k One-way Heads Cannot Do String-Matching

Tao Jiang* McMaster University

Ming Li† University of Waterloo

Abstract

We settle a conjecture raised by Z. Galil and J. Seiferas 14 years ago [6]: k-head one-way deterministic finite automata cannot perform string-matching (*i.e.*, accept the language $\{x\#y \mid \exists u \exists v \; y = uxv\}$), for any k non-sensing heads.

1 Introduction

The string-matching problem is defined as follows [6]: Given a character string x, called the *pattern*, and a character string y, called the *text*, determine if x is a subword of y. It is well-known that the string-matching problem is of great importance in practice, for example the "grep" function in UNIX.

Since the invention of the linear-time algorithms of [2, 3, 8], there have been constant efforts to find linear-time algorithms which use less space. See, *e.g.*, [1]. The best theoretical result is in [6], where it was shown that string-matching can be performed by a six-head *two-way* deterministic finite automaton in linear time. That is, in linear time, such a machine can accept the language

$$L = \{x\#y | x \text{ is a substring of } y\}. \tag{1}$$

The next obvious question is whether string-matching can be performed without backing up the pointers in *either* the text *or* the pattern. (Many existing string-matching algorithms do not back up in the text. But they all have to back up in the pattern.) One can formalize this question as whether a k-head *one-way* deterministic finite automaton (k-DFA) can accept L, for some k.

In [6], Galil and Seiferas conjectured that a k-DFA cannot do string-matching, for any constant k. This question was also listed in [5] as a major open question in string-matching

*Supported in part by NSERC Operating Grant OGP0046613. Address: Department of Computer Science, McMaster University, Hamilton, Ont. L8S 4K1, Canada. E-mail: jiang@maccs.mcmaster.ca

†Supported in part by NSERC Operating Grant OGP0046506. Address: Department of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada. E-mail: mli@math.uwaterloo.ca

research. Here, we assume that the heads are *non-sensing*, *i.e.*, they cannot detect their coincidences. Another version of this question adds power to the heads by allowing them to detect each other when they coincide. Here the machine is slightly more powerful and it is more difficult to prove the lower bound. At this moment, our proof does not apply to this version.

The conjecture, although it looks obviously true, turns out to be quite difficult to prove. In [9], the conjecture was proved for $k = 2$. Then, in [7], the conjecture was proved for $k = 3$. Chrobak and Rytter [4] solved some other variants of this problem. We shall confirm this conjecture by proving: no k-DFA accepts $L = \{x\#y|\ x \text{ is a substring of } y, \text{ where } x, y \in \{0,1\}^*\}$. The case when the heads are sensing remains open.

Although [9, 7] only solved some very special cases, they provide some useful tools for the final proof here. Reference [9] contains the Matching Lemma. Reference [7] contains the idea that when some heads move independently through some sufficiently long random segments, then these heads will move obliviously to the end (called the Moving Lemmas in [7]). The proof uses Kolmogorov random strings and tries to establish a contradiction through compression of the input. But this was not enough to prove the general case since the (global) compression scheme used in the Moving Lemmas in [7] does not work when the heads are sometimes near and sometimes far from each other. In this paper, we develop a new (and much more sophisticated) recursive (local) compression scheme that finally enables us to deal with the problem.

2 Preliminaries and Notation

A k-DFA $M = <\Sigma, Q, \delta, q_0, F>$ is a deterministic finite automaton with k one-way read-only heads $h_1, h_2, \ldots, h_k$. The symbols Σ, Q, δ, q_0 and F represent the alphabet, the set of states, the transition function, the starting state, and the set of final states, respectively. Without loss of generality, let $\Sigma = \{0, 1\}$. M has a one-way read-only input tape. We assume that the input to M is of the form, \$*pattern*#*text*\$ where $pattern, text \in \Sigma^*$. The \$ signs serve as endmarkers. Initially, all k heads are on the first \$ sign. At each step, depending on the current state and the ordered k-tuple of symbols seen by the heads $h_1, h_2, \ldots, h_k$, M deterministically changes its state and moves at least one of the heads one position to the right. On each input all heads will eventually reach and stay at the final \$ sign. *M accepts* an input if it is in a final state when all heads reach the final \$ sign. The heads $h_1, \ldots, h_k$ are non-sensing.

An *instantaneous description* (or ID) of M is a $(2k+1)$-tuple $(q, p_1, a_1, \ldots, p_k, a_k)$, where q is the state and p_i and a_i, $1 \le i \le k$, are the position of the i-th head and the symbol read by the i-th head. On a given input, we write $\mathrm{ID}_1 \vdash_M^* \mathrm{ID}_2$ if M, starting from ID_1, reaches ID_2 in zero or more steps.

Kolmogorov complexity has played an important role in lower bound proofs. Its usage in lower bound proofs was first introduced in [11, 12]. Consider binary strings only. Informally, the Kolmogorov complexity of a string x, denoted by $K(x)$, is the length of the shortest program that prints x. The conditional Kolmogorov complexity of x with respect to y, denoted by $K(x|y)$, is the length of the shortest program which, with extra input information y, prints x. In some cases, we want to encode x in the *self-delimiting* form $\bar{x}$. Here $\bar{x}$ is defined to be $l'01x$, where l is the binary representation of $|x|$ and l' is obtained by doubling every bit in l. Thus $|\bar{x}| \leq |x| + 2\log|x| + 2$. Given $\bar{x}y$, one can uniquely decompose $\bar{x}y$ into x and y. We state without proofs some well-known facts which we will use implicitly or explicitly. For formal definitions of Kolmogorov complexity and the proofs, see [10].

Fact 1. For each integer n and each string y, there exists a string x of length n such that $K(x|y) \leq n$. It is easy to show that most strings x have high Kolmogorov complexity $K(x)$ close to $|x|$. We call such strings *random strings.*

Fact 2. If $K(x|y) \geq f(x)$ and $x = uvw$, then $K(v|uw) \geq f(x) - 2\log|x|$.

Fact 3. (Symmetry of Information) Up to a logarithmic additive term, $K(x) + K(y|x) = K(y) + K(x|y)$. Thus, if y does not contain too much information about x, then x also does not contain too much information about y. This fact is still true conditioning to another string z everywhere.

We use $y \setminus x$ to denote the string obtained by deleting all occurrences of x from y. We will always make sure that the occurrences of x in y do not overlap, and thus $y \setminus x$ is unique. All log's are base 2 in this article.

3 The Main Theorem

In this section, we prove that no k-DFA can do string-matching. The proof relies on two key lemmas: the Matching Lemma mentioned earlier and a new Moving Lemma which asserts that, for any k-DFA, if some heads move "too many" steps during the computation on a random input while the other heads remain stationary, then these heads will move *unconsciously* (*i.e.*, regardless of the subsequent input symbols read) until one head reaches the endmarker. We will use this lemma to draw all the heads of a k-DFA out of the pattern x so that we can apply the Matching Lemma.

Definition 1 Let x and y be any two disjoint segments of the input I of a k-DFA M. We say that M *matches* x and y if, on input I, there is a time such that M moves one head at least one step in x while another head is in y and vice versa.

The above definition and the following lemma simply try to capture our intuition that to make sure x is a substring of y, the k-DFA must have a head in pattern x and another head in a copy of x in the text y and move these two heads simultaneously to match them. The proof of a slight variant of the following Matching Lemma can be found in [7, 9], so we omit the proof of the lemma here.

