the set  $R_1(l)$ . Therefore, all  $f$  values for Class 2 and 3 comparisons in this case are defined as in Subcases 2a and 2b. Refer to these subcases. Note that, in this case,  $O$  consists of the pattern instances in  $A_1$  and  $O' = \{p_{r'}\}$ . From Lemma 5.13, all  $f$  values for Class 2 and 3 comparisons are to the left of the suffix  $x_{r-1}$  of  $p_1$  in this case.  $\square$

This concludes the definition of the  $f$  function.

## 6 Presuf Shifts with $|x'_1| \geq \frac{m}{2}$

This case can occur only for periodic patterns. So assume that  $p$  is periodic and has the form  $u_p v_p^{i_p}$ , where  $v_p$  and  $u_p$  are the core and head of  $p$ , respectively, and  $i_p \geq 2$ .

Recall that the lower bound of  $\frac{m+1}{2}$  on the distance between consecutive presuf shifts was crucial in deriving the comparison complexity for the case  $|x'_1| < \frac{m}{2}$ . This lower bound does not hold if  $|x'_1| \geq \frac{m}{2}$ . Consecutive presuf shifts can occur distance  $|v_p| \ll \frac{m+1}{2}$  apart. Even a single mismatch per presuf shift leads to a large comparison complexity. Since the problem in this case lies only in the frequency of occurrence of presuf shifts, we use the same basic algorithm changing only the presuf shift handler. The new presuf shift handler ensures that either two consecutive presuf shifts are at least distance  $\frac{m+1}{2}$  apart or no mismatch occurs between consecutive presuf shifts. In fact, we show the following stronger claim about the performance of the presuf shift handler. A presuf shift has overhead 0 if the next presuf shift occurs distance less than  $\frac{m+1}{2}$  ahead, overhead 1 if the next presuf shift occurs distance less than  $\frac{3(m+1)}{4}$  ahead, and overhead at most 2 otherwise. A comparison complexity of  $n(1 + \frac{8}{3(m+1)})$  comparisons follows.

### 6.1 The Presuf Shift Handler for $|x'_1| \geq \frac{m}{2}$

Before describing the presuf shift handler, we recall some definitions and assumptions made in Section 4. Let  $t_A$  refer to the portion of the text with which the prefix  $x'_1$  of  $p$  is aligned following the shift. We assume that prefix  $x'_1$  of  $p$  matches  $t_A$  following a presuf shift and that the variable  $t_{last}$  has been appropriately set to prevent this assumption from leading to an incorrect inference. Let  $t_a$  refer to the rightmost character in  $t_A$ .

As for the case  $|x'_1| < \frac{m}{2}$ , the presuf shift handler considers all presuf pattern instances, i.e., those pattern instance in which a prefix (possibly null) of  $p$  matches some suffix of  $t_A$ . In a presuf pattern instance, the prefix matching a suffix of  $t_A$  is a presuf of  $p$  and is called the presuf corresponding to this presuf pattern instance. Presuf pattern instances are of two types.

The first type consists of those presuf pattern instances whose corresponding presufs have the form  $u_p v_p^l$ ,  $1 \leq l \leq i_p - 1$ . The second type consists of those presuf pattern instances whose corresponding presufs are less than  $|v_p|$  in length. We identify a presuf pattern instance  $p'_\alpha$  of the second type as follows. If  $|u_p| > 0$  then  $p'_\alpha$  is the presuf pattern instance corresponding to the presuf  $u_p$ . If  $|u_p| = 0$  then  $p'_\alpha$  is the presuf pattern instance corresponding to the null presuf, i.e.,  $|p'_\alpha$  is to the immediate right of  $t_a$ . The following observation enables us to work with only presuf pattern instances of the second type while making comparisons within a text window  $\gamma$  of length  $|v_p|$  to the right of  $t_a$ .

**Lemma 6.1** *A presuf pattern instance of the first type matches all text characters in the window  $\gamma$  if and only if  $p'_\alpha$  matches all characters in that window.*

**Proof:** The portion of  $p'_\alpha$  which overlaps  $\gamma$  is identical to  $v_p$ , as is the corresponding portion of any presuf pattern instance of the first type.  $\square$

If  $p'_\alpha$  is eliminated by comparisons in  $\gamma$  then so are all presuf pattern instances of the first type. This forces the next presuf shift to occur at least distance  $m - |v_p| \geq \frac{m}{2} + 1$  to the right. If  $p'_\alpha$  is not eliminated by comparisons in  $\gamma$  then presuf pattern instances of the first type also survive and therefore, the next presuf shift can occur as little as distance  $|v_p|$  to the right. In this case, it is important to ensure that no mismatches are made in  $\gamma$ .

The presuf shift handler has five steps and works broadly as follows. As in the presuf shift handler of Section 4.2, the first two steps identify a presuf pattern instance  $p'_e$  with the following property: all presuf pattern instances that survive the first two steps are presuf overlaps of  $p'_e$ . This is accomplished by making comparisons in a manner similar to the presuf shift handler of Section 4.2, but with a single difference. This difference is aimed at ensuring that the first mismatch eliminates  $p'_\alpha$ . Steps 3, 4 and 5 are, however, identical to the corresponding steps of the earlier presuf shift handler.

Steps 1 and 2 proceed as follows to determine  $p'_e$ . They consider only presuf pattern instances of the second type and eliminate all but one of these. The survivor determines  $p'_e$ , i.e., if the survivor is  $p'_\alpha$  then  $p'_e$  is the leftmost presuf pattern instance and otherwise,  $p'_e$  is the survivor itself. We show that in order to eliminate among presuf pattern instances of the second kind it suffices to consider suitable prefixes of these presuf pattern instances. We need the following definitions in order to describe Steps 1 and 2 in detail. Consider the leftmost presuf pattern instance of the second type and let  $\beta$  be the length of the corresponding presuf. Suppose there are  $k'' + 1$  presuf pattern instances of the second type. We define  $p''_1, \dots, p''_{k''}, p''_{k''+1}$  such that  $p''_j$ ,  $1 \leq j \leq k'' + 1$ , is the prefix of length  $m'' = \beta + |v_p|$  of the  $j$ th leftmost presuf pattern instance of the second type. Let  $x''_j$ ,  $1 \leq j \leq k'' + 1$ , be the presuf corresponding to the  $j$ th leftmost presuf pattern instance of the second type. We call  $x''_j$  the presuf corresponding to  $p''_j$ . Let  $p''_\alpha$  refer to the length  $m''$  prefix of  $p'_\alpha$  and  $p''$  refer to the length  $m''$  prefix of  $p$ . The following lemma shows that in order to eliminate all but one of the presuf pattern instances of the second kind, it suffices to consider just  $p''_1, \dots, p''_{k''}, p''_{k''+1}$ .

**Lemma 6.2** *At most one of  $p''_1, \dots, p''_{k''}, p''_{k''+1}$  matches  $\gamma$ .*

**Proof:** Let  $x$  and  $y$  be the portions overlapping  $\gamma$  in some two of  $p''_1, \dots, p''_{k''+1}$ . Then  $x$  and  $y$  are different cyclic shifts of  $v_p$ . If  $x = y$  then  $v_p$  is cyclic, a contradiction.  $\square$

Let  $V'' = \{p''_1, \dots, p''_{k''+1}\}$ . Note that Lemmas 4.3–4.7 continue to hold if  $p_j$ ,  $x_j$ ,  $V$ ,  $p$  and  $m$  are replaced by  $p''_j$ ,  $x''_j$ ,  $V''$ ,  $p''$  and  $m''$ , respectively, for  $1 \leq j \leq k'' + 1$ . Henceforth, these

substitutions are implicit in all references to these lemmas. The elements in  $V''$  are divided into groups  $A''_1, \dots, A''_{g''}$ , in accordance with Lemma 4.3.

**Remark.** The presuf shift handler being described does not work for patterns for which  $|x''_{k''}| = 1$  and  $g'' = 1$ . Presuf shifts for these exception patterns are handled separately in Section 6.5.

With this background, we describe the 5 steps of the presuf shift handler.

**Step 1.** The characters in  $p''_1, \dots, p''_{k''}$  aligned with  $p''_1[m'']$ , the rightmost character in  $p''_1$ , are identical. If the character in  $p''_{k''+1}$  aligned with  $p''_1[m'']$  is also identical to it, then  $p''_1[m'']$  is compared with the aligned text character. A mismatch eliminates all of  $p''_1, \dots, p''_{k''}, p''_{k''+1}$  and the basic algorithm is restarted with  $|p$  placed immediately to the right of  $|p_{k''+1}$ . A match leads to Step 2.

**Step 2.** All but one of  $p''_1, \dots, p''_{k''+1}$  are eliminated in this step by making up to  $k''$  comparisons, at most 2 of which are unsuccessful. Further,  $p''_\alpha$  is eliminated by the first unsuccessful comparison. As in Step 2 of Section 4.2, there are two phases.

Phase 1 is identical to Phase 1 of Section 4.2, i.e., at every step the rightmost character  $c$  in  $p''_1$  having the following property is compared with the aligned text character: the character aligned with  $c$  in at least one of the surviving elements in  $V''$  is different from  $c$ . By Lemma 4.6, the outcome of Phase 1 is a half-done set  $O$ .

**Lemma 6.3**  $p''_\alpha \in O$  if and only if all the comparisons in Phase 1 are successful.

**Proof:** All comparisons in Phase 1 are made in the suffix  $x''_1$  of  $p_1$  because, by Lemma 4.5, successful comparisons in that suffix leave a half-done set uneliminated.  $x''_1$  is a presuf of  $p''_1$  and therefore a suffix of  $v_p$ . By the manner in which  $p''_\alpha$  is defined, the portion of  $p''_\alpha$  that overlaps  $\gamma$  is identical to the string  $v_p$ . Therefore, a mismatch in Phase 1 eliminates both  $p''_1$  and  $p''_\alpha$  while a match in Phase 1 eliminates neither. The lemma follows.  $\square$

If Phase 1 ends with a mismatch or if  $p''_\alpha = p''_1$  then Phase 2 is identical to Phase 2 of Section 4.2, i.e., all but one of the elements in  $O$  are eliminated by making comparisons according to a right to left sequence. Note that in both cases, the first mismatch eliminates  $p''_\alpha$ . Otherwise, if  $p''_\alpha \neq p''_1$  and all comparisons in Phase 1 are successful then we modify Phase 2 as follows so as to ensure that the first mismatch eliminates  $p''_\alpha$ .

