2016 IEEE 57th Annual Symposium on Foundations of Computer Science

Which Regular Expression Patterns are Hard to Match?

Arturs Backurs
EECS
MIT
Cambridge, MA, USA
backurs@mit.edu

Piotr Indyk
EECS
MIT
Cambridge, MA, USA
indyk@mit.edu

***Abstract*—Regular expressions constitute a fundamental notion in formal language theory and are frequently used in computer science to define search patterns. In particular, regular expression matching and membership testing are widely used computational primitives, employed in many programming languages and text processing utilities. A classic algorithm for these problems constructs and simulates a non-deterministic finite automaton corresponding to the expression, resulting in an $O(mn)$ running time (where m is the length of the pattern and n is the length of the text). This running time can be improved slightly (by a polylogarithmic factor), but no significantly faster solutions are known. At the same time, much faster algorithms exist for various special cases of regular expressions, including dictionary matching, wildcard matching, subset matching, word break problem etc.**

In this paper, we show that the complexity of regular expression matching can be characterized based on its *depth* (when interpreted as a formula). Our results hold for expressions involving concatenation, OR, Kleene star and Kleene plus. For regular expressions of depth two (involving any combination of the above operators), we show the following dichotomy: matching and membership testing can be solved in near-linear time, except for "concatenations of stars", which cannot be solved in strongly sub-quadratic time assuming the Strong Exponential Time Hypothesis (SETH). For regular expressions of depth three the picture is more complex. Nevertheless, we show that all problems can either be solved in strongly sub-quadratic time, or cannot be solved in strongly sub-quadratic time assuming SETH.

An intriguing special case of membership testing involves regular expressions of the form "a star of an OR of concatenations", e.g., $[a|ab|bc]^*$. This corresponds to the so-called *word break* problem, for which a dynamic programming algorithm with a runtime of (roughly) $O(n\sqrt{m})$ is known. We show that the latter bound is not tight and improve the runtime to $O(nm^{0.44\cdots})$.

***Keywords*-Regular expressions; conditional hardness; SETH; classification**

I. Introduction

A regular expression (regexp) is a formula that describes a set of words over some alphabet Σ. It consists of individual symbols from Σ, as well as operators such as *OR* "|" (an alternative between several pattern arguments), *Kleene star* "$*$" (which allows 0 or more repetitions of the pattern argument), *Kleene plus* "$+$" (which allows 1 or more repetitions of the pattern argument), wildcard "." (which matches an arbitrary symbol), etc. For example, $[a|b]^+$ describes any sequence of symbols a and b of length at least 1. See Preliminaries for the formal definition.

In addition to being a fundamental notion in formal language theory, regular expressions are widely used in computer science to define search patterns. Formally, given a regular expression (pattern) p of size m and a sequence of symbols (text) t of length n, the goal of regular expression matching is to check whether a substring of t can be derived from p. A closely related problem is that of membership testing where the goal is to check whether the text t itself can be derived from p. Regular expression matching and membership testing are widely used computational primitives, employed in several programming languages and text processing utilities such as Perl, Python, JavaScript, Ruby, AWK, Tcl and Google RE2. Apart from text processing and programming languages, regular expressions are used in computer networks [19], databases and data mining [13], computational biology [22], human-computer interaction [17] etc.

A classic algorithm for both problems constructs and simulates a non-deterministic finite automaton corresponding to the expression, resulting in the "rectangular" $O(mn)$ running time. A sequence of improvements, first by Myers [21] and then by [6], led to an algorithm that achieves roughly $O(mn/\log^{1.5} n)$ running time. The latter result constitutes the fastest algorithm for this problem known to date, despite an extensive amount of research devoted to this topic. The existence of faster algorithms is a well-known open problem ([12], Problem 4).

However, significantly faster algorithms are known for various well-studied special cases of regular expressions. For example:

1) If the pattern is a concatenation of symbols (i.e., we search for a specific sequence of symbols in the text), the pattern matching problem corresponds to the "standard" string matching problem and can be solved in linear time, e.g., using the Knuth-Morris-Pratt algorithm [18].
2) If the pattern is of the form $p_1|p_2|\ldots|p_k$ where p_i are sequences of symbols, then the pattern matching problem corresponds to the *dictionary matching* prob-

0272-5428/16 $31.00 © 2016 IEEE
DOI 10.1109/FOCS.2016.56

lem that can be solved in linear time using the Aho-Corasick algorithm [3].