Lemma 1 (Matching Lemma) *Let M be any k-DFA, $I = \$x\#y\$$ an input, and $\mathrm{ID}_1 \vdash_M^* \mathrm{ID}_2$ a computation of M on I. Assume that $K(x|y \setminus x) \geq n - O(\log|I|)$, where $n = |x|$, and the occurrences of x in y do not overlap.*

1. *If the pattern x is not matched with any copy of x in the text y, then there are $2^n/n^{O(1)}$ strings x' such that on input $\$x'\#y\$$, we also have $\mathrm{ID}_1 \vdash_M^* \mathrm{ID}_2$. Here the constant $O(1)$ depends only on M and the constant factor in the hypothesis $K(x|y \setminus x) \geq n - O(\log|I|)$.*
2. *Suppose that there are c copies of x in y and that each is not matched with the pattern x or any other copy of x in y during the computation $\mathrm{ID}_1 \vdash_M^* \mathrm{ID}_2$. Then there is an $x' \neq x$ such that replacing these c copies of x in y with x' in the input also results in $\mathrm{ID}_1 \vdash_M^* \mathrm{ID}_2$. In fact, there are $2^n/n^{O(1)}$ such x', where the constant $O(1)$ depends only on M, c, and the constant in $K(x|y \setminus x) \geq n - O(\log|I|)$.*

Remark. Note that y contains only $|y| - n + 1$ substrings of length n. If $|y|$ is polynomial in n, as it will be the case in our construction of a difficult input, then there is an x' which is not a substring of y in Item 1 above such that $\mathrm{ID}_1 \vdash_M^* \mathrm{ID}_2$ on input $\$x'\#y\$$. Thus, if ID_1 and ID_2 are the initial and final ID's of M, respectively, then either both $\$x\#y\$$ and $\$x'\#y\$$ are accepted by M or both are rejected. This will give us the desired contradiction. Item 2 will be used to adaptively construct a bad input in the proof of the main theorem. Our construction will always choose an x' such that substituting x' for x in y will not create new instances of x.

Our Moving Lemma is a strong extension of the 2-head Moving Lemma in [7]. The 2-head Moving Lemma in [7] only deals with two independent heads, *i.e.*, two heads that are "far away" from each other.

From now on, let $M =< \{0, 1, \#\}, Q, \delta, q_0, F >$ be a fixed DFA with a set $H_M = \{h_1, \ldots, h_k\}$ of k one-way heads. We say that a set of heads move if some head in the set moves. For convenience, we will always only consider heads that have not yet reached the right endmarker \$. In our Moving Lemma, we will want to say that if some heads do not make a move for a while, then these heads will never move again no matter what input lays ahead of them until some other moving head reaches the right endmarker \$. In this case, we say that M is unconscious of these heads. Below, we formalize this concept.

Definition 2 Let G be a group of m heads of M and $\$x\#y\$$ an input. At time t, let M be in the following *configuration*:

1. M is in state q,
2. The heads in G are reading symbols $a_1, \ldots, a_m$ (we do not care about locations of these heads),
3. The heads not in G are all in y and are at locations $p_1, \ldots, p_{k-m}$. Suppose that the rightmost of these heads is at the last bit of y_1, where $y = y_1 y_2$.

We say that M is *conscious of* G *at time* t if there exists a string y_2' such that, if M is started on input $\$x\#y_1 y_2'\$$ with exactly the same configuration as above, some head in G will *make a move* before any head not in G reaches the right endmarker \$. Otherwise, M is *unconscious of* G *at time* t. We also say that the heads not in G move unconsciously of heads in G.

Lemma 2 (Moving Lemma) *For any constants $\alpha > 0$ and β, there exists a constant φ such that, during the computation of M on an input $\$x\#y\$$, where y is a binary string such that $|y| \geq \varphi$ and $K(y) \geq |y| - \beta \log |y|$, if some heads move $|y|^\alpha$ steps (totally) in string y while the other heads remain completely stationary, then these heads will move unconsciously of the other heads until one head reaches the right endmarker \$.*

Remark. The Moving Lemma can be trivially extended to the case when we replace the full text y by a fragment y' of the text. *i.e.* when the input looks like $\$x\#z_1 y' z_2\$$ for some $z_1, z_2 \in \Sigma^*$. When we prove the main theorem, we actually use the Moving Lemma this way.

We give some intuition on how the Moving Lemma is used. For a long random pattern x and a long random segment y' of the text such that $|y'| = |x|^3$, consider the situation that some heads are in x and other heads are at the beginning of y'. Then either all heads will shift out of the pattern x in less than $k|x|^2$ steps or there is a long period ($|x|$ steps) that *all heads in the pattern* x do not move. Then the Moving Lemma will apply (with $\alpha = 1/3$ and β sufficiently large) and thus some heads will move unconsciously to the right endmarker \$, while all the heads in x remain still (thus not doing the matching). If we can somehow keep on doing this, driving all the heads out of the pattern x without doing any matching, then the Matching Lemma will yield our main theorem.

We postpone the proof of the Moving Lemma to the next section, where we first prove a Great Compression Lemma which is really the main lemma of this paper. Here we prove the main theorem using the Moving Lemma and the Matching Lemma.

Theorem 3 *No k-DFA accepts $L = \{x\#y |\ x$ is a substring of y, where $x, y \in \{0,1\}^*\}$.*

Proof. Assume that a k-DFA M accepts L. We aim for reaching a contradiction. For simplicity, we use big-O notation. In the Moving Lemma, let $\alpha = 1/3$, β be sufficiently large such that for each of the concerned random strings u below, $K(u) \geq |u| - \beta \log |u|$. Fix n to be larger than φ in the Moving Lemma. Also let $l = 2 + \sum_{i=1}^{k-1} i = 1 + (k^2 - k + 2)/2$.

We now construct a difficult input for M. Fix a string z of length $n + n^{3l}$ such that $K(z) \geq |z|$. Cut z into two segments. Let x be the segment of the first n bits of z and y be the rest. So $|y| = n^{3l}$. Note that, by the randomness of z and Fact 2 in Section 2, x and y are random relative to each other. Thus y does not contain x, or a big fraction of x, as a substring. Let $y = y_1 \cdots y_l$, where $|y_i| = n^{3i} - n^{3i-3}$ for each $i > 1$ and $|y_1| = n^3$. Thus $|y_1 \ldots y_i| = n^{3i}$ and $\sum_{i=1}^{l} |y_i| = n^{3l}$. Our strategy is to construct some input like $\$x\#y'\$$, where y' is obtained from y by inserting some copies of x and some other random strings. We would like to use the Moving Lemma to claim that all the heads will move unconsciously (of the other heads which might be in some copies of x) when they pass the copies of x in y'. Then the Matching Lemma will give us the contradiction. However the naive construction of putting a copy of x "very far" from $\#$ does not work. This is because, for example, when some head h reaches the right endmarker \$, some other moving heads moving together with h might happen to be very near to this copy of x and hence we cannot continue to use the Moving Lemma to claim that no matching will be done. To avoid this, we use the following construction.

Let $y^{(0)} = y$. In $y^{(0)}$, we insert a copy of x after every y_i, except for y_l, to obtain $y^{(1)}$. So $l - 1 = 1 + \sum_{i=0}^{k-1} i$ copies of x are inserted in total and $y^{(1)}$ looks like $y_1 x y_2 x \ldots y_{l-1} x y_l$. It is easy to prove that, for any substring u of $y^{(1)}$ of length n^3, $K(u) \geq n^3 - O(\log n)$.