Phase 2 proceeds exactly as Phase 2 of Step 2 in Section 4.2 until  $p''_\alpha$  becomes the rightmost element in  $O$ . Any mismatch in this process terminates Phase 2 and eliminates  $p''_\alpha$  and all elements in  $O$  to the left of  $p''_\alpha$ . If no mismatch occurs in this process then let the surviving elements in  $O$  be  $\{p''_{h_1}, \dots, p''_{h_e}\}$ , where  $p''_{h_1} = p''_1$  and  $p''_{h_e} = p''_\alpha$ . These elements are eliminated using a left to right sequence of comparisons instead of the right to left sequence used in Step 2 of Section 4.2. This left to right sequence ensures that a mismatch eliminates  $p''_\alpha$ . Let  $d_e$  be the leftmost character in  $p''_{h_e}$  such that  $p''_{h_e}$  differs from the aligned character in  $p''_{h_{e-1}}$ . By Lemma 6.2,  $d_e$  is aligned with or to the left of  $p''_1[m'']$  and to the right of  $t_a$ . If  $e = 2$  then a comparison at  $d_e$  terminates Phase 2 with a mismatch eliminating  $p''_{h_e}$  and a match eliminating  $p''_{h_1}$ . Suppose  $e > 2$ . Then  $x''_{h_1}$  is periodic with core, say,  $v$ . Let  $d_j, 2 \leq j \leq e - 1$ , be the character in  $p''_{h_e}$  which is distance  $(e - j)|v|$  to the left of  $d_e$ . The characters  $d_2, \dots, d_e$  are compared with the aligned text characters in sequence until either a mismatch occurs or the sequence is exhausted. The following lemma shows that at most one element of  $O$  survives these comparisons.

**Lemma 6.4** A mismatch at  $d_j$  leaves only  $p_{i_{j-1}}$  uneliminated. A match at  $d_j$  eliminates  $p_{i_{j-1}}$ .



**Proof:** If  $e = 2$  then the lemma is clearly true. Suppose  $e > 2$ . From the definition of  $d_e$ , it follows that the prefix of  $p''_{h_e}$  ending at  $d_e$  is periodic with core of size  $|v|$  while the prefix of  $p''_{h_{e-1}}$  is not; therefore,  $d_2 = d_3 = \dots = d_e \neq d_{e+1}$ , where  $d_{e+1}$  is the character which is distance  $|v|$  to the right of  $d_e$ . It follows that the characters in  $p''_{h_j}, \dots, p''_{h_e}$  aligned with  $d_j$  are identical to each other but different from the character in  $p''_{h_{j-1}}$  aligned with  $d_j$ . Therefore, a match at  $d_j$  eliminates  $p''_{h_{j-1}}$ . A mismatch at  $d_j$  eliminates  $p''_{h_j}, \dots, p''_{h_e}$  and the preceding successful comparisons at  $d_2, \dots, d_{j-1}$  eliminate  $p''_{h_1}, \dots, p''_{h_{j-2}}$ . The lemma follows.  $\square$

This completes Step 2. At most  $k''$  comparisons are made in this step, at most 2 of which result in mismatches. Further,  $p''_\alpha$  survives only if there are no mismatches. The sequence of comparisons made in Step 2 can be represented by a tree  $ET''$ , akin to the tree  $ET$  of Section 4.2. The only difference between  $ET''$  and  $ET$  is that the sequence corresponding to a portion of Phase 2 may now be a left to right sequence if Phase 1 does not end in a mismatch and  $p''_\alpha \neq p''_1$ . We conclude Step 2 with the following lemma.

**Lemma 6.5** *All but at most one of  $p''_1, \dots, p''_{k''}, p''_{k''+1}$  can be eliminated by making up to  $k''$  comparisons using the  $O(k'')$  sized binary comparison tree  $ET''$ . At most two of these comparisons result in mismatches. If  $p''_\alpha$  survives, then no comparisons result in mismatches. Moreover, the sequence of comparisons made by the elimination strategy consists of two sequences: a right to left sequence followed by either another right to left sequence or a left to right sequence.*

We describe Steps 3, 4 and 5 next. Let  $p''_e$  be the only element of  $V''$  to survive Steps 1 and 2. If  $p''_e = p''_\alpha$  then define  $p'_e$  to be the leftmost presuf pattern instance. If  $p''_e \neq p''_\alpha$  then let  $p'_e$  be the presuf pattern instance of which  $p''_e$  is a prefix, i.e.,  $p'_e$  and  $p''_e$  have their left ends aligned. Clearly,  $p'_e$  is the leftmost presuf pattern instance to survive Steps 1 and 2. Let  $Q$  denote the set of pattern instances which overlap  $p'_e$  and have their left end to the right of  $|p''_{k''+1}|$ . In the elimination process, some elements of  $Q$  may also have been eliminated from being potential matches. They need not be reconsidered. To this end, a subset  $Q_x$  of  $Q$  consisting of pattern instances consistent with comparisons in Steps 1 and 2 is associated with each terminal node  $x$  in  $ET''$ . The maintenance of  $Q_x$  is similar to the description in section 4.5 and is described in Section 6.4. Suppose that the elimination process terminates at terminal node  $x$ . Let  $Q' = \{p'_e\} \cup Q_x$ . Steps 3, 4, and 5 are now identical to the corresponding steps in Section 4.2.

## 6.2 Comparison Complexity

In order to determine the comparison complexity, we need to define a transfer function  $f''$  akin to the transfer function  $f$  defined in Section 5. We state the following lemma describing the properties of  $f''$ . The proof of this lemma is deferred to Section 6.3.

**Lemma 6.6** *Let  $C$  be the set of text characters involved in comparisons in Steps 1 and 2 of the presuf shift handler of Section 6.1. For each character  $t_c \in C$ , with at most two exceptions, there exists a text character  $f''(t_c) = t_d$  satisfying the following properties.*

1.  $t_d$  is to the right of  $|p''_{k''+1}|$ .
2.  $t_d$  either coincides with  $t_c$  or lies to the left of  $t_c$ .

3. The pattern instance whose left end is aligned with  $t_d$  is eliminated as a result of comparisons in Steps 1 and 2 of the presuf shift handler.
4. For every distinct  $t_{c_1}, t_{c_2} \in C$ ,  $f(t_{c_1}) \neq f(t_{c_2})$ .

Furthermore, mismatches, if any, are always included among the exceptions.

The following lemma determines the comparison complexity of the algorithm.

**Lemma 6.7** *If  $p$  is not a special case pattern then the comparison complexity of the algorithm is bounded by  $n(1 + \frac{8}{3(m+1)})$ .*

**Proof:** A presuf shift occurs either with  $|x'_1| < \frac{m}{2}$  or with  $|x'_1| \geq \frac{m}{2}$ . In the former case, it was shown in Lemma 4.12 that a presuf shift can have overhead at most two and that an overhead of two implies that the next presuf shift occurs at least distance  $\frac{3(m+1)}{4}$  to the right. Further,  $\frac{m+1}{2}$  is a lower bound on the distance between two consecutive presuf shifts in this case. We show similar properties for presuf shifts with  $|x'_1| \geq \frac{m}{2}$ . Specifically, we show that a presuf shift can have overhead at most two. Further, we show that an overhead of one forces the next presuf shift to occur at least distance  $\frac{m+1}{2}$  to the right and an overhead of two forces the next presuf shift to occur at least distance  $\frac{3(m+1)}{4}$  to the right. We show the above by giving a charging scheme for the presuf shift handler of Section 6.1. The comparison complexity of the algorithm now follows.

**Charging Scheme.** As in Lemma 4.12, the run of the algorithm is divided into phases; a phase can be of one of four types. The ranges of the text characters charged in each phase type remain exactly the same as in Lemma 4.12. The charging scheme for Type 1 and Type 2 phases also remains exactly the same. Only the charging scheme for Type 3 and Type 4 phases is modified in accordance with the presuf shift handler of Section 6.1.

We consider a single phase, which could be a Type 3 or a Type 4 phase. We assume that this phase begins with a presuf shift with  $|x'_1| \geq \frac{m}{2}$ . Let  $q_1$  and  $q_2$  refer to the leftmost surviving pattern instances at the beginning and end of that phase, respectively. Note that  $q_1$  is a presuf overlap of the pattern instance  $q'$ , the leftmost uneliminated pattern instance prior to the presuf shift which initiated this phase. Specifically, the prefix  $x'_1$  of  $q_1$  is aligned with the suffix  $x'_1$  of  $q'$  (Recall that on a presuf shift, we assume that the prefix  $x'_1$  of  $q_1$  matches the text). Recall that  $t_a$  is the text character aligned with  $q'$ .

Consider the comparisons made by the current use of the presuf shift handler of Section 6.1. If a mismatch occurs in Step 1, the current phase ends immediately and the basic algorithm is resumed. The presuf shift in this case has overhead 1 and the next presuf shift occurs at least distance  $m + 1$  to the right. Next, suppose that the comparison in Step 1 is successful. Let  $p'_e$  be the presuf pattern instance to survive the elimination using tree  $ET''$  in Step 2. After the presuf shift handler finishes, one of three scenarios ensues. We consider each in turn.

1. All pattern instances overlapping  $p'_e$  are eliminated apart from its presuf overlaps, and  $p'_e$  or at least a suffix of  $p'_e$  is matched. This is a Type 4 phase. We consider two cases, depending upon whether  $p'_e$  is a presuf pattern instance of the first or the second type.