3) If the pattern is a concatenation of symbols and single character wildcards "." matching any symbol, the pattern matching problem is known as the *wildcard matching* and can solved in (deterministic) $O(n \log m)$ time using convolutions [10], [11], [15], [16].
4) More generally, if the pattern is a concatenation of *single* character ORs of the form $s_1|s_2|\dots|s_k$ for $s_1, \dots, s_k \in \Sigma$ (e.g., $[a|b][a|c][b|c]$), the pattern matching problem is known as *superset matching* and can be solved in (deterministic) $O(n \log^2 m)$ time [9], [10].
5) Finally, if the goal is to test whether a text t can be derived from a pattern p of the form $(p_1|p_2|\dots|p_k)^+$ where p_i are sequences of symbols, the problem is known as the *word break* problem. It is a popular interview question [20], [23], and the known solutions can be implemented to run in $\sqrt{m}n \log^{O(1)} n$ time.

The first two examples were already mentioned in [12] as a possible reason why a faster algorithm for the general problem might be possible. Despite the existence of such examples, any super-polylogarithmic improvements to the algorithms of [6], [21] in the general case remain elusive. Furthermore, we are not aware of any systematic classification of regular expressions into "easy" and "hard" cases for the pattern matching and membership testing problems. The goal of this paper is to address this gap.

Results: Our main result is a classification of the computational complexity of regular expression matching and membership checking for patterns that involve operators "|", "+", "$*$" and concatenation "$\cdot$", based on the pattern *depth*. Our classification enables us to distinguish between the cases that are solvable in sub-quadratic time (including the five problems listed above) and the cases that do not have strongly sub-quadratic time algorithms under natural complexity theoretic assumptions, such as Strong Exponential Time Hypothesis [14] and Orthogonal Vectors conjecture [8], [25]. Our results therefore demonstrate a non-trivial dichotomy for the complexity of these problems.

To formulate our results, we consider pattern formulas that are *homogeneous*, i.e., in which the operators at the same level of the formula are equal (note that the five aforementioned problems involve patterns that satisfy this condition). We say that a homogeneous formula of depth k has *type* $o_1 o_2 \dots o_k$ if for all levels i the operators at level i are equal to o_i (note that, in addition to the operators, a level might also contain leaves, i.e. symbols; for example the expression $[a|b]a[b|c]$ is a depth-2 formula of type "$\cdot|$"). We will assume that no two consecutive operators in the type descriptor are equal, as otherwise they can be collapsed into one operator.

Our results are described in Table I (for depth-2 expressions) and Table II (for depth-3 expressions). The main findings can be summarized as follows:

1) Almost all pattern matching and membership problems involving depth-2 expressions can be solved in near-linear time. The lone exception involve patterns of type "$\cdot*$", for which we show that matching and membership problems cannot be solved in time $O((mn)^{1-\delta})$ for any constant $\delta > 0$ and $m \leq n$ assuming the Strong Exponential Time Hypothesis (SETH) [14]. Interestingly, we show that pattern matching with a very similar depth-2 type, namely "$\cdot+$", can be solved in $O(n \log^2 m)$ time.
2) Pattern matching problems with depth-2 expressions contain a "high density" of interesting algorithmic problems, with non-trivial algorithms existing for types "$\cdot+$" (this paper), "$\cdot|$" [10] , "$|\cdot$" [3] and "$+\cdot$" (essentially solved in [18], since $+$ can be dropped). In contrast, membership problems with depth-2 expressions have a very restrictive structure that makes them mostly trivially solvable in linear time, with the aforementioned exception for the "$\cdot*$" type
3) Pattern matching problems with depth-3 expressions have a more diversified structure. All types starting with $\cdot$ are SETH-hard; all types starting with $|$ are either-SETH hard (if followed by $\cdot$) or easily solvable in linear time; all types starting from $*$ are trivially solvable in linear time (since $*$ allows zero repetitions); all types starting from $+$ inherit their complexity from the last two operators in the type description (since $+$ allows exactly one repetition).
4) Finally, membership checking with depth-3 expressions presents the most complex picture. As before, all types starting with $\cdot$ are SETH-hard. On the other hand, types starting with $|$ (except for $|\cdot*$) have linear time algorithms, with difficulty levels that range from trivial observations to undergraduate-level exercises[1]. Types starting with $*$ or $+$ include "$*|\cdot$" and "$+|\cdot$", which correspond to the aforementioned word break problem [20], [23]. This is the only problem in the table whose (conditional) complexity is not determined up to logarithmic factors. However, we show that the running time of the standard dynamic-programming based algorithm can be improved, from roughly $nm^{0.5}$ to roughly $nm^{0.5 - 1/18}$.