Then we simulate M on input $\$x\#y^{(1)}\$$. Before the leading head reaches the first copy of x in $y^{(1)}$, either all heads are out of the pattern or there is an interval during which M makes at least $|y_1|^\alpha$ moves, with $\alpha = 1/3$, and no head in the pattern moves. If the former case is true, then we are done by Item 1 of the Matching Lemma (and the remark following it). If the latter case is true, then since $|y_1| \geq \varphi$ and $K(y_1) \geq |y_1| - \beta \log |y_1|$, the Moving Lemma implies that some heads in y_1 start to move unconsciously of all the heads in the pattern. We wait until a head reaches the \$ sign. At this moment, there are at most $k - 1$ (in fact $k - 2$ since at least one head must stay in the pattern) copies of x which are less than n^3 bits away from some head. We will replace these up to $k - 1$ copies of x with some other strings relatively random to x and $y^{(1)}$ in the following.

It follows from the Moving Lemma and the increasing separations of the instances of x in $y^{(1)}$ that no two instances of x are matched in the above process. To see this, observe that the heads that have reached the i-th copy of x must be unconscious of all the heads that are in or before the $(i-1)$-th copy of x.

Thus, by Item 2 of the Matching Lemma, there are at least $2^n/n^{O(1)}$ strings of length n that can replace these at most $k-1$ copies of x and make M reach the same ID. Out of these strings, we can choose a string $x^{(1)}$ of length n such that $K(x^{(1)}|x, y^{(1)}) \geq n - O(\log n)$. Then replace these up to $k-1$ copies of x with $x^{(1)}$ in $y^{(1)}$ to obtain $y^{(2)}$. By symmetry of information (Fact 3 in Section 2),

$$K(x, y^{(1)}|x^{(1)}) \geq K(x, y^{(1)}) - O(\log n).$$

So x and $y^{(1)}$ are relatively random to $x^{(1)}$. Hence, for any substring u of $y^{(2)}$ of length n^3, $K(u) \geq n^3 - O(\log n)$ holds.

In general, at stage i, suppose that we have already constructed $y^{(i)}$. By the construction, for each of the first $i-1$ stages, some group of heads move unconsciously until one head reaches the end. At the i-th stage, we keep on simulating M with $y^{(i)}$ as the text and apply the Moving Lemma again: either all heads leave the pattern before any head reaches a copy of x or some heads will again start moving unconsciously until reaching the right endmarker \$. In the former case, the pattern x is not matched and Item 1 of the Matching Lemma applies. In the latter case[1], when the leading head reaches the right endmarker \$, we will again find an $x^{(i)}$ of length n which is relatively random to x and $y^{(i)}$ and use $x^{(i)}$ to replace the (at most) $k-i$ copies of x that are less than n^3 bits away from some head. Denote the result by $y^{(i+1)}$. By Item 2 of the Matching Lemma, this will guarantee that M, on input $\$x\#y^{(i+1)}\$$, reaches the same ID as if the input is $\$x\#y^{(i)}\$$, by the end of i-th stage. Note, we still have that for any substring u of $y^{(i+1)}$ of length n^3, $K(u) \geq n^3 - O(\log n)$ holds.

The above construction ends when eventually $k-1$ heads have left the pattern x without matching it against a remaining copy of x in the text. This takes at most $k-1$ stages. Note that there is still at least a copy of x left in the text since we initially have $l - 1 = 1 + \sum_{i=0}^{k-1} i$ copies of x in $y^{(0)}$ and we replace at most $k-i$ copies at stage i. The theorem then follows from Item 1 of the Matching Lemma. ■

4 Proving the Moving Lemma

In this section, we prove the Moving Lemma. The proof makes an essential use of Kolmogorov complexity and a recursive compression scheme. Before we give the formal proof, we first sketch an outline of the proof, and then prove a few necessary technical lemmas.

[1]Note that it is possible that some head is very close to the right endmarker \$, then it reaches \$ right away. We consider this as a special case of the latter case.

4.1 An Outline of the Proof

In order to help the reader grasp the main lines of the proof of the Moving Lemma and not be obstructed by the technical details which are quite involved, we first sketch the main strategy and the difficulties to be solved.

The basic idea of the proof is that for any constants α, β and sufficiently long string y satisfying $K(y) \geq |y| - \beta \log |y|$, if some heads can move $|y|^\alpha$ steps in y *consciously* of the others while the others remain still, then we can compress y by at least $|y|^\epsilon$ bits for some constant $\epsilon > 0$ depending on α and M only, which is a contradiction to the choice of y.

The key to our proof is the compression scheme. The intuition behind the compression scheme is as follows. If M is conscious of some head h, there must be some "correct segment" which, if present in the input, can cause h to move. Now suppose that h has been *pausing* for a long time. This means that somehow such "correct segment" did not appear at the right time in the sections scanned by the other heads. Therefore we can partially "predict" this part of the input because we know that it lacks certain substrings. Each such "prediction" allows a compression, saving a fraction of a bit. A large number of such "small" compressions accumulate to a significant amount. The details of the compression scheme is given in the next subsection.

The above intuition may sound misleadingly "simple". There are two intricate issues that we have two solve to make the compression scheme possible. The first is that we have to know where the next "small" compressions begin in y *without* explicitly specifying their locations. (We need to know the location of each compressed segment when we decode the compressed string into y.) This is difficult since in general the compression does not proceed in a smooth left-to-right fashion. We cannot afford to record the locations explicitly since each such "small" compression may save less than a bit. Secondly, we have to make sure the parts of the input that are compressed *do not overlap* since a string cannot be compressed twice. In other words, we can only compress *disjoint* parts of the input. How to identify a large number of disjoint parts which are eligible for compressions is actually very challenging. This is because different heads may read the same segment of input at different times. As soon as a segment is compressed, it becomes an obstacle to further compression, since a subsequent compression is impossible unless all heads (actually, all heads that are moving) have shifted out of the segment, which may take a long time to happen. We have to use intricate techniques to overcome these two obstacles.

4.2 The Compression Lemmas

Now we formally describe how to compress y when some heads move $|y|^\alpha$ steps consciously while the others remain still. The central issues are resolved in Lemma 5. We will refer to

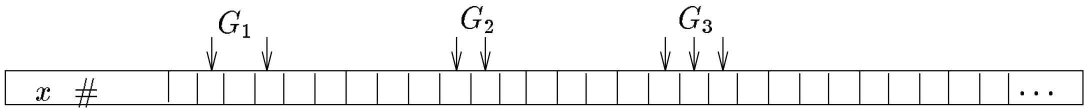

Figure 1: A $(3, 5)$-grouping of 7 heads, $g = 3$.

the heads of M that are moving during the concerned period as the *moving heads* and other heads as the *pausing* heads. Unless otherwise stated, we will always assume that *there are some pausing heads which have not reached the right endmarker \$ yet* and *M is conscious of the pausing heads*. All the moving heads we consider will be in the text y. Pausing heads may be anywhere in $\$x\#y\$$. Let's further assume that the moving heads do not reach the right endmarker \$ during the concerned period.

We often need to partition the moving heads into *groups* according to their current positions on the input tape. When we partition a set G of heads into disjoint groups $G_1, \ldots, G_g$, we write $G = (G_1, \ldots, G_g)$.

Definition 3 Let $s_1, s_2 \geq 1$ be integers. An (s_1, s_2)-grouping of the moving heads at time t is a partition of the heads into groups such that

1. the heads in each group are located in a region of size at most s_1 on the tape, disjoint from other groups;
2. the distance between any two heads in adjacent groups is greater than s_2,
3. the distance from the rightmost group to the right endmarker \$ is greater than s_2.