First, suppose  $p'_e$  is of the first type, i.e., it is the leftmost presuf pattern instance. Then no mismatches are made in Steps 1, 2, 3, 4 or 5. All comparisons made by the presuf shift handler are charged to the text characters compared. The bit vector  $BV$  ensures that each of these

comparisons involves a different text character. Thus each text character which lies to the right of  $t_a$  and is aligned with or to the left of  $p'_e$  is charged at most once. In this case, the overhead of this presuf shift is 0.

Next, suppose  $p'_e$  is of the second type. Then  $p''_\alpha$  is eliminated in Step 2. All comparisons in Steps 1, 3, 4 and 5 and all but at most two comparisons in Step 2 are successful. Each successful comparison is charged to the text character compared. The bit vector  $BV$  ensures that each of these comparisons involves a different text character. Thus each text character which lies to the right of  $t_a$  and is aligned with or to the left of  $p'_e$  is charged at most once. At most two comparisons in Step 2 are unsuccessful, so this shift has overhead at most two. If there are two mismatches in Step 2 then we claim that  $p''_1$  is eliminated; in addition, if  $x''_1$  is periodic, with core  $v$  and head  $u$  say, then all elements in  $V''$  whose associated presufs have the form  $uv^o$ ,  $o \geq 1$ , are also eliminated. (this can be shown in a manner similar to the corresponding proof in Lemma 4.12). Let  $p''_e$  be the  $m''$  length prefix of  $p'_e$  and let  $x''_e$  be the presuf associated with  $p''_e$ . From the above, it follows that  $x''_1 = x''_e z x''_e$ , for some non-empty string  $z$ . As  $|x''_1| < |v_p|$ ,  $p''_e = x''_1 w x''_1$  for some non-empty string  $w$ . Therefore  $|x''_e| \leq \frac{m''-3}{4} \leq \frac{m-3}{4}$ . This guarantees that the next presuf shift occurs at least distance  $\frac{3(m+1)}{4}$  to the right. If there is just one mismatch in Step 2 then, since  $|x''_1| < |v_p| \leq \frac{m}{2}$ , the next presuf shift occurs at least distance  $\frac{m+1}{2}$  to the right.

2.  $p'_e$  is eliminated. In addition, there is some pattern instance  $q_c$  overlapping  $p'_e$ , such that all pattern instances overlapping  $q_c$  are eliminated apart from its presuf overlaps; further,  $q_c$  or at least a suffix of  $q_c$  is matched. This is also a Type 4 phase.

Each comparison in Steps 1 and 2 with a text character to the left of  $|q_c$  for which function  $f''$  is defined is charged to the text character specified by the function  $f''$ , called its  $f''$  value;  $f''$  values are distinct by definition. Comparisons in Step 3 fall into one of three categories:

1. Comparisons which eliminate patterns instances whose left ends lie to the right of  $|p''_{k''+1}|$  and to the left of  $|q_c|$ .
2. Comparisons which eliminate pattern instances whose left ends lie to the right of  $|q_c|$ .
3. The comparison which eliminates  $p'_e$ .

Each comparison in the first category is charged to the text character aligned with the left end of the pattern instance eliminated. By the definition of the function  $f''$ , these text characters do not occur in the range of  $f''$  values. Comparisons in the second category, along with the comparisons made in Steps 4 and 5 and those successful comparisons in Steps 1 and 2 that involve text characters overlapping  $q_c$ , are charged to the text characters compared.  $BV$  ensures that each of these comparisons involves a distinct text character. Thus each text character which lies to the right of  $|p''_{k''+1}|$  and is aligned with or to the left of  $|q_c|$  is charged at most once. The comparison that eliminates  $p'_e$  is charged to the text character aligned with  $|p''_{k''+1}|$ . Since all  $f''$  values lie to the right of  $|p''_{k''+1}|$  and all pattern instances eliminated by comparisons in the first category have left ends to the right of  $|p''_{k''+1}|$ , this text character is charged exactly once. The two comparisons in Step 2 lacking  $f''$  values constitute the overhead of this presuf shift. Since  $p'_e$  is eliminated, the next presuf shift occurs at least distance  $m+1$  to the right of the current presuf shift.

3.  $p'_e$  is eliminated as are all pattern instances overlapping  $p'_e$ . This is a Type 3 phase.

Let  $q_d$  denote the leftmost surviving pattern instance. All comparisons in Steps 1 and 2 for which function  $f''$  is defined are charged to their  $f''$  values.  $f''$  values are distinct by definition.

Excluding the comparison which eliminates  $p'_e$ , each comparison in Steps 3 and 4 eliminates some pattern instance whose left end lies to the right of  $|p''_{k''+1}$  and to the left of  $|q_d$ . Each such comparison is charged to the text character aligned with the left end of the pattern instance eliminated. These text characters cannot occur in the range of the function  $f''$  and hence are charged only once. Thus each text character which lies to the right of  $|p''_{k''+1}$  and to the left of  $|q_d$  is charged at most once. The comparison that eliminates  $p'_e$  is charged to the text character aligned with  $|p''_{k''+1}$ . The two comparisons in Step 2 lacking  $f''$  values constitute the overhead of this presuf shift. Since  $p'_e$  is eliminated, the next presuf shift occurs at least distance  $m + 1$  to the right of the current presuf shift.  $\square$

### 6.3 The Transfer Function $f''$

In this section, we prove Lemma 6.6. The definition of the function  $f''$  is similar to that of the function  $f$  in Section 5. This is hardly surprising as the elimination procedure  $ET''$  is similar to the elimination process  $ET$ , the only difference between the two being that the former switches to a left to right comparison sequence in some cases.

First, note that each of the definitions and lemmas in Section 5.1 continue to hold if  $p''_j, x''_j, V'', p'', A'', g'', k''$  and  $m''$  replace  $p_j, x_j, V, p, A, g, k$  and  $m$ , respectively, for  $1 \leq j \leq k'' + 1$ .

Since patterns with  $g'' = 1$  and  $|x''_k| = 1$  are special case patterns, we assume that  $g'' > 1$  if  $|x''_k| = 1$ . If  $p''_1[m'']$  does not match the text, then Steps 1 and 2 of the presuf shift handler make at most one comparison. Therefore, we also assume that  $p''_1[m]$  matches the text. Let  $p''_l$  be the rightmost element in  $A''_1$ . Let  $p''_r$  be the rightmost element in  $V''$  outside  $A''_{g''}$ , if such a pattern instance exists.

As in Section 5.2, we split the sequence  $C'$  of comparisons made in Steps 1 and 2 of the presuf shift handler into three classes as follows.

1. Class 1 consists of the comparison in Step 1. In addition, if  $|x''_k| = 1$ , then Class 1 contains the comparisons which comprise the smallest prefix of  $C'$  having the following property: either the last comparison in that prefix is unsuccessful or following that comparison, exactly one pattern instance in  $A''_{g''}$  survives.
2. Class 2 consists of the comparisons in  $C'$  which follow all Class 1 comparisons and are made in the suffix  $h(x''_l)$  of  $p''_l$ .
3. Class 3 consists of comparisons in  $C'$  which follow all Class 2 comparisons.

$f''$  values are defined by considering 3 cases.

**Case 1.** Suppose Phase 1 of Step 2 terminates with a mismatch or  $p''_\alpha = p''_1$ . Then  $ET''$  eliminates among elements in  $V''$  exactly as  $ET$  eliminates among the elements of  $V$ . So  $f''$  values for comparisons are defined exactly as in Section 5.2 with  $p''_j, x''_j, V'', p'', A'', g'', k''$  and  $m''$  replacing  $p_j, x_j, V, p, A, g, k$  and  $m$ , respectively, for  $1 \leq j \leq k'' + 1$ .

**Case 2.** Suppose  $p''_\alpha = p''_2$  or the half-done set left uneliminated by Phase 1 has at most two elements. The only difference between the way  $ET''$  eliminates among the elements in  $V''$  and  $ET$  eliminates among elements in  $V$  is in the last comparison of Step 2. Note that in Section 5.2, the last comparison in Step 2 is not given an  $f$  value. Therefore,  $f''$  values for comparisons in this case are again defined exactly as in Section 5.2 with  $p''_j, x''_j, V'', p'', A'', g'', k''$  and  $m''$  replacing  $p_j, x_j, V, p, A, g, k$  and  $m$ , respectively, for  $1 \leq j \leq k'' + 1$ .

**Case 3.** Suppose all comparisons in Phase 1 are successful,  $p''_\alpha \neq p''_1, p''_2$  and the half-done set which survives Phase 1 has at least three elements. The only difference between the way  $ET''$  eliminates among the elements in  $V''$  and  $ET$  eliminates among elements in  $V$  is in the portion of Phase 2 that makes comparisons according to a left to right sequence. As we will show in Lemma 6.11, this left to right sequence involves only text characters to the left of the the suffix  $x''_1$  of  $p''_1$ . Consequently, Class 1 and Class 2 comparisons are not affected by this sequence.

$f''$  values for comparisons in Class 1 are defined exactly as in Section 5.2 with  $p''_j, x''_j, V'', p'', A'', g'', k''$  and  $m''$  replacing  $p_j, x_j, V, p, A, g, k$  and  $m$ , respectively, for  $1 \leq j \leq k'' + 1$ . Consider Class 2 comparisons next. At most one element of  $V''$  will survive a mismatch in Class 2, if any, because Phase 1 has no mismatches. Therefore, if a mismatch occurs in Class 2 then Class 3 is empty and  $f''$  values for Class 2 comparisons are defined exactly as in Section 5.2 with the appropriate substitutions mentioned above. Otherwise, if all comparisons in Class 2 are successful then  $f''$  values for all but some  $s, s \leq 2$ , of the comparisons in Class 2 are defined in the same manner. It remains to define  $f''$  values for Class 3 comparisons and  $s$  Class 2 comparisons when all comparisons in Class 2 are successful. This involves modifying only Case 2 of the definition of  $f$  values for Class 2 and Class 3 comparisons in Section 5.2. We define  $f''$  values for all but two of these comparisons. The range of these  $f''$  values is the same as the range of the  $f$  values defined for this subcase, i.e., to the left of the suffix  $x''_1$  of  $p''_1$  and to the right of  $|p''_{k+1}$ .