Our techniques: Our upper bounds for depth-2 expressions follow either from known near-linear time algorithms for specific variants of regular expressions, or relatively simple constructions of such algorithms. In particular, we observe that type "$\cdot|$" expressions (concatenations of ORs) correspond to superset matching, type "$|\cdot$" expressions (OR of sequences) correspond to dictionary matching and type "$+\cdot$" reduces to "$\cdot$" and thus corresponds to the standard

[1]Incidentally, the second author used some of the problems from Table II in an undergraduate algorithms course.

Type	Example	Pattern matching	Membership
$\cdot+$	a^+ab^+	$O(n\log^2 m)$ (full version)	$O(n+m)$ (immediate)
$\cdot*$	a^*ab^*	$\Omega((mn)^{1-\alpha})$ (Section III-C)	$\Omega((mn)^{1-\alpha})$ (full version)
$\cdot\|$	$[a\|b][b\|c]$	$O(n\log^2 m)$ (**superset matching** [10])	$O(n+m)$ (immediate)
$\|\cdot$	$ab\|c$	$O(n+m)$ (**dictionary matching** [3])	$O(n+m)$ (immediate)
$\|*$	$a^*\|a\|b^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$\|+$	$a^+\|a\|b^+$	$O(n+m)$ (immediate - reducible to "|")	$O(n+m)$ (immediate)
$*\cdot$	$[ab]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+$	$[a^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate - reducible to "+")
$*\|$	$[a\|b]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$+\cdot$	$[ab]^+$	$O(n+m)$ (**string matching** [18])	$O(n+m)$ (immediate - equivalent to "$*\cdot$")
$+\|$	$[a\|b]^+$	$O(n+m)$ (immediate - reducible to "|")	$O(n+m)$ (immediate - equivalent to "$*\|$")
$+*$	$[a^*]^+$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate - reducible to "+")

Table I: Classification of the complexity of the pattern matching and the membership test problems for depth-2 expressions. All lower bounds assume SETH and $m \leq n$. Some upper bounds use randomization (notably hashing). For an explanation of reducibility see "Our techniques".

pattern matching problem. Furthermore, we give a near-linear time algorithm for pattern matching with type "$\cdot+$" expressions, where patterns are concatenations of expressions of the form $s^{\geq k}$ or s^k, where $s^{\geq k}$ denotes a sequence of symbols s repeated at least $k \geq 1$ times.[2] We show that this problem can be solved in near-linear time by reducing it to one instance of subset matching and one instance of wildcard matching. All other problems can be solved in linear time, with the exception of type "$\cdot*$". The latter expressions correspond to patterns obtained by concatenating patterns of the form $s^{\geq k}$ and s^k. Unlike in the "$\cdot+$" case, however, here we cannot assume that $k \geq 1$, since each symbol could be repeated zero times. We show that this simple change makes the problem SETH-hard. This is accomplished by a reduction from an intermediate problem, namely the (unbalanced version of the) Orthogonal Vectors Problem (OVP) [8], [25]. The problem is defined as follows: given two sets $A, B \subseteq \{0,1\}^d$ such that $|A| = M$ and $|B| = N$, determine whether there exists $x \in A$ and $y \in B$ such that the dot product $x \cdot y = \sum_{j=1}^{d} x_j y_j$ is equal to 0.[3]

Our results for depth-3 expression pattern matching are multi-fold. First, all types starting from $*$ are trivially solvable in linear time, since $*$ allows zero repetitions. Second, all types starting from $+$ inherit their complexity from the last two operators in the type description, since $+$ allows exactly one repetition. Third, all types starting from $|*$ or $|+$ have simple linear time solutions.