Figure 1 illustrates an example for Definition 3. In actual applications of the definition, we always have $s_1 < s_2$. When the parameter s_1 or s_2 is immaterial, we sometimes omit it.

Definition 4 Consider a time in the computation of M on an input $\$x\#y\$$. Let the moving heads of M be partitioned into groups $G_1, \ldots, G_g$ according to their locations as above. The *expanded state* of M (with respect to this partition of moving heads) includes the state of M, the substrings of y surrounded by the leading and trailing heads in each group, and input symbols scanned by the pausing heads.

Lemma 4 (Local Compression) *For each constant $\gamma_1 \geq 1$, there exists a constant γ_2 large enough to validate the following claim. Consider the computation of M on the input $\$x\#y\$$. Suppose that $G = (G_1, \ldots, G_g)$ is a (γ_1, γ_2)-grouping of the moving heads, all located in y, at*

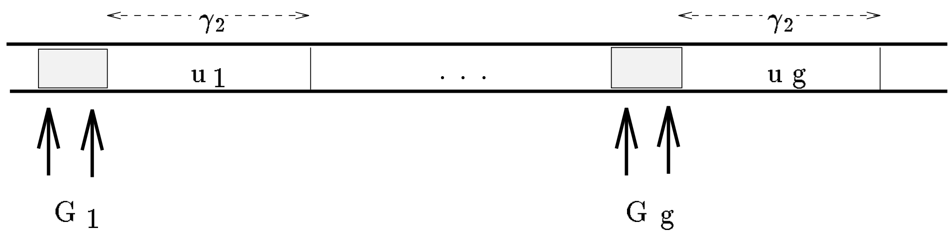

Figure 2: The groups and the strings to be compressed.

time t. For each $i = 1, \ldots, g$, let u_i be the substring of y of length γ_2 immediately to the right of the leading head in group G_i. (See Figure 2.) If all heads outside G remain stationary before a head in some group G_i reaches the last bit of u_i, then the string $u_1 \cdots u_g$ must belong in a set S of cardinality at most $2^{g\gamma_2} - 1$. Moreover, S depends only on the expanded state of M at time t.

Remark. Trivially, the string $u_1 \cdots u_g$ belongs in a set S of cardinatlity at most $2^{g\gamma_2}$ since $|u_1 \cdots u_g| = g\gamma_2$ and it is a binary string. Thus, the whole point of the above compression is just the term "-1". Although it seems tiny, it is nontrivial since both g and γ_2 are *constants* and it can save us a (constant) fraction of a bit as discussed after the proof of the lemma.

Proof. There are only a finite number of distinct expanded states given the constant γ_1 with respect to all possible $(\gamma_1, -)$-groupings of the heads. For each expanded state C, let $l(C)$ be the length of the shortest strings $u_1, \ldots, u_g$ ($|u_1| = \ldots = |u_g|$) which can lead to the movement of some pausing head if placed to the immediate right of their respective groups of moving heads. $l(C)$ is undefined if such strings do not exist, *i.e.*, M is unconscious at the expanded state C. Clearly, $l(C)$ is a constant depending only on M and C. Then we can simply let $\gamma_2 = \max_C l(C)$.

Thus, in the lemma, the string $u_1, \ldots, u_g$ is a member of the set of all binary strings of length $g\gamma_2$ minus at least a string that leads to the movement of some pausing head, since we assume that M is conscious at time t. ■

Remark. We describe how Lemma 4 will be applied. Observe that in general γ_2 is increasing in γ_1. We will use Lemma 4 with a constant number of different γ_1's in our compression algorithm to be designed. Every time we use Lemma 4, we will actually concatenate the string $u_1 \cdots u_g$ to a big string U, which is separate from the other big string V that is supposed to store all the bits not involved in the compression. (Note that we can extract the strings $u_1, \ldots, u_g$ from $u_1 \cdots u_g$ if we know g, since $|u_1| = \cdots = |u_g|$.) Lemma 4 will be repeatedly applied, and U is a concatenation of many strings of form $w = u_1 \ldots u_g$, each is from a set of size at most $2^{|w|}-1$. Thus, if U contains m such strings $w_1,\ldots,w_m$, then U is a string from a set of size at most

$$\prod_{i=1}^{m}(2^{|w_i|}-1).$$

The string U will be compressed at the end by considering the total number of choices of U. The amount of compression is

$$\sum_{i=1}^{m}|w_i|-\log\prod_{i=1}^{m}(2^{|w_i|}-1)=\sum_{i=1}^{m}(|w_i|-\log(2^{|w_i|}-1)),$$

which is at least δm for some constant $\delta>0$ depending on M only since each $|w_i|$ is bounded by some constant. The final encoding W takes the form $\overline{U'}V$, where U' is the compressed U and $\overline{U'}$ is its self-delimiting representation. In the following lemma, we will ignore the $2\log|U'|$ bits overhead for the self-delimiting representation of U'. This overhead will be considered later in the proof of the Moving Lemma. (Such overhead is much less than the saving δm in any case.) Hence, informally, in what follows we will say that we *save* $g\gamma_2-\log(2^{g\gamma_2}-1)$ *bits* each time Lemma 4 is applied. Of course, this is only a tiny, but *constant*, fraction of a bit.

Definition 5 Suppose that $G=(G_1,\ldots,G_g)$ is a grouping of some heads. Let $|G|=|G_1|+\cdots+|G_g|$ denote the number of heads involved, $||G||=|G_1|^2+\cdots+|G_g|^2$, and $l(G)=||G||(||G||-|G|)$.

The number $l(G)$ defines an order of groupings, which will be used in the inductive proof of the next lemma. Roughly, we want groupings with fewer heads or more groups to precede ones with more heads or fewer groups. The factor $||G||-|G|$ is just to make sure that the groupings whose groups are all singletons are considered first.

Lemma 5 (Great Compression) *Let y be a binary input segment. Suppose that $G=(G_1,\ldots,G_g)$ is a $(-,2s)$-grouping of some heads in y at time t for some $s\geq\phi(G)$, where $\phi(G)$ is some constant to be fixed depending only on G. (See Figure 3.) Then, if the heads not in G remain stationary for the next s steps, we can compress the portions of y scanned by the heads in G during the s steps by at least $s^{\epsilon(G)}$ bits, given the configuration of M (i.e., the state of M, the bits read by the pausing heads, and the locations of the moving heads) at time t. Here $\epsilon(G)>0$ is another constant depending on G.*

Remark. In our real situtation, the input is $\$x\#y\$$, where x,y are binary strings. In order to apply this lemma, we only need to assume that all the heads of G are in region y. Other heads, because they are pausing anyway, may be anywhere.

The key to the proof of the lemma is a *recursive* compression scheme which will be described below. The scheme is in a sense *adaptive* since where and how the compression is done

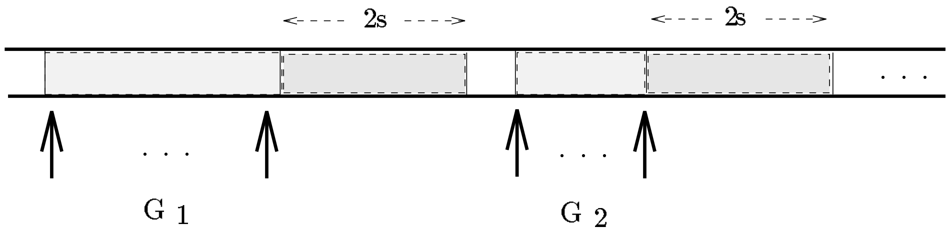

Figure 3: The initial distribution of the groups.

totally depend on the way M works. In other words, we could implement the compression (and its reverse process - decompression) by simulating M. This is essential because we want to minimize the overhead of recording the locations of the segments compressed.