Following Class 1 and 2 comparisons, at most  $\min\{r_l, 2\} - s$  of the elements of  $V''$  to the right of  $p''_1$  survive along with the elements in  $A''_1$ . Let  $O'$  refer to the set of  $\min\{r_l, 2\}$  elements in  $V''$  which includes elements which survive comparisons in  $h(x''_1)$  and elements which are eliminated by one of the  $s$  Class 2 comparisons under consideration. Let  $O$  refer to the largest half-done set consisting of elements in  $A''_1$  and  $O'$ . Redefine  $O'$  by removing pattern instances in it which are also in  $O$ . Considering comparisons which eliminate pattern instances in  $O$  and  $O'$  is equivalent to considering Class 3 comparisons plus  $s$  of the Class 2 comparisons. Let  $O = \{p''_{h_1}, \dots, p''_{h_e}\}$ . Let  $v$  and  $u$  be the core and head, respectively, of  $x''_{h_1}$  and let  $v = u'u$ .  $|v| > 1$  because either  $|x''_k| > 1$  or  $|x''_k| = 1$  and  $g' > 1$ . By Lemma 5.10,  $v$  contains a misfit character. If  $l = 1$ , then the number of comparisons in Class 3 plus  $s$  is at most  $2 - s + s = 2$ . In this case, we do not define an  $f''$  value for the comparisons in Class 3 and the  $s$  comparisons in Class 2. So suppose that  $l > 1$ .

The comparisons given by tree  $ET''$  in this case form two sequences; the first sequence which includes Phase 1 and part of Phase 2 is a right to left sequence and the second sequence is a left to right sequence. The following lemmas show some properties which are necessary for defining  $f''$ .

**Lemma 6.8** *The portion of  $p''_\alpha$  which overlaps the suffix  $x''_i, 1 \leq i \leq \alpha$ , of  $p''_1$ , matches  $x''_i$ .*

**Proof:**  $x''_i$  is a suffix of  $v_p$ . The length  $|v_p|$  substring of  $p''_\alpha$  which is to the immediate right of  $t_\alpha$  is identical to  $v_p$ .  $\square$

**Lemma 6.9**  *$p''_\alpha \in O$  and  $|O| \geq 3$ .*

**Proof:** Since all comparisons in Phase 1 are successful,  $p''_1$  survives Phase 1. By Lemma 6.3,  $p''_\alpha$  survives too. If  $p''_\alpha \notin O$  then  $p''_1, p''_\alpha$  do not form a half-done set with any other element in  $V''$ . Therefore, the cardinality of the half-done set which survives Phase 1 would be at most two, which is a contradiction. Thus  $p''_\alpha \in O$ .  $p''_2$  must form a half-done set along with  $p''_1$  and

$p''_\alpha$ , otherwise no other element in  $V''$  forms a half-done set with  $p''_1$  and  $p''_\alpha$  and consequently, at most two elements in  $V''$  would survive the successful Phase 1 comparisons. By Lemma 6.8,  $p''_1$  and  $p''_\alpha$  survive successful comparisons in  $h(x''_1)$ , and then by Lemma 5.3,  $p''_2$  also survives these comparisons. Therefore,  $p''_2 \in O$  also. Since  $p''_\alpha \neq p''_1, p''_2$ , the lemma follows.  $\square$

**Corollary 6.10** *The half-done set which survives Phase 1 must be a subset of  $O$ .*

**Proof:** Both  $p''_1$  and  $p''_\alpha$  survive successful comparisons in Phase 1 and both are elements of  $O$ . The only elements in  $V''$  which can form a half-done set with  $p''_1$  and  $p''_\alpha$  are those in  $O$ .  $\square$

**Lemma 6.11** *The leftmost character compared by  $ET''$  in the first (right to left) sequence is at least distance  $|v|$  to the right of the rightmost character compared in the second (left to right) sequence. The rightmost character compared in the latter sequence is to the left of the suffix  $x''_1$  of  $p''_1$ .*

**Proof:** Let  $d''$  be the rightmost position in  $p''_\alpha$  such that  $p''_\alpha[d'']$  is aligned with some character in  $p''_1$  and  $p''_\alpha[d''] \neq p''_\alpha[d'' + |v|]$ . Such an index exists by Lemma 4.7. All comparisons in the second sequence are aligned with or to the left of  $p''_\alpha[d'']$ . All characters in the first sequence compared in Phase 2 are aligned with or to the right of  $p''_\alpha[d'' + |v|]$ . All characters compared in Phase 1 involve characters in the suffix  $x''_1$  of  $p''_1$ . By Lemma 6.8,  $p''_\alpha[d'']$  is to the left of the suffix  $x''_1$  of  $p''_1$ . The lemma follows.  $\square$

**Corollary 6.12** *Successful comparisons which eliminate elements of  $O$  are made at least distance  $|v|$  apart.*

**Lemma 6.13** *The portion of  $p''_{h_e}$  that overlaps the suffix  $x''_{h_{e-2}}$  of  $p''_1$  matches that suffix.*

**Proof:** Since  $p''_\alpha \in O$  and  $p''_\alpha \neq p''_1, p''_2$ , it follows from Lemma 6.8 that  $(uu')^2$  is a suffix of  $p''[1 \dots m'' - |x''_1|]$ . Therefore, the portion of  $p''_{h_e}$  that overlaps the suffix  $x''_{h_{e-2}}$  of  $p''_1$  matches that suffix.  $\square$

**Corollary 6.14** *All successful comparisons which eliminate an element of  $O$  are made to the left of the suffix  $x''_{h_{e-2}}$  of  $p''_1$ .*

We now define the  $f''$  function for this case.

First, consider comparisons which eliminate elements of  $O'$ . From Corollary 6.10, it follows that all elements of  $O'$  must be eliminated by Phase 1 comparisons. These comparisons have to be successful because all comparisons in Phase 1 are successful. If  $|O'| = 2$  then, by Lemma 4.5, the first such comparison is made in the suffix  $x''_{h_e}$  of  $p''_1$ . If  $|O'| = 2$  or  $|O'| = 1$ , then, by Lemma 6.13 the portion of  $p''_{h_e}$  which overlaps the the suffix  $x''_{h_{e-1}}$  of  $p''_1$  matches that suffix and therefore, by Lemma 4.5, the last comparison which eliminates an element of  $O'$  is made in the suffix  $x''_{h_{e-1}}$  of  $p''_1$ . Consider the text characters  $t_c$  and  $t'_c$  with which  $|p''$  is aligned when the rightmost misfit character in the prefixes  $x''_{h_{e-1}}$  and  $x''_{h_{e-2}}$ , respectively, of  $p''$  are aligned with  $t_b$ . Since  $v$  is a suffix of  $x''_{h_{e-1}}$  and  $x''_{h_{e-2}}$  and since  $v$  contains a misfit character,  $t_c$  is aligned with the suffix  $x''_{h_{e-1}}$  of  $p''_1$  and to the left of the suffix  $x''_{h_e}$  of  $p''_1$  while  $t'_c$  is aligned with the suffix  $x''_{h_{e-2}}$  of  $p''_1$  and to the left of the suffix  $x''_{h_{e-1}}$  of  $p''_1$ . If  $|O'| = 2$  then  $f''$  is defined to map the text characters involved in comparisons which eliminate elements of  $O'$  to the text characters  $t_c$  and  $t'_c$ . If  $|O'| = 1$  then  $f''$  is defined to map the text character involved in the

comparison which eliminates the only element of  $O'$  to the text character  $t'_c$ . A simple case analysis ( $p''_l = p''_{h_e}, p''_{h_{e-1}}, p''_{h_{e-2}}$ ) shows that these  $f''$  values are to the left of  $p''_l$ , as claimed. The two  $f''$  values are clearly distinct and to the left of their respective text characters. Further, they are aligned with the suffix  $x''_1$  of  $p''_1$ . Since  $|x''_1| < \frac{m''}{2}$ , these  $f''$  values are to the right of  $|p''_{k''+1}$ .

Next, consider comparisons which eliminate elements of  $O$ , excluding the leftmost and the last such comparison. The remaining comparisons must be successful.  $f$  is defined to map the text character  $t_c$  involved in such a comparison to the text character with which  $|p''$  is aligned when the leftmost character in  $p''$  which differs from  $t_c$  is aligned with  $t_c$ . Clearly,  $f''(t_c)$  is aligned with or to the left of  $t_c$ . Since  $uu'$  contains at least two characters,  $f''(t_c)$  is at most distance  $|v| - 1$  to the left of  $t_c$ . It follows from Corollary 6.12 that  $f''(t_c)$  is distinct from the  $f''$  values for all other text characters involved in successful comparisons which eliminate elements of  $O$ . By Corollary 6.14, these  $f''$  values are to the left of  $f''$  values for successful comparisons which eliminate element of  $O'$  and therefore, to the left of  $p''_l$ . Only the leftmost text character involved in a comparison which eliminates an element of  $O$  is within distance  $|v|$  of  $t_a$ ; the rest are at least distance  $|v| + 1$  to the right of  $t_a$ . Therefore, these  $f''$  values are to the right of  $|p''_{k''+1}$ .

This concludes the definition of the transfer function  $f''$ .

## 6.4 Data Structure Details

It remains to describe the maintenance of the sets  $Q_x$  for each terminal node  $x$  of tree  $ET''$ . These sets can be maintained exactly as described in Section 4.5 but for the following difference: the sequence of comparisons corresponding to Phase 2 in Step 2 of the elimination strategy using  $ET''$  is a left to right sequence if all comparisons in Phase 1 are successful.