The remaining cases lead to SETH-hard problems. For six types this follows immediately from the analogous result for type "$\cdot*$". For the six remaining types the hardness is shown via individual reductions from OVP. For some types, such a reduction is immediate. For example, for type "$|\cdot|$" expressions (ORs of concatenations of ORs), we form the text by concatenating all vectors in B (separated by some special symbol), and we form the pattern by taking an OR of the vectors in A, modified by replacing 0 with $[0|1]$ and 1 with 0. A similar approach works for type "$|\cdot+$" expressions.

The remaining four types are grouped into two classes: "$\cdot+\cdot$" is grouped with "$\cdot|\cdot$" and "$\cdot+|$" is grouped with "$\cdot|+$". For each group, we first show hardness of the first type in the group (i.e., of "$\cdot+\cdot$" and "$\cdot+|$", respectively). We then show that the second type in each group is hard by making changes to the hardness proof for the first type.

The hardness proof for "$\cdot+\cdot$" proceeds as follows. We form the pattern p by concatenating (appropriately separated) pattern vector gadgets for each vector in A, and form the text t by concatenating (appropriately separated) text vector gadgets for each vector in B. We then show that if there is a pair of orthogonal vectors $a^i \in A, b^j \in B$ then p can be matched to a substring of t, and vice versa. To show this, we construct p and t so that any pair of gadgets (in particular the gadgets for a^i and b^j) can be aligned. We then show that (i) each vector gadget for a vector in A can be matched with "most" of the gadget for the corresponding $b \in B$ (ii) matching the gadgets for *orthogonal* vectors a^i and b^j allows us to make a "smaller step", i.e., to match the gadget for a^i with a smaller part of the gadget for b^j, and (iii) at least one "smaller step" is necessary to completely derive a substring of t from p. We then conclude that there is a pair of orthogonal vectors $a^i \in A, b^j \in B$ if and only if p can be matched to a substring of t. The hardness proof for "$\cdot+|$" follows a similar general approach, although the technical development is different. In particular, we construct the gadgets such that the existence of orthogonal vectors makes it possible to make a "bigger step", i.e., to derive a

[2]For example, the expression aa^+bc^+ generates all words of the form $a^{\geq 2}b^1c^{\geq 1}$.

[3]The reduction is somewhat complex, so we will not outline it here. However, we give an overview of other reductions from OVP in the next few paragraphs.