We prove the lemma by induction on $l(G)$, which is bounded by a *constant* depending only on k. The constants $\phi(G)$ and $\epsilon(G)$ will be set as the induction goes. For each G, we also need to define a pair of auxiliary constants $\gamma_1(G)$ and $\gamma_2(G)$ before we use Lemma 4. It will be helpful to keep in mind that $\phi(G)$ will be increasing in $l(G)$ and $\epsilon(G)$ will be decreasing in $l(G)$.

Since the complete proof is very involved, it is described in two steps. Claim 6 establishes the basis of the induction and Claim 7 shows the inductive step. The proof of Claim 6 should give a flavor of the compression scheme, which can be very helpful before one gets into the intricate constructions in Claim 7. From now on, let's assume that the pausing heads (not in G) remain still for the next s steps.

Claim 6 *The lemma holds when* $l(G) = 0$.

Proof. $l(G) = 0$ if and only if each group contains just one head. (See Figure 4.) We prove the claim by induction on the number of heads $|G|$. Because each group contains just one head, the γ_1 in Lemma 4 is 1 here. Since M has only $|Q|$ states, the shortest strings $u_1, \ldots, u_g$ that can lead some pausing head to move have length at most $|Q|$. So we can set $\gamma_2 = |Q|$ for the γ_2 in Lemma 4.

We first prove that the claim holds when $|G| = 1$. Let the string read by M in the next s steps be w, $|w| = s$. We apply Lemma 4, with $\gamma_1 = 1$ and $\gamma_2 = |Q|$, for $s/|Q|$ times. Lemma 4 states that each of the $s/|Q|$ s-sized blocks of w belongs in a set of size at most $2^{|Q|} - 1$. Therefore, w belongs in a set of size $(2^{|Q|} - 1)^{s/|Q|}$. Thus, w can be specified in $\log(2^{|Q|} - 1)^{s/|Q|}$ bits and the compression is $s - \log(2^{|Q|} - 1)^{s/|Q|}$ bits. As discussed in the above remark, an equivalent view of this is that we save $|Q| - \log(2^{|Q|} - 1)$ bits each time Lemma 4 is applied.

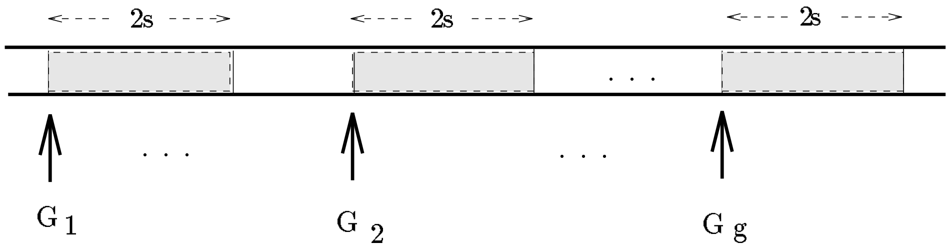

Figure 4: The heads are separated by at least $2s$ bits.

If we let

$$\phi(G) = \frac{|Q|^2}{|Q| - \log(2^{|Q|} - 1)}$$

and

$$\epsilon(G) = \frac{|Q| - \log(2^{|Q|} - 1)}{\log_{|Q|} \phi(G)} = \frac{1}{\log_{|Q|} \frac{|Q|^2}{|Q| - \log(2^{|Q|}-1)}} = \frac{1}{2 - \log_{|Q|}(|Q| - \log(2^{|Q|} - 1))},$$

then $s - \log(2^{|Q|} - 1)^{s/|Q|} \geq s^{\epsilon(G)}$ for any $s \geq \phi(G)$.

Now assume that the claim holds when $|G| = g - 1$. We prove the claim for $|G| = g$. Consider two cases. We will use a bit to indicate which case applies in the compressed string. *Case 1:* During the next s steps, some head h in G pauses for $s^{1/2}$ steps while the other $g - 1$ heads move. We record the locations of the g heads (relative to their positions at time t) at the step h begins to pause, view h as a new pausing head, and compress the segments (of y) scanned by the $g - 1$ moving heads in the next $s^{1/2}$ steps. Note that, the heads are always at least s bits away from each other during the whole process. By the induction hypothesis, we can save at least $s^{\epsilon(G')/2}$ bits on these segments, where G' is obtained from G by removing the group $\{h\}$. The heads' locations should be recorded in the self-delimiting form and would cost us at most $g(\log s + 2 \log\log s + 2) \leq 2g \log s < 2k \log s$ bits for not too small s. Thus the net saving is at least $s^{\epsilon(G')/2} - 2k \log s$. We will have to make sure that $\epsilon(G) < \epsilon(G')/2$ and that $\phi(G)$ large enough, so that

$$\phi(G)^{\epsilon(G')/2} - 2k \log \phi(G) \geq \phi(G)^{\epsilon(G)}. \tag{2}$$

Case 2: No head pauses for more than $s^{1/2}$ steps during the next s steps. We encode the segments of y scanned by the g heads in the next s steps using the following algorithm.

1. Do a compression using Lemma 4 with $\gamma_1 = 1$ and $\gamma_2 = |Q|$. (The involved strings will be appended to U as mentioned above and will be compressed later on together with others.)

2. Faithfully simulate the machine until all the g heads are out of their last compressed regions. As we shift the heads, concatenate each new input bit read to the string V.

3. Repeat step 1 until the heads have made totally s moves.

Because step 1 is done at least once in every $g|Q|s^{1/2}$ steps, step 1 is executed at least $\frac{s}{g|Q|s^{1/2}}$ times and each time we save $g|Q| - \log(2^{g|Q|} - 1)$ bits. Thus the total compression is $(g|Q| - \log(2^{g|Q|} - 1))\frac{s}{g|Q|s^{1/2}}$ bits. So, to guarantee

$$\begin{aligned}(g|Q| - \log(2^{g|Q|} - 1))\frac{s}{g|Q|s^{1/2}} &= \frac{(g|Q| - \log(2^{g|Q|} - 1))s^{1/2}}{g|Q|} \\ &\geq \frac{(k|Q| - \log(2^{k|Q|} - 1))s^{1/2}}{k|Q|} \\ &> s^{\epsilon(G)},\end{aligned}$$

we just have to make sure that $\epsilon(G) < 1/2$ and that $\phi(G)$ is large enough so that

$$\phi(G)^{0.5-\epsilon(G)} > \frac{k|Q|}{k|Q| - \log(2^{k|Q|} - 1)}. \tag{3}$$

In conclusion, we choose $\epsilon(G) = \min\{\epsilon(G')/3, 1/3\} = 1/3^{g-1}$. But for some reason to be seen later, we will choose $\phi(G)$ so large that the Inequalities 2 and 3 and the following inequality can hold:

$$\phi(G)^{\epsilon(G')/2} - 10k\log(2k\phi(G)) - 24k\log(k\phi(G)) \geq 1 \tag{4}$$

■

Now we give the inductive step of the proof of Lemma 5. Assume that the lemma holds for every G such that $l(G) < l_0$ for some $l_0 > 0$.

Claim 7 *The lemma holds for any G with $l(G) = l_0$.*

Proof. We recursively partition and compress the input. Consider three cases. Now we will have to use two bits in the compressed string to indicate these three cases.