As is Section 4.5, let  $l_1, \dots, l_h$  be the nodes, in the order of appearance, on the leftmost path from the root of  $ET''$ . Consider the largest  $i$  such that  $tc_{l_i}, \dots, tc_{l_{h-1}}$  (recall from Section 4.5 that  $tc_x$  is the text character compared at node  $x$  of  $ET''$ ) is a left to right sequence. For all terminal nodes in  $ET''$  which are not in the subtree rooted at  $l_i$ , the data structure is maintained exactly as in Section 4.5.  $Q_{l_h}$  can be stored explicitly. It remains to describe the data structure for terminal nodes in the right subtrees of  $l_i, \dots, l_{h-1}$ .

Note that if a mismatch occurs at  $tc_{l_j}$ ,  $i \leq j \leq h - 1$ , at most two elements in  $V''$  survive. Therefore, for each terminal node  $x$  in the right subtree of  $l_j$ , either  $p(x)$  or  $p(p(x))$  equals  $l_j$ , where  $p(x)$  is the parent of  $x$ . From the definition of the sets  $Q_x$  in Section 4.5, it follows that for terminal nodes  $x$  and  $y$  in the right subtree of  $l_j$ ,  $Q_x = Q_y$ . The following lemma is crucial.

**Lemma 6.15** *Let terminal node  $x_1$  be in the right subtree of  $l_{j_1}$  and terminal node  $x_2$  be in the right subtree of  $l_{j_2}$ ,  $i \leq j_1, j_2 \leq h - 1$ ,  $j_2 > j_1$ . If  $q \in Q_{x_1}$  and  $q \in Q_{x_2}$  then  $q$  occurs at all terminal nodes in the right subtrees of  $l_i, \dots, l_{j_1}$ .*

**Proof:** Clearly,  $q$  cannot overlap  $tc_{l_{j_1}}$ . Since  $tc_{l_i}, \dots, tc_{l_{h-1}}$  form a left to right sequence,  $q$  cannot overlap  $tc_i, \dots, tc_{j_1}$ . Further, since  $q$  occurs at some terminal node in the subtree rooted at  $l_i$ , characters in  $q$  which overlap  $tc_{l_1}, \dots, tc_{l_{i-1}}$  match the characters  $c_1, \dots, c_{h-1}$ , respectively. The lemma follows from the definition of the sets  $Q_x$ .  $\square$

**Corollary 6.16** *Suppose  $q$  occurs at some terminal node in the subtree  $T$  rooted at  $l_i$ . Further, suppose  $j$  is the largest number, if any, such that  $i \leq j \leq h - 1$  and  $q$  does not overlap  $tc_{l_j}$ . Then  $q$  occurs at all terminal nodes in the right subtrees of  $l_i, \dots, l_j$ .*

Corollary 6.16 immediately gives a linear space scheme for storing the sets  $Q_x$  for terminal nodes  $x$  in the subtree  $T$  rooted at  $l_i$ . Two sets  $Com_j$  and  $Spec_j$  are maintained at each node  $l_j$ ,  $i \leq j \leq h - 1$ . A pattern instance  $q$  is added to  $Com_j$  if it occurs at some terminal node in  $T$  and overlaps  $tc_{l_{j+1}}$  but not  $tc_{l_j}$ . A pattern instance  $q$  is added to  $Spec_j$  if it overlaps  $tc_{l_j}$  and occurs at a terminal node in the right subtree of  $l_j$ . Each  $q$  can be added to at most one  $Com$  set and one  $Spec$  set; thus, the total space used is linear.  $Q_x$  is readily seen to equal  $Com_j \cup Com_{j+1} \cup \dots \cup Com_{h-1} \cup Spec_j$ . Note that each pair of  $Com$  sets is disjoint and  $Com_k$  is disjoint from  $Spec_j$ , for each  $j \leq k \leq h - 1$ . In order to obtain  $Q_x$  as a sorted list it suffices to maintain each of the  $Com$  and  $Spec$  sets as ordered lists which are then appended together. Thus obtaining any particular  $Q_x$  takes  $O(m)$  time.  $Q_{l_h}$  is stored explicitly and hence can be obtained as a list in constant time.

## 6.5 Presuf Shift Handler for Special Case Patterns

We describe the presuf shift handler for patterns for which  $|x''_k| = 1$  and  $g'' = 1$ . This presuf shift handler leads to an overhead of at most 2 per presuf shift. We show that if a presuf shift has overhead 2, then the next presuf shift must occur distance at least  $\frac{3(m+1)}{4}$  to the right, and if a presuf shift has overhead 1, then the next presuf shift must occur distance at least  $\frac{m+1}{2}$  to the right. A comparison complexity of  $n(1 + \frac{8}{3(m+1)})$  follows.

Let  $b = x''_k$ .  $p$  contains at least two different characters. Therefore,  $v_p$  and  $p''$  both contain at least two different characters. Let  $p''[j]$  and  $p''[j']$  be, respectively, the leftmost and rightmost characters in  $p''$  which differ from  $b$ . Let  $t_c$  be the text character to the immediate right of  $t_a$ .

We consider two cases, namely,  $|x''_1| < \frac{|v_p|}{2}$  and  $|x''_1| \geq \frac{|v_p|}{2}$ . The former case has the advantage that if all presuf pattern instances of the first type (recall that presuf pattern instances were classified into two types in Section 6) are eliminated then the next presuf shift occurs distance at least  $\frac{3(m+1)}{4}$  to the right. The absence of this property in the latter case makes it more complicated.

**Case 1.**  $|x''_1| < \frac{|v_p|}{2}$ .

**Step 1.** Step 1 locates the leftmost non- $b$  text character  $t_d$  to the right of  $t_a$ . Following Step 1, either the basic algorithm is resumed or  $p'_e$ , the leftmost surviving pattern instance, is determined and Step 2 follows. This is done as follows. Text characters to the right of  $t_a$  and to the left of  $p''_\alpha[j]$  are compared from left to right with the character  $b$ . A mismatch in this process terminates Step 1. If no mismatch occurs then  $p''_\alpha[j]$  is compared with the aligned text character. A match terminates Step 1. In case of a mismatch, text characters aligned with or to the right of  $p''_\alpha[j]$  are compared from left to right with the character  $b$ . Step 1 then terminates when a mismatch occurs or when the right end of the text is reached.

One of the following situations holds now.

1.  $t_d$  is to the left of  $p''_1[j]$ .  $p''_1, \dots, p''_{k+1}$  are eliminated and the basic algorithm is resumed with the  $|p$  placed to the right of the text character that mismatched.
2.  $t_d$  is aligned with  $p''_i[j]$ ,  $i \neq \alpha$ .  $p'_e$  is the pattern instance whose left end is aligned with  $|p''_i$ .
3.  $t_d$  is aligned with  $p''_\alpha[j]$  and  $t_d = p''_\alpha[j]$ .  $p'_e$  is defined to be the leftmost presuf pattern instance.



4.  $t_d$  is aligned with  $p''_\alpha[j]$  but  $t_d \neq p''_\alpha[j]$ . The basic algorithm is resumed with  $|p$  immediately to the right of  $t_e$ .
5.  $t_d$  exists but does not satisfy any of the above cases.  $p'_e$  is the pattern instance such that  $p'_e[j]$  is aligned with  $t_d$ .
6.  $t_d$  does not exist. There are no further occurrences of the pattern in the text and the algorithm terminates.

**Steps 2 and 3.** Let  $q_c$  denote  $p'_e$ . Then Steps 2 and 3 are identical to the corresponding steps in the presuf shift handler for special case patterns described in Section 4.4.

Note that at most two mismatches are made in Step 1 and the first mismatch eliminates  $p''_\alpha$ .

**Lemma 6.17** *If  $p$  is a special case pattern and  $|x''_1| < \frac{|v_p|}{2}$  then the comparison complexity of the algorithm is  $n(1 + \frac{8}{3(m+1)})$ .*

**Proof:** We give charging strategies to show that a presuf shift can have overhead at most two. Further, we show that an overhead of one forces the next presuf shift to occur at least distance  $\frac{m+1}{2}$  to the right and an overhead of two forces the next presuf shift to occur at least distance  $\frac{3(m+1)}{4}$  to the right. The lemma follows.

As in Lemma 4.12, the run of the algorithm is divided into phases; a phase can be of one of four types. The range of text characters charged in each type of phase remains exactly the same as in Lemma 4.12. The charging scheme for Type 1 and Type 2 phases also remains exactly the same. Only the charging scheme for Type 3 and Type 4 phases is modified in accordance with the presuf shift handlers described above.

We consider a single phase, which could be a Type 3 or a Type 4 phase. We assume that this phase begins with a presuf shift with  $|x'_1| \geq \frac{m}{2}$ .

**The Charging Scheme.** Let  $q_c$  be the leftmost pattern instance which survives Step 1. Note that  $q_c$  is the leftmost presuf pattern instance if and only if no mismatches occur in Step 1. All successful comparisons in Step 1 are charged to the text characters compared. These text characters lie to the left of  $q_c[j]$  if  $q_c$  is not the leftmost presuf pattern instance and are aligned with or to the left of  $q_c[j]$ , otherwise. If unsuccessful comparisons occur in Step 1 then these comparisons constitute the overhead of this shift. Otherwise, if all comparisons in Step 1 are successful, the only possible comparison which constitutes the overhead of this shift is the comparison in Step 2 which eliminates  $q_c$ . Thus, the overhead is at most 2. Since the first mismatch in Steps 1 and 2 eliminates all presuf pattern instances of the first type and since  $|x''_1| < \frac{|v_p|}{2}$ , either the overhead is 0 or the next presuf shift occurs distance at least  $\frac{3(m+1)}{4}$  to the right.

Consider two cases now.