Type	Example	Pattern matching	Membership
$\cdot\|\cdot$	$[a\|bb][ba\|b]$	$\Omega((mn)^{1-\alpha})$ (full version)	$\Omega((mn)^{1-\alpha})$ (full version)
$\cdot\|*$	$[a^*\|b^*][c^*\|b]$	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")
$\cdot\|+$	$[a^+\|b^+][c^+\|b]$	$\Omega((mn)^{1-\alpha})$ (full version)	$\Omega((mn)^{1-\alpha})$ (full version)
$\cdot+\cdot$	$[ab]^+[bca]^+$	$\Omega((mn)^{1-\alpha})$ (full version)	$\Omega((mn)^{1-\alpha})$ (full version)
$\cdot+\|$	$[a\|b]^+[a\|c\|d]^+$	$\Omega((mn)^{1-\alpha})$ (full version)	$\Omega((mn)^{1-\alpha})$ (full version)
$\cdot+*$	$[a][a^+]^*[b^+]$	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")
$\cdot*\cdot$	$[ab]^*[bca]^*$	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")
$\cdot*\|$	$[a\|b]^*[a\|b\|c]^*$	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")
$\cdot*+$	$[a^*]b[b^+]^*$	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")
$\|\cdot\|$	$[(a\|b)(b\|c)]\|[(a\|c)b]$	$\Omega((mn)^{1-\alpha})$ (Section III-A)	$O(n+m)$ (immediate)
$\|\cdot*$	$[a^*b^*]\|[b^*c^*]$	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")
$\|\cdot+$	$[a^+b^+]\|[b^+c^+]$	$\Omega((mn)^{1-\alpha})$ (Section III-B)	$O(n+m)$ (full version)
$\|*\cdot$	$[abc]^*\|[bc]^*$	$O(n+m)$ (immediate - reducible to "$\|$")	$O(n+m)$ (full version)
$\|*\|$	$[a\|b\|c]^*\|[b\|c]^*$	$O(n+m)$ (immediate - reducible to "$\|$")	$O(n+m)$ (immediate)
$\|*+$	$[a^+]^*\|[b^+]^*$	$O(n+m)$ (immediate - reducible to "$\|$")	$O(n+m)$ (immediate)
$\|+\cdot$	$[abc]^+\|[bc]^+$	$O(n+m)$ (reducible to "$\|\cdot$")	$O(n+m)$ (full version)
$\|+\|$	$[a\|b\|c]^+\|[b\|c]^+$	$O(n+m)$ (immediate - reducible to "$\|$")	$O(n+m)$ (same as "$\|*\|$")
$\|+*$	$[a^*]^+\|[b^*]^+$	$O(n+m)$ (immediate - reducible to "$\|$")	$O(n+m)$ (immediate)
$*\cdot\|$	$[[a\|b][b\|c]]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*\cdot*$	$[a^*b^*c^*]^*$	$O(n+m)$ (immediate)	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")
$*\cdot+$	$[a^+b^+c^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (full version)
$*\|\cdot$	$[a\|ab\|bc]^*$	$O(n+m)$ (immediate)	$O(nm^{0.44\ldots})$ (**word break** - full version)
$*\|*$	$[a^*\|b^*\|c^*]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*\|+$	$[a^+\|b^+\|c^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+\cdot$	$[[abcd]^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+\|$	$[[a\|b\|c\|d]^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+*$	$[[a^*]^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$+\cdot\|$	$[[a\|b][b\|c]]^+$	$O(n\log^2 m)$ (reducible to "$\cdot\|$")	$O(n+m)$ (same as "$*\cdot\|$")
$+\cdot*$	$[a^*b^*c^*]^+$	$\Omega((mn)^{1-\alpha})$ (from "$\cdot*$")	$\Omega((mn)^{1-\alpha})$ (same as "$*\cdot*$")
$+\cdot+$	$[a^+b^+c^+]^+$	$O(n\log^2 m)$ (reducible to "$\cdot+$")	$O(n+m)$ (same as "$*\cdot+$")
$+\|\cdot$	$[a\|ab\|bc]^+$	$O(n+m)$ (reducible to "$\|\cdot$")	$O(nm^{0.44\ldots})$ (**word break** - full version)
$+\|*$	$[a^*\|b^*\|c^*]^+$	$O(n+m)$ (reducible to "$\|*$")	$O(n+m)$ (same as "$*\|*$")
$+\|+$	$[a^+\|b^+\|c^+]^+$	$O(n+m)$ (reducible to "$\|+$")	$O(n+m)$ (same as "$*\|+$")
$+*\cdot$	$[[abcd]^*]^+$	$O(n+m)$ (reducible to "$*\cdot$")	$O(n+m)$ (same as "$*\cdot$")
$+*\|$	$[[a\|b\|c\|d]^*]^+$	$O(n+m)$ (reducible to "$*\|$")	$O(n+m)$ (same as "$*\|$")
$+*+$	$[[a^+]^*]^+$	$O(n+m)$ (reducible to "$*+$")	$O(n+m)$ (same as "$*+$")

Table II: Classification of the complexity of the pattern matching and the membership test problems for depth-3 expressions. See Figure 1 for the visualization of the table.

bigger part of t, and that one bigger step is necessary to complete the derivation.