Case 1: Some group G_i remains completely stationary for $s^{1/3}$ steps during the next s steps. We record the locations of all heads in G at the step G_i begins to pause, and use the induction hypothesis to compress the regions scanned by the $g-1$ moving groups in the next $s^{1/3}$ steps. Let G' be obtained from G by removing G_i. Then we can save at least $s^{\epsilon(G')/3}$ bits on these segments. Add the $2k\log s$ bits for recording the heads' locations and cases, the net saving is at least $s^{\epsilon(G')/3} - 2k \log s$ bits, which is at least $s^{\epsilon(G)}$ bits if $\epsilon(G) < \epsilon(G')/3$ and $\phi(G)$ is large enough so that the following holds:

$$\phi(G)^{\epsilon(G')/3} - 2k \log \phi(G) \geq \phi(G)^{\epsilon(G)}. \tag{5}$$

Case 2: Some group of G splits during the next s steps, *i.e.*, at some time (called the split point), some two adjacent heads in the group are more than $2s^{1/3}$ bits apart. We record the locations of the moving heads at the split point and re-group the moving heads with the maximum number of groups such that the heads in different groups are more than $2s^{1/3}$ bits away from each other. Note that in the above re-grouping, we split some original groups, but never join them since they are far away from each other. Let G' denote the new grouping. Clearly $l(G') < l(G)$. If we make sure that

$$\phi(G) \geq \phi(G')^3 \tag{6}$$

then $s^{1/3} \geq \phi(G)^{1/3} \geq \phi(G')$. Thus, we can use the induction hypothesis and obtain a net saving of at least

$$s^{\epsilon(G')/3} - 2|G| \log s \geq s^{\epsilon(G')/3} - 2k \log s$$

bits on the regions scanned by the heads in the $s^{1/3}$ steps after the split point. So we just have to also make sure that $\epsilon(G) < \epsilon(G')/3$ and $\phi(G)$ is large enough so that Inequality 5 holds.

Case 3: Every group moves at least once in every $s^{1/3}$ steps and no group splits during the next s steps. Clearly the no-splitting requirement means that the heads in group G_i will always be within distance $2|G_i|s^{1/3} \leq 2ks^{1/3}$ for each $i = 1, \ldots, g$. Our basic idea in this case is to dynamically (*i.e.*, as the heads shift) partition y into *sections*, each of length at most $3ks^{1/3}$, and recursively or directly do a compression on each g-tuple of sections (depending on the initial positions of the moving heads in the g-tuples) currently scanned by the g groups of heads. (So each section is associated with a group.) We try to save at least one bit (after the overhead for recording the cases and locations of the heads *etc.*) on each g-tuple of sections and compress as many g-tuples as possible. Of course, we can compress a g-tuple only if it is *disjoint* from the last compressed g-tuple. Below we describe an algorithm to select the g-tuples to compress. Call the g-tuples selected by the algorithm the *compressible g-tuples.* Define

$$\phi_0 = \max_{l(G') < l_0} \phi(G'). \tag{7}$$

The first g-tuple contains the g sections of length $3ks^{1/3}$ to the immediate right of the trailing heads in each group. We then simulate M and follow the moving heads to select the rest of g-tuples. Suppose that, at time t_0, we select a g-tuple $T = (T_1, \ldots, T_g)$, where T_i is

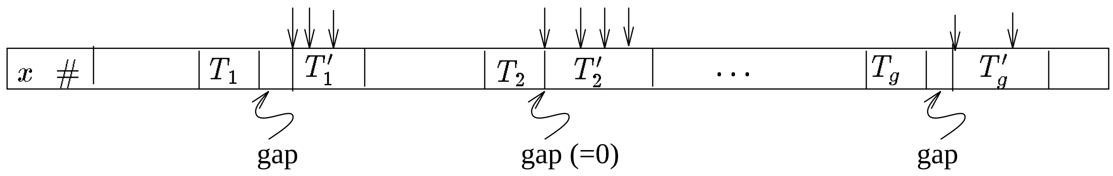

Figure 5: An A-type g-tuple.

the section associated with group G_i for $i = 1, \ldots, g$. For notational convenience, at time t_0, let $P = (P_1, \ldots, P_g)$, where P_i is the portion of y between the trailing head of G_i and the right boundary of T_i. Note that, although $P = T$ for the first g-tuple T, they are different in general (for the B-type g-tuples defined later). At time t_0, for each head h in group G_i, let d_h denote the distance from h to the right boundary of P_i (which is same as the right boundary of T_i) and $d = \sum_{h \in G} d_h$. So d represents the minimum number of moves it takes for the heads to shift out of P.

Now we need to locate the next compressible g-tuple $T' = (T_1', \ldots, T_g')$ which should be not too far from T. Intuitively, We would like to simulate M until all the heads in G shift out of P, and select our g-tuple. But if some heads pause in P for a long time, then the other heads can move very far before all heads shift out of P. To resolve this problem, when this happens, we will temporarily ignore these heads (*i.e.*, view them as pausing heads temporarily) and determine T' by considering the other (moving) heads only. So we have to consider two cases.

Case A: All heads will clear P in at most $(\phi_0 d)^4$ steps, where ϕ_0 is defined in Equation 7. We simulate M until all heads are out of P. Let $2s_1$ be the largest distance between any two heads in a same group at this moment. For each i, let T_i' be the segment of y beginning at the position of the trailing head in group G_i and ending at s_1 bits to the right of the leading head in G_i. T' will be called an *A-type* g-tuple. The portions of y between the right boundary of P and the left boundary of T' will be called a *gap*. Figure 5 shows an example.

Case B: Some heads will remain in P after $(\phi_0 d)^4$ steps. Then it is easy to verify that there must be a time t_1 such that (i) $(\phi_0 d)^2 \le t_1 < (\phi_0 d)^4$, (ii) the heads that are still in P at the time t_1 will remain completely stationary during the next $t_1^{1/2}$ steps. Simulate M to time t_1. Consider the heads still in P as pausing during the next $t_1^{1/2}$ steps. We define T_i' to be the portion of y beginning at the right boundary of T_i and ending at $t_1^{1/2}$ bits to the right of the leading (moving) head in G_i, for each i. T' will be called a *B-type* g-tuple. Figure 6 shows an example.

Remark. Before we proceed, we clarify several points here. For each A-type g-tuple T, $P = T$. For a B-type g-tuple, the trailing heads might fall far behind (not more than

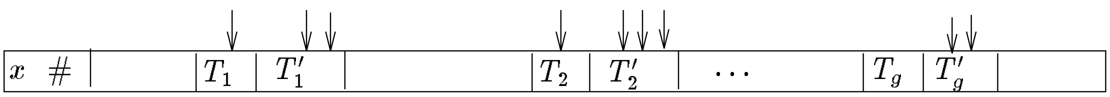

Figure 6: A B-type g-tuple.

$2ks^{1/3}$ bits from their respective leading heads of course). So in this case, a B-type g-tuple $T = (T_1, \ldots, T_g)$ is properly contained in $P = (P_1, \ldots, P_g)$, *i.e.*, T_i is a proper subsection of P_i. In fact, such situation can happen for several consecutive B-type tuples, and it may be the case that a P contains several consecutive B-type g-tuples. One might worry that because d might be as large as $O(s^{1/3})$, $(\phi_0 d)^4$ could be greater than s. This is no problem since if this happens we just stop. Later we will argue that this way we can identify a sufficient number of g-tuples to compress. The important point here is that $(\phi_0 d)^4$ is a polynomial function in the size of the previously compressed g-tuple(s). Hence the overhead of recording the relative position of the next g-tuple, which is logarithmic of in d, can be charged to the saving on the previously compressed g-tuple(s), which is polynomial in d.