1. Suppose  $q_c$  survives Step 2. All comparisons made in Steps 2 and 3 are charged to the text characters compared. Thus, each text character which lies to the right of  $t_a$  and is aligned with or to the left of  $q_c|$  is charged at most once over Steps 1, 2 and 3. All future comparisons will be charged to text characters to the right of  $q_c|$ .
2. Suppose  $q_c$  does not survive Step 2. Each successful comparison in Step 2 eliminates some pattern instance lying entirely to the right of  $q_c[j]$  and is charged to the text character

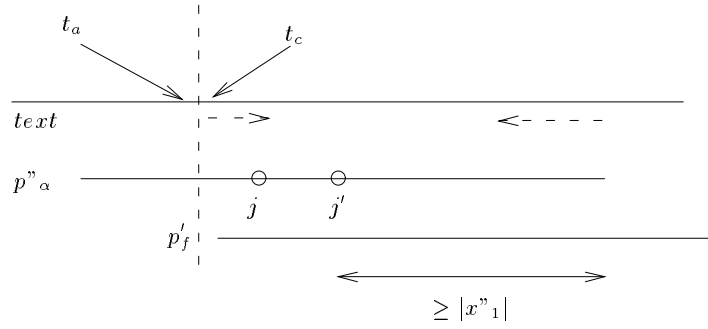


Figure 7: Step 1 of Case 2

aligned with the left end of that pattern instance. The unsuccessful comparison which eliminates  $q_c$  in Step 2 is charged to the text character aligned with  $q_c[j]$  if  $q_c$  is not the leftmost presuf pattern instance. Thus, each text character lying between  $t_a$  and  $|q_d|$  is charged at most once, where  $q_d$  is the leftmost surviving pattern instance at the end of Step 2. All future comparisons will be charged to text characters aligned with or to the right of  $|q_d|$ .

□

**Case 2.**  $|x''_1| \geq \frac{|v_p|}{2}$ .

There are 5 steps in the presuf shift handler for this case. At most 5 mismatches are made in these steps. We show that three of these mismatches can be charged to unmatched text characters; consequently, the overhead of the current presuf shift is at most two. Further, the first mismatch in Step 1 eliminates  $p''_\alpha$  and the second mismatch eliminates all the presuf pattern instances.

**Step 1.** Step 1 eliminates all but at most one of  $p''_1, \dots, p''_{k+1}$  as follows. See Fig.7. The following sequence of text characters is compared with the aligned characters in  $p''_\alpha$ :  $t_b$ , followed by the text characters strictly between  $t_b$  and  $p''_\alpha[j']$  considered right to left, followed by the text characters strictly between  $t_a$  and  $p''_\alpha[j]$  considered left to right. Step 1 terminates when the first mismatch occurs or when this sequence gets exhausted.

Let  $p'_e$  be the leftmost surviving presuf pattern instance following Step 1. Consider the pattern instance  $p'_f$ ,  $|p'_f|$  aligned with the text character to the immediate right of  $t_c$ . Let  $t_e$  be the text character at which the mismatch occurred, if any. Note that since  $j \geq |x''_1| + 1$  and  $|x''_1| \geq \frac{|v_p|}{2}$ , by Lemma 6.8,  $p'_f[j]$  must be to the right of  $p''_\alpha[j]$ . The outcome of Step 1 depends upon which of the following two cases occur (see Fig.8).

**Case 1.1.**  $p'_f[j]$  is aligned with or to the left of  $t_e$  (first diagram in Fig.8). Clearly,  $p''_\alpha[j']$  is to the left of  $t_e$ . We show that a transfer function similar to the function  $f''$  of Section 6.1 (see Lemma 6.6) can be used to account for the comparisons made in Step 1. In this case, the rest of steps are identical to Steps 3, 4 and 5 of the presuf shift handler of Section 6.1.

**Case 1.2.** Either there is no mismatch in Step 1 or  $p'_f[j]$  is to the right of  $t_e$  (second and third diagrams in Fig.8). The leftmost surviving pattern instance with left end to the right of  $t_c$  has

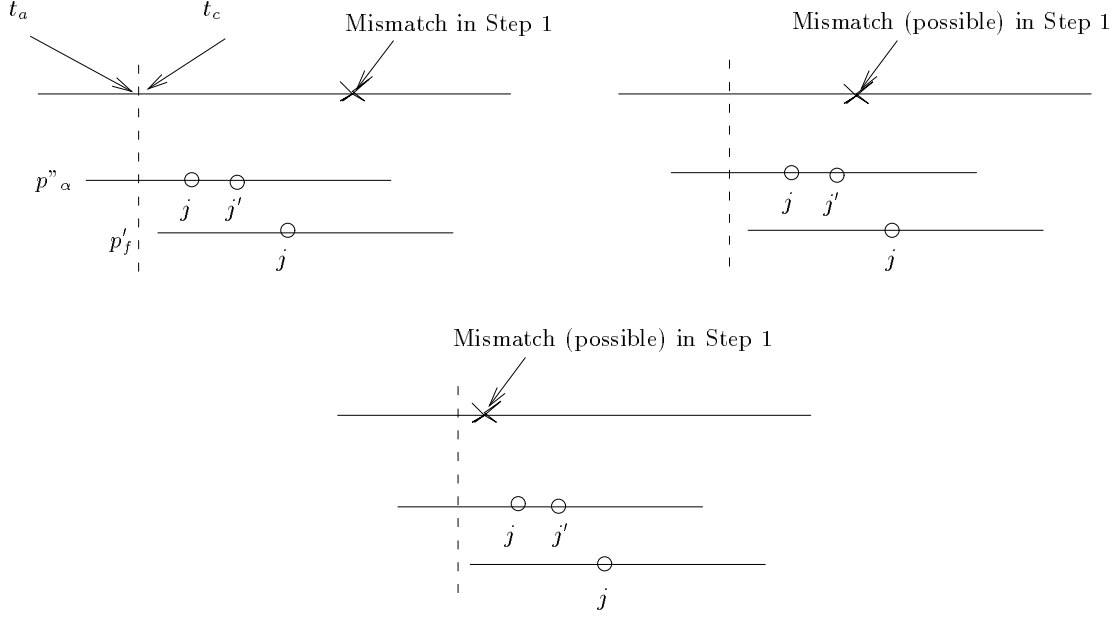


Figure 8: Possible Outcomes of Step 1

its  $j$ th character to the right of  $t_b$ ; we show this claim in the next paragraph. Step 2 follows in this case.

Recall that  $p'_f[j]$  is to the right of  $p''_\alpha[j']$ . The mismatch, if any, in Step 1 occurs to the left of  $p'_f[j]$ . Therefore, all text characters aligned with or to the right of  $p'_f[j]$  and to the left of (and including)  $t_b$  are identical to  $b$ . The claim follows.

**Step 2.** If  $p'_e$  does not extend to the right of  $t_b$  then no comparisons are made in this step (this happens if and only if  $p'_e$  is the leftmost presuf pattern instance). Otherwise, Step 2 attempts to extend the match of  $p'_e$ . Characters in  $p'_e$  to the right of  $t_b$  (if any) are compared from left to right until a mismatch occurs or a non- $b$  character is matched against the text. To see that  $p'_e$  will have a non- $b$  character to the right of  $t_b$  if it extends to the right of  $t_b$ , note that the distance between  $t_b$  and  $t_c$  equals  $|v_p| - 1$  and that  $p'_e$  has at least two non- $b$  characters distance  $|v_p|$  apart, neither of which can be to the left of  $t_c$ .

The successful comparisons in this step will be charged to the text characters compared. Clearly, all these text characters are to the right of the text characters compared in Step 1.

**Step 3.** A pattern instance  $p'_g$  with the following properties is determined in this step.

1.  $p'_g$  is the leftmost surviving pattern instance.
2. All surviving pattern instances which overlap  $p'_g[i]$  are presuf overlaps of  $p'_g$ , where  $i$  is defined as follows. If  $p'_g \neq p'_e$ ,  $i = j$ . If  $p'_g = p'_e$  and  $p'_e$  is the leftmost presuf pattern instance then  $p'_g[i]$  is the character aligned with  $t_b$ . Otherwise, if  $p'_g = p'_e$  and  $p'_e$  is not the leftmost presuf pattern instance then  $p'_g[i]$  is the leftmost non- $b$  character in  $p'_g$  which is to the right of  $t_b$ .

All text characters compared successfully in this step will be distinct from all text characters compared successfully in Steps 1 and 2. There are two cases depending upon the outcome of Step 2.

**Case 2.1.**  $p'_e$  is eliminated in Step 2. At most two mismatches could have occurred in Steps 1 and 2. Note that  $p'_e$  cannot be the leftmost presuf pattern instance in this case. The leftmost surviving pattern instance must have its left end to the right of  $t_c$ . As shown in Step 1, its  $j$ th character must be to the right of  $t_b$ . There are two subcases.

**Case 2.1.a.** Step 2 terminates in a mismatch at a non- $b$  character  $t_h$  in  $p'_e$ . Then, starting at  $t_h$ , a left to right pass is made in which each text character is compared with  $b$ . This pass ends when a mismatch occurs or when the right end of the text is reached. In the latter case, there are no further occurrences of the pattern and the algorithm terminates. In the former case,  $p'_g$  is defined to be the pattern instance in which  $p'_g[j]$  is aligned with the text character  $t_x$  at which the mismatch occurs. Since all text characters strictly between  $t_b$  and  $t_x$  are identical to  $b$ ,  $p'_g$  is the leftmost surviving pattern instance and all pattern instances to the right of  $p'_g$  which overlap  $p'_g[j]$  are eliminated. Note that the number of mismatches made in Steps 1–3 is at most 3 in this case.

**Case 2.1.b.** Step 2 terminates in a mismatch at a character  $t_h$  in  $p'_e$  which is a  $b$ .  $p'_g$  is defined to be the pattern instance in which  $p'_g[j]$  is aligned with the text character at which the mismatch occurs. As in the previous case,  $p'_g$  is the leftmost surviving pattern instance and all pattern instances to the right of  $p'_g$  which overlap  $p'_g[j]$  are eliminated. The number of mismatches made in Steps 1–3 is at most 2 in this case.

**Case 2.2.**  $p'_e$  survives Step 2. At most one mismatch could have occurred so far. There are two subcases.

**Case 2.2.a.**  $p'_e$  is not the leftmost presuf pattern instance, i.e., it extends to the right of  $t_b$ . Let  $t_x$  be the rightmost text character matched in Step 2.  $t_x$  must be a non- $b$  character. Consider the pattern instance  $p'_h$ , where  $p'_h[j]$  is aligned with  $t_x$ .

Clearly, all pattern instances to the right of  $p'_h$  which overlap  $t_x$  are eliminated as each has a  $b$  aligned with  $t_x$ . We claim that all pattern instances strictly between  $p'_e$  and  $p'_h$  have also been eliminated. This is shown as follows. All pattern instances to the right of  $p'_e$  which overlap  $t_c$  have been eliminated in Step 1. Recall from Step 1 that the leftmost surviving pattern instance after Step 1 with left end to the right of  $t_c$  has its  $j$ th character to the right of  $t_b$ . Since all text characters to the right of  $t_b$  and up to but not including  $t_x$  are identical to  $b$ , the claim follows.

If  $p'_e$  and  $p'_h$  lack a difference point or if  $p'_h[j]$  does not match  $t_x$  then  $p'_g = p'_e$ . Otherwise, if  $p'_e$  and  $p'_h$  have a difference point, the character in  $p'_e$  at that difference point is compared with the aligned text character and one of  $p'_e$  and  $p'_h$  is eliminated; the difference point itself is to the right of  $p'_h[j]$ . Let  $p'_g$  denote the survivor. Clearly,  $p'_g$  is the leftmost surviving pattern instance in both cases. Further, all pattern instances to the right of  $p'_g$  which overlap  $t_x$  (note that  $t_x$  is aligned with  $p_g[i]$ ) have either been eliminated or are presuf overlaps of  $p'_g$ .

At most two mismatches are made in Steps 1–3 in Case 2.2.a.

**Case 2.2.b.** Second, suppose  $p'_e$  is the leftmost presuf pattern instance. Recall that  $p'_e[m]$  is aligned with  $t_b$ . In this case, no comparisons are made in Step 2. Consider the pattern instance  $p'_h$ , where  $p'_h[j]$  is to the immediate right of  $t_b$ . Recall from Step 1 that  $p'_h$  is the leftmost surviving pattern instance with left end to the right of  $t_c$ . The only surviving pattern instances to the right of  $p'_e$  which overlap  $t_c$  are presuf overlaps of  $p'_e$ . Therefore  $p'_h$  is the leftmost surviving

pattern instance, barring  $p'_e$  and its presuf overlaps. In addition, note that any pattern instance which overlaps  $p'_e$  but not  $p''_\alpha[j']$  and has its  $j$ th character to the right of  $t_b$  is a presuf overlap of  $p'_e$ .

If  $|p'_h$  is to the right of  $p''_\alpha[j']$  then  $p'_h$  is a presuf overlap of  $p'_e$  as are all pattern instances to the right of  $p'_h$  which overlap  $p'_e$ . Step 5 follows with  $p'_g = p'_e$  in this case.

Otherwise, if  $p'_h$  overlaps  $p''_\alpha[j']$  then  $p'_h$  is not a presuf overlap of  $p'_e$ . The character  $p''_\alpha[j']$  is then compared with the text. A match eliminates all pattern instances which overlap  $p''_\alpha[j']$  but are not presuf overlaps of  $p'_e$  (this can be seen from the following two facts: (a) all pattern instances with left end to the right of  $t_c$  which survive Step 1 have their  $j$ th character to the right of  $p''_\alpha[j']$ , and (b) all surviving pattern instances which overlap  $t_c$  are presuf overlaps of  $p'_e$ ). Clearly, all surviving pattern instances which overlap  $t_b$  are presuf overlaps of  $p'_e$ . In this case, Step 5 follows with  $p'_g = p'_e$ . Otherwise, if a mismatch occurs at  $p''_\alpha[j']$ ,  $p'_e$  is eliminated as are all its presuf overlaps which overlap  $t_c$ . Text characters to the right of  $t_b$  are now compared from left to right with the character  $b$  until either a mismatch occurs or the right end of the text is reached. In the former case, Step 4 follows with  $p'_g$  denoting the pattern instance such that  $p'_g[j]$  is aligned with the text character at which the mismatch occurs. Clearly,  $p'_g$  is the leftmost surviving pattern instance and all pattern instances which overlap  $p'_g[j]$  have been eliminated. In the latter case (i.e., the right end of the text is reached), the algorithm terminates as there are no further occurrences of the pattern.

At most two mismatches are made in Steps 1–3 in Case 2.2.b.

**Step 4.** In this step, either all surviving pattern instances which overlap  $p'_g$  are eliminated (except for presuf overlaps) or  $p'_g$  is eliminated. In the latter case, the basic algorithm is resumed with the leftmost surviving pattern instance. In the former case, Step 5 follows. All comparisons in this step are to the right of all text characters matched in the previous steps. In addition, the left end of each pattern instance eliminated in this step is also to the right of any text character matched in one of the previous steps.

In Step 4, difference point comparisons are used. This step has a number of iterations. In each iteration, a different pattern instance overlapping  $p'_g$  but strictly to the right of  $p'_g[i]$  is considered. If it is a presuf overlap of  $p'_g$  then nothing is done. Otherwise, if it is not a presuf overlap of  $p'_g$ , the character in  $p'_g$  at the difference point of the two pattern instances is considered. If the text character aligned with this character has not been successfully compared earlier (this is ascertained using a bit vector), the two characters are compared. Step 4 ends when a mismatch occurs or when all pattern instances overlapping  $p'_g$  (excluding presuf overlaps) are eliminated.

If no mismatch occurs in Step 4 then all comparisons in this step will be charged to the text characters compared, otherwise they will be charged to left ends of the pattern instances eliminated. In both cases, the text characters charged are to the right of all text characters matched in previous steps.

**Step 5.** This step attempts to complete the match of  $p'_g$ . Characters in  $p'_g$  which have not yet been matched are compared with the aligned text characters from right to left till a mismatch occurs or all its characters are matched. In either case, another presuf shift follows. All comparisons in this step will be charged to the text characters compared.

**Lemma 6.18** *If  $p$  is a special case pattern with  $|x''_1| \geq \frac{|v_p|}{2}$  then the comparison complexity of the algorithm is  $n(1 + \frac{8}{3(m+1)})$ .*

**Proof:** We give charging strategies to show that a presuf shift can have overhead at most two. Further, we show that an overhead of one forces the next presuf shift to occur at least distance  $\frac{m+1}{2}$  to the right and an overhead of two forces the next presuf shift to occur at least distance  $\frac{3(m+1)}{4}$  to the right. The lemma follows.

As in Lemma 4.12, the run of the algorithm is divided into phases; a phase can be of one of four types. The range of text characters charged in each type of phase remains exactly the same as in Lemma 4.12. The charging scheme for Type 1 and Type 2 phases also remains exactly the same. Only the charging scheme for Type 3 and Type 4 phases is modified in accordance with the presuf shift handler described above.

We consider a single phase, which could be a Type 3 or a Type 4 phase. We assume that this phase begins with a presuf shift with  $|x'_1| \geq \frac{m}{2}$ .

**The Charging Scheme.** We consider two cases.

**Case A.** Suppose a mismatch occurs in Step 1 at a text character  $t_e$  to the right of  $p''_\alpha[j]$  and  $p'_f[j]$  is aligned with or to the left of  $t_e$  (i.e. Case 1.1 in Step 1 holds).

Each successful comparison in Step 1 matches the character  $b$  against the text. The charging scheme for this case is identical to the charging scheme in Lemma 6.7 with the function  $f''$  defined as follows.  $f''$  is defined to map each text character compared successfully in Step 1 to the text character which is distance  $j - 1$  to its left. This definition of  $f''$  is easily verified to satisfy all four required properties of  $f''$  stated in Lemma 6.6. Further, only the last Step 1 comparison can possibly be unsuccessful and might not receive an  $f''$  value. From the above charging scheme, it follows that the overhead of the current presuf shift is at most one.

**Case B.** Suppose all comparisons in Step 1 are successful or  $p'_f[j]$  is to the right of  $t_e$ , the character at which the mismatch in Step 1 occurs (i.e., Case 1.2 in Step 1 holds).

Recall from Step 1 that the leftmost surviving pattern instance completely to the right of  $t_c$  must have its  $j$ th character to the right of  $t_b$ . There are three subcases to consider.

**Subcase B.1.** Suppose  $p'_e$  (the leftmost of the presuf pattern instances to survive Step 1) survives Steps 2, 3 and 4.

All successful comparisons in Steps 1, 2, 3 and 4 and all comparisons in Step 5 are charged to the text characters compared. All these comparisons involve distinct text characters and thus, each text character which is to the right of  $t_a$  and aligned with  $p'_e$  is charged at most once. Further, at most one mismatch is made in Step 1 and no mismatches are made in Steps 2, 3 and 4 (for otherwise,  $p'_e$  would be eliminated). In addition, a mismatch in Step 1 eliminates  $p''_\alpha$ , thus forcing the next presuf shift to occur at least distance  $\frac{m+1}{2}$  to the right. Thus, the current presuf shift has overhead at most one and an overhead of one forces the next presuf shift to occur distance at least  $\frac{m+1}{2}$  to the right.

**Subcase B.2.** Suppose  $p'_e$  is eliminated in one of Steps 2, 3 and 4; further, suppose  $p'_e$  is the leftmost presuf pattern instance.

In this case, the next presuf shift occurs distance at least  $m + 1$  to the right. We show an overhead of at most two for this case.