Now let us see how we can compress the compressible g-tuples to achieve the desired saving. Again, there are two cases. For convenience, we will refer to the first time that the involved moving heads have all entered a g-tuple as the *start* of the g-tuple. (Note that some heads in G may be temporarily pausing at the moment and will not be in the g-tuple at its start.)

Subcase 3.1: At least half of the compressible g-tuples are B-type g-tuples or A-type g-tuples with some split group (*i.e.*, some two adjacent heads in the group are more than $2\phi_0$ bits apart) at the start of the A-type g-tuple. We will compress the g-tuples from left to right. Again, suppose we have just compressed a g-tuple T and let P and d be defined as above (corresponding to T). We record the locations of all heads (including those temporarily pausing ones in Case B) at the start of the next compressible g-tuple T' (relative to their locations at the start of T). The compression is done according to three cases.

- If T' is of B-type, we recursively compress T' for the next $t_1^{1/2}$ steps. Let G' be obtained from G by removing the temporary pausing heads. By the induction hypothesis, we can save at least $t_1^{\epsilon(G')/2}$ bits on the g-tuple. The $2k \log t_1$ bits for recording the heads' locations will be charged to T' itself. We will make sure that the compressed amount is sufficient to cover the overhead in this case.

- If T' is of A-type with some split group at the start, we first concatenate the bits in the gap to V, and split the heads into the largest number of groups so that the groups are at least $2s_1$ bits apart, and recursively compress the g-tuple for s_1 steps to save

$s_1^{\epsilon(G')/2}$ bits, where G' is the new grouping. In this case, the heads' (relative) locations cost $8k\log(\phi_0 d)$ bits, since they are at most $(\phi_0 d)^4$ bits from T. Note that, in general here we cannot claim

$$s_1^{\epsilon(G')/2} > 8k\log(\phi_0 d),$$

since s_1 might be exponentially small compared to d. The trick is to charge this overhead to all the compressible g-tuples contained (or partially contained) in region P. Since P has size (*i.e.*, the total length of its sections) at least d/k, the compression achieved in this region should be large enough to compensate the overhead $8k\log(\phi_0 d)$.

The overhead is charged to the g-tuples (in region P) according to their sizes. So a g-tuple of size r is charged $8kr\log(\phi_0 d)/\sum_i |P_i|$ bits. Since $d < k\sum_i |P_i|$ and $r \leq \sum_i |P_i|$,

$$8kr\log(\phi_0 d)/\sum_i |P_i| < 8kr\log(\phi_0 k\sum_i |P_i|)/\sum_i |P_i| \leq 8k\log(k\phi_0 r). \tag{8}$$

Each g-tuple gets at most one such charge, because once we encounter an A-type g-tuple and charge the overhead to the region P, all heads would have already moved out of P by the start of the A-type g-tuple. Thus, the compressible g-tuples in this P region will not belong to any other subsequent P regions and hence will not be charged again. Note that here we do not need bits to indicate the cases since the locations of the heads can tell the correct case.

- If T' is of A-type without any split groups, we will not try to compress the g-tuple and simply concatenate the bits in T' to string V in the order they are read the first time. But this compressible g-tuple T' still gets charged of the overhead as the others. We can "shift" the charge to some truly compressed g-tuple. Clearly in this case the size of T' is less than $2k\phi_0$.

Since at least half of the compressible g-tuples will be compressed, we claim that a compressed g-tuple T' of size r gets charged of the location overheads at most three times totally in the above:

- once by itself, costing at most $2k\log t_1$ bits;
- once by some subsequent (compressed or uncompressed) A-type g-tuple, costing at most $8k\log(k\phi_0 r)$ bits; and
- once due to the shifting of charge from some uncompressed A-type g-tuple (of size less than $2k\phi_0$), costing at most $8k\log(2k^2\phi_0^2)$ bits.

Note that in the above compression of T', the two parameters t_1 and s_1 have the property: $t_1 \le r \le t_1 + t_1^{1/2} \le 2t_1$ and $r < ks_1$. Hence, the compressed g-tuple T' is charged of at most

$$2k \log r + 8k \log(k\phi_0 r) + 8k \log(2k^2\phi_0^2) < 10k \log(2r) + 24k \log(k\phi_0)$$

bits of overhead. Putting everything together, we have a net saving of at least

$$(r/k)^{\epsilon(G')/2} - 10k \log(2r) - 24k \log(k\phi_0)$$

on T', where G' is a grouping with $l(G') < l_0$ and $r/k \ge \phi_0$, which is at least 1 by Inequality 4.

So now the question is how many g-tuples we will compress. Since, by the initial assumption of Case 3, each group moves at least once every $s^{1/3}$ steps and each section of a g-tuple has size at most $3ks^{1/3}$, there should be at least $s^{1/3}/(3k)$ compressible g-tuples in the worst case. Thus we should be able to compress at least $s^{1/3}/(6k)$ g-tuples and save $s^{1/3}/(6k)$ bits, which is at least $s^{\epsilon(G)}$ if $\epsilon(G) < 1/3$ and and $\phi(G)$ is not too small so that

$$\phi(G)^{(1-3\epsilon(G))/3} \ge 6k. \tag{9}$$

Subcase 3.2: At least half of the compressible g-tuples are of A-type without any split group at the start. Hence, the heads in each group of G must be within distance $2k\phi_0$ at the start of these *majority* g-tuples. We now compress y as follows: faithfully simulate M and concatenate each new bit read to string V until all the heads shift out of the last compressed region and all the heads in each group are within distance $2k\phi_0$. (Notice that we now do not need to record heads' positions since they can be recovered by simulating M on the bits in V.) Then we apply Lemma 4 (with $\gamma_1 = 2k\phi_0$) to make a compression. Let $\gamma_1(G) = 2k\phi_0$ and $\gamma_2(G)$ be the corresponding constant. Then the compression saves at least $g\gamma_2(G) - \log(2^{g\gamma_2(G)} - 1) \ge k\gamma_2(G) - \log(2^{k\gamma_2(G)} - 1)$ bits. By the above discussion, we should be able to make at least $s^{1/3}/(6k\gamma_2(G))$ such compressions because of the existence of $s^{1/3}/(6k)$ non-split A-type compressible g-tuples. (Compressing such a g-tuple can disqualify at most $\gamma_2(G)$ other such g-tuples.) So here we just have to make $\phi(G)$ large enough and $\epsilon(G)$ small enough so that

$$(k\gamma_2(G) - \log(2^{k\gamma_2(G)} - 1))\frac{\phi(G)^{1/3}}{6k\gamma_2(G)} \ge \phi(G)^{\epsilon(G)} \tag{10}$$

will hold.