All comparisons in Step 1 are successful and are charged to the text characters compared. No comparisons are made in Step 2.  $p'_e$  must be eliminated in Step 3 as Step 5 follows directly from Step 3 otherwise (see Case 2.2.b in Step 3). All successful comparisons in Step 3 are charged to the text characters compared. At most two mismatches are made in Step 3 and these constitute the overhead of this shift. All text characters matched in Steps 1–3 are to the left of  $p'_g[j]$ . If

$p'_g$  survives Step 4, then all comparisons in Step 4 are charged to the text characters compared. If  $p'_g$  does not survive Step 4 then the comparison which eliminates  $p'_g$  is charged to the text character aligned with  $p'_g[j]$  and all other comparisons in Step 4 are charged to the left ends of the respective pattern instances eliminated (from the definition of  $p'_g$  in Step 3, note that the left ends of these pattern instances are to the right of  $p'_g[j]$ ). Thus all text characters charged in Step 4 are distinct and aligned with or to the right of  $p'_g[j]$ . All comparisons in Step 5 are charged to the text characters compared. These text characters are distinct from all text characters matched in the previous steps. Therefore, if  $p'_g$  survives Step 4 then each text character to the right of  $t_a$  and aligned with or to the left of  $p'_g$  is charged at most once. Otherwise, if  $p'_g$  is eliminated in Step 4 and  $p'_l$  is the leftmost surviving pattern instance following Step 4, each text character strictly between  $t_a$  and  $|p'_l$  is charged at most once.

**Subcase B.3.** Suppose  $p'_e$  is eliminated in one of Steps 2, 3 and 4 and  $p'_e$  is not the leftmost presuf pattern instance.

In this case, the next presuf shift occurs distance at least  $m + 1$  to the right. We show an overhead of at most two for this case.

All successful comparisons in Steps 1 and 2 and all comparisons in Step 5 are charged to the text characters compared. If  $p'_e$  does not survive Step 2 (Case 2.1 of Step 3) then all successful comparisons in Step 3 are charged to the text characters compared. Otherwise, if  $p'_e$  survives Step 2 (Case 2.2.a of Step 3), there is at most one comparison in Step 3 and it is accounted for later. If  $p'_g$  survives Step 4 then all successful comparisons in Step 4 are charged to the text characters compared. In this case, each text character to the right of  $t_a$  and aligned with or to the left of  $p'_g$  is charged at most once. Otherwise, if  $p'_g$  is eliminated in Step 4, each successful comparison in Step 4 (except the one which eliminates  $p'_g$ ) is charged to the text character aligned with the left end of the pattern instance eliminated by this comparison; this text character is to the right of  $p'_g[j]$ . In this case, each text character strictly between  $t_a$  and  $|p'_l$  is charged at most once, where  $p'_l$  is the leftmost surviving pattern instance after  $p'_g$  is eliminated.

At most four comparisons have not yet been accounted for. These include the mismatch in Step 1, the mismatch in Step 4 (which eliminates  $p'_g$ ) and either the mismatches in Steps 2 and 3 or the only comparison in Step 3, depending on whether or not  $p'_e$  survives Step 2. Note that if mismatches occur in all of Steps 2, 3 and 4 then the text character aligned with  $p'_g[j]$  is not charged for any comparison. In addition, we show that the text character aligned with  $p''_\alpha[j]$  is also not charged for any comparison. An overhead of 2 for the current presuf shift follows immediately.

Clearly,  $p''_\alpha[j]$  is not compared in Step 1. All comparisons in Steps 2, 3, and 4 are made to the right of  $t_b$ , and hence to the right of  $p''_\alpha[j]$ . Further,  $p'_g[i]$  ( $i$  as defined in Step 3) is aligned with or to the right of  $t_b$ . Therefore, the text character aligned with  $p''_\alpha[j]$  is not charged for any of the comparisons made in Steps 1–4. Consider Step 5 next. If  $p'_e$  survives Steps 2 and 3 and is eliminated in Step 4 then the presuf shift handler terminates after Step 4 and the basic algorithm is resumed. So suppose that  $p'_e$  is eliminated in Step 2 or Step 3. From the definition of  $p'_g$  in Step 3,  $p'_g \neq p'_e$  and therefore  $i = j$ . To show that all comparisons in Step 5 are made to the right of  $p''_\alpha[j]$ , it suffices to show that  $|p'_g$  is to the right of  $p''_\alpha[j]$ .

We show this by considering two cases. First, suppose  $p''_\alpha \neq p''_1$ . Then, by Lemma 6.8, it follows that the character in  $p''$  which is to the immediate left of its suffix  $x''_1$  is a  $b$ . Since  $x''_1$  is the longest presuf of  $p''$ , it follows that  $j = |x''_1| + 1$ . By Lemma 6.8,  $p''_\alpha[j]$  and hence  $p''_\alpha[j]$  are to the left of the suffix  $x''_1$  of  $p''_1$ . Since  $|p'_g[j]$  is to the right of  $t_b$ ,  $|p'_g$  is aligned with or to the

right of the suffix  $x_1''$  of  $p_1''$ . The claim follows for this case.

Next, suppose  $p_\alpha'' = p_1''$ . A mismatch occurs in Step 1 as  $p_e'$  is not the leftmost presuf pattern instance. Further, this mismatch occurs at some text character  $t_e$  to the right of  $p_\alpha''[j]$  as no comparisons are made to the left of  $p_\alpha''[j]$  in Step 1 in this case. Each comparison in Step 1 compares a text character with  $b$ . Since  $p_g'[j]$  must be to the right of  $t_b$ , either  $p_g'$  is to the right of  $t_e$  or a  $b$  in  $p_g'$  overlaps  $t_e$ . But  $p_g'$  will not survive in the latter case. The claim follows.

The lemma follows.  $\square$

Finally, we state the following theorem; the proof is similar to the proof of Theorem 4.17.

**Theorem 6.19** *There is a string matching algorithm with a comparison complexity of  $n(1 + \frac{8}{3(m+1)})$  comparisons which uses  $O(m)$  space and takes  $O(n + m)$  time following preprocessing of the pattern; the preprocessing time is  $O(m^2)$ .*

## 7 Acknowledgements

We thank Dany Breslauer for a number of comments and suggestions and in particular for the observation that  $p_{k+1}$  could be viewed as a presuf pattern instance. This contributed to an improvement of our upper bound from  $n + \frac{3(n-m)}{m+1}$  to  $n + \frac{8(n-m)}{3(m+1)}$ .

## References

- [AC89] A. APOSTOLICO, M. CROCHEMORE *Optimal Canonization of all substrings of a string*, Technical Report No. TR 89-75, LITP, Universite Paris 7, FRANCE, October 1989.
- [AG86] A. APOSTOLICO, R. GIANCARLO *The Boyer-Moore-Galil string searching strategies revisited*, SIAM Journal on Computing, Vol.15, 1(1986), pp. 98–105.
- [BM77] R. BOYER, S. MOORE *A fast string matching algorithm*, CACM, 20(1977), pp. 762–772.
- [B94] D. BRESLAUER *Saving comparisons in the Crochemore-Perrin string matching algorithm*, Manuscript, 1994.
- [BCT93] D. BRESLAUER, L. COLUSSI, L. TONIOLO *Tight comparison bounds for the string prefix-matching problem*, Information Processing Letters, 47(1993), pp. 51–57.
- [BG92] D. BRESLAUER, Z. GALIL *Closing the gap in the exact complexity of string matching*, To appear, Journal of Complexity.
- [Co91] R. COLE *Tight Bounds on the complexity of the Boyer-Moore algorithm*, SIAM Journal on Computing, Vol. 23, No. 5, 1994, pp. 1075–1091.
- [CHPZ92] R. COLE, R. HARIHARAN, M. PATERSON, U. ZWICK *Tighter lower bounds on the exact complexity of string matching*, SIAM Journal on Computing, Vol. 24, No. 1, 1995, pp. 30–45.
- [CCG92] M. CROCHEMORE, A. CZUMAJ, L. GASINIEC, S. JAROMINEK, T. LECROQ, W. PLANDOWSKI, W. RYTTER *Speeding Up Two String-Matching Algorithms*, Algorithmica, 5(1994), pp. 247–267.



- [Col91] L. COLUSSI *Correctness and efficiency of pattern matching algorithms*, Information and Computing, Vol. 5, 1991, pp. 225–251.
- [CGG90] L. COLUSSI, Z. GALIL, R. GIANCARLO *On the Exact Complexity of String Matching*, Proceedings of the Thirty First Annual IEEE Symposium on the Foundations of Computer Science, 1990, pp. 135–143.
- [CP89] M. CROCHEMORE, D. PERRIN. *Two-way pattern matching*, Technical Report, Laboratoire Informatique, Théorique et Programmation, Université Paris 7, 1989.
- [FW65] N. FINE, H. WILF *Uniqueness theorem for periodic functions*, American Mathematical Society, 16(1965), pp. 109–114.
- [GG91] Z. GALIL, R. GIANCARLO *On the Exact Complexity of String Matching: Lower Bounds*, SIAM Journal on Computing, 6(1991), pp. 1008–1020.
- [GG92] Z. GALIL, R. GIANCARLO *On the Exact Complexity of String Matching: Upper Bounds*, SIAM Journal on Computing, 3(1993), pp. 407–437.
- [GS80] Z. GALIL, J. SEIFERAS *Saving space in fast string-matching*, SIAM Journal on Computing, 2(1980), pp. 417–438.
- [GO80] L. J. GUIBAS, A. M. ODLYZKO *A new proof of the linearity of the Boyer-Moore string searching algorithm*, SIAM Journal on Computing, Vol.9, 4(1980), pp. 672–682.
- [Ha93] C. HANCART *On Simon's string searching algorithm*, Information Processing Letters, 1993.
- [KMP77] D. E. KNUTH, J. MORRIS, V. PRATT *Fast pattern matching in strings*, SIAM Journal on Computing, 6(1973), pp. 323–350.
- [Lo82] M. LOTHAIRE *Combinatorics on words*, *Encyclopaedia of Mathematics and its Applications*, Addison-Wesley, Vol. 17, 1982.
- [LS62] R. LYNDON, M. SCHUTZENBERGER *The equation  $a^m = b^n c^p$  is a free group*, Journal of Michigan Mathematics, 9(1962), pp. 289–298.
- [Vi85] U. VISHKIN *Optimal parallel pattern matching in strings*, Information and Control, 67(1985), pp. 91–113.
- [ZP92] U. ZWICK, M. PATERSON *Lower bounds for string matching in the sequential comparison model*, Manuscript.