Now we fix the constants $\phi(G)$ and $\epsilon(G)$. From the above, we know that

$$\epsilon(G) < \min_{l(G')<l_0} \{1/3, \epsilon(G')/3\} = \min_{l(G')<l_0} \epsilon(G')/3$$

So we can just choose $\epsilon(G) = \min_{l(G')<l_0} \epsilon(G')/4$. Then we should choose a large $\phi(G)$ such that Inequalities 4,5,6,9,10 all hold for all G', $l(G') < l_0$. This completes the inductive step and thus the proof of Lemma 5. ■

Remark. After seeing the proof, one might wonder why we need define B-type g-tuples in the above. For example, why not consider A-type g-tuples only by waiting until all heads have cleared P? This way we still roughly have $s^{1/3}/(6k)$ such A-type g-tuples, and we can consider two cases depending on whether more than half of them split or not (similar to Subcases 3.1 and 3.2). Such an approach fails since in the case that there are more than half splitting A-type g-tuples, we will have to record the (relative) heads' positions of each g-tuple and then do a recursive compression. But the savings from such compressions may not be enough to compensate the overhead of recording the heads' positions. Each such overhead may be as large as $\Omega(\log s)$ bits, which can be much larger than the saving achieved in the corresponding g-tuple which can be as small as 1 bit. The B-type g-tuples were introduced to solve this problem. Now we know for sure that the position overhead for an A-type g-tuple is logarithmic in the size of the previously compressed B-type g-tuple(s) and can thus be compensated by the saving on these B-type g-tuple(s).

The following corollary is obvious. Let grouping $G_M = (H_M)$ with a single group, and define $\phi_M = \phi(G_M)$ and $\epsilon_M = \epsilon(G_M)$.

Corollary 8 *Let y be a binary input segment. Suppose that some heads move s steps, for some $s \geq \phi_M$, while the others stay stationary starting from some time t, then given the configuration of M at time t, we can compress the segments of y scanned by the moving heads during the s steps by at least s^{ϵ_M} bits.*

4.3 The Final Proof

Now we are ready to prove the Moving Lemma, using the above compression lemmas.

Proof of the Moving Lemma. Let α, β be arbitrary constants, $0 < \alpha \leq 1$. Starting at some time t, suppose that some heads move $|y|^\alpha$ steps (before reaching \$) while the other heads remain completely stationary. Then we can encode y as follows.

- Append the prefix of y to the left of the trailing moving head at time t to an initially empty string V.
- Record the current configuration of M, *i.e.*, the state of M, bits read by the pausing heads, and locations of the moving heads relative to #, in the self-delimiting form, and append them to V. This should take at most $2\log|Q| + 2k\log|y|$ bits.
- Compress the portions of y read by the moving heads during the next $|y|^\alpha$ steps using the compression algorithm given in the proof of Lemma great, and store the compressed string in U'. Note that, we should know the boundaries of the compressed segments after performing the actual compression. By Corollary 8, we can save at least $|y|^{\alpha\epsilon_M}$ bits here.

- Append the rest of the uncompressed segments to V.
- Finally, let $W = \overline{U'}V$ as the final encoding of y.

It is easy to see that from W we can reconstruct y by reversing the compression algorithm. The self-delimiting form of U' would cost us at most $2\log|U'| \leq 2\alpha\log|y|$ bits. Choose a large φ so that

$$\varphi^{\alpha} \geq \phi_M$$

and

$$\varphi^{\alpha\epsilon_M} > 2\log|Q| + 2(k+\alpha)\log\varphi + \beta\log\varphi + C$$

hold, where C is the total number of bits required to describe M and the above compression algorithm in the self-delimiting form. Then, since $|y| > \varphi$, the above encoding has a net saving of at least

$$|y|^{\alpha\epsilon_M} - 2\log|Q| - 2(k+\alpha)\log|y| - C > \beta\log|y|$$

bits, which contradicts the assumption that $K(y) \geq |y| - \beta\log|y|$. ∎

5 Concluding Remarks

The referee of the paper raised the question of other variants of the string-matching problem. For example, can a k-DFA with non-sensing heads check if x is a substring of y with input of form $\$y\#x\$$ (rather than $\$x\#y\$$)? Our results imply that this is also not possible by a simple reduction as follows. If we were given a k-DFA M which does string-matching with input $\$y\#x\$$, we can construct a $2k$-DFA M' doing string-matching with input $\$x\#y\$$: On input $x\#y$, the $2k$-DFA M' first moves k of its $2k$ heads to the $\#$ sign passing x, and then M' starts to mimic the behavior of the k-DFA M. Each time a head reaches the end of y, we activate a head waiting at the beginning of x to continue the simulation of M. The existence of such an M' contradicts Theorem 3.

Another variant proposed by the referee is to have the pattern x on one tape with k one-way heads and the text y on another tape with another k one-way heads. Suppose that such a machine can do string-matching. Then, similar to above, one can construct a $2k$-DFA with all $2k$ heads on one tape, doing the same job and contradicting Theorem 3.

It remains open whether k one-way *sensing* heads can do string-matching. For $k = 2, 3$, it turns out that the impossibility results in [9, 7] also hold for sensing heads. We believe that the answer is also in the negative for all constant k. However, the Moving Lemma fails now. It is easy to program a 7-DFA so that on a random input y, three heads can move

$3|y| - O(\log|y|)$ steps consciously without "pulling" the other heads. (First, use four heads to compute $\log|y|$ so that one head will be $\log|y|$ bits away from the left endmarker. Then it uses three heads to look for a run of $\log|y|$ 0's and if it finds such a run, the 7-th head will start to move. The three heads will make at least a total of $3|y| - O(\log|y|)$ moves before they can find such a run since y is random.) But we hope that some of the ideas in this paper can still be useful in dealing with the sensing heads.

6 Acknowledgments

The authors wish to thank Yaacov Yesha and Mihály Geréb-Graus for early cooperation on this work, which produced the solutions for $k = 2$ and $k = 3$ and some useful tools for this final solution, and Tom Jurdzinski for several corrections. We also thank the anonymous referee for many helpful comments and insisting on seeing a clear presentation of the proofs. This conjecture was first brought to the attention of the second author by Zvi Galil at Cornell around 1983/1984.

References

[1] A.V. Aho. Algorithms for finding patterns in strings. In *Handbook of Theoretical Computer Science*, Jan van Leeuwen, ed. (1990) 255-300.

[2] R.S. Boyer and J.S. Moore. A fast string searching algorithm, *CACM* 20:10 (1977) 762-772.

[3] S. Cook. Linear time simulation of deterministic two-way pushdown automata. *Proc. IFIP Congress 71*, TA-2, North-Holland, Amsterdam (1971), 172-179.

[4] M. Chrobak and W. Rytter. Remarks on string-matching and one-way multihead automata. *Inform. Process. Lett.*, 24 (1987) 325-329.

[5] Z. Galil. Open Problems in Stringology. In *Combinatorial Algorithms in Words.* A. Apostolico and Z. Galil eds., Springer Verlag Lecture Notes (1985) 1-8.

[6] Z. Galil and J. Seiferas. Time-space-optimal string-matching. *Proceedings 13th ACM Symp. Theory of Computing*, (1981) 106-113. *J. Comput. Syst. Sci.*, 26 (1983) 280-294.

[7] M. Geréb-Graus and M. Li. Three one-way heads cannot do string-matching. *J. Comput. Syst. Sci.*, 48:1 (1994) 1-8.

[8] D.E. Knuth, J.H. Morris, Jr., and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Computing* 6:2 (1977) 323-350.

[9] M. Li and Y. Yesha. String-matching cannot be done by a two-head one-way deterministic finite automaton. *Inform. Process. Lett.*, 22 (1986) 231-235.

[10] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Springer-Verlag, 1993.

[11] W. Paul. Kolmogorov complexity and lower bounds. *2nd International conference on fundamentals of computation theory* (1979).

[12] W. Paul, J. Seiferas, and J. Simon. An information theoretic approach to time bounds for on-line computations. *J. Comput. Syst. Sci.*, 23 (1981) 108-126.