To show hardness of the second type in each group, we adapt the arguments for the first type in the group. In particular, to show hardness for type "$\cdot|\cdot$" , we construct p and t as in the reduction for type "$\cdot+\cdot$" and then transform p into a type "$\cdot|\cdot$" regular expression p'. The transformation has the property that p' is *less* expressive than p (i.e., the language corresponding to p is a *superset* of the language corresponding to p'), but the specific substrings of the text t needed for the reduction can be still derived from p'. The hardness proof for "$\cdot|+$" is obtained via a similar transformation of the hardness proof for "$\cdot+|$" .

Finally, consider the membership checking problem for depth-3 expressions. As before, all types starting with $\cdot$ are shown to be SETH-hard. The reductions are similar to those for the pattern matching problem, but in a few cases require some modifications. On the other hand, types starting with $|$ (with the exception of $|\cdot*$) have linear time algorithms. The algorithms are not difficult, but require the use of basic algorithmic notions, such as periodicity (for types "$|*\cdot$" and "$|+\cdot$") and run-length encoding (for type "$|\cdot+$"). Types starting with $*$ are mostly solvable in linear time, with two exceptions: type "$*\cdot*$" inherits the hardness from "$\cdot*$", while

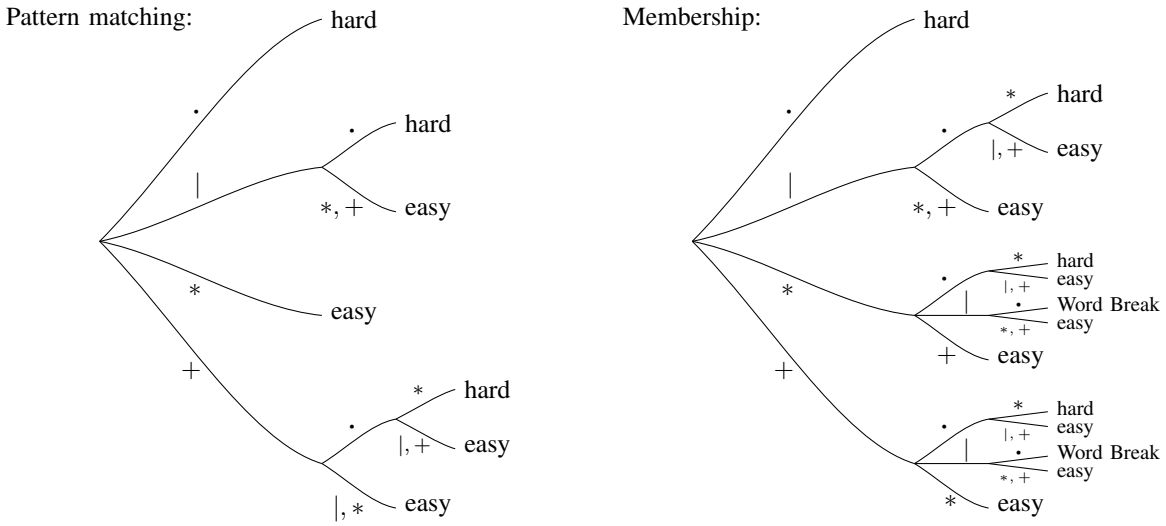

Figure 1: Tree diagrams visualizing Table II. Depth 3 types are classified as "easy" (near-linear time), "hard" (near-quadratic time, assuming SETH), or "Word Break" (whose complexity is not determined). The leftmost operators in each tree correspond to the leftmost operators in type descriptions.

the type "$*|\cdot$" corresponds to the aforementioned word break problem which we discuss in the next paragraph. Finally, types starting with $+$ are analogous to those starting with $*$.

The word break problem is the only problem in the table whose (conditional) complexity is not determined up to logarithmic factors. There are several known solutions to this problem based on dynamic programming [20], [23]. A careful implementation of those algorithms (using substring hashing and pruning) leads to a runtime of $O(nm^{0.5}\log^{O(1)} n)$. However, we show that this bound is not tight, and can be further improved to roughly $nm^{0.5\,-\,1/18}$. Our new algorithm speeds up the dynamic program by using convolutions to pre-compute information that is reused multiple times during the execution of the algorithm. We note that the algorithm is randomized and has a one-sided error.

Due to space limitations, most of the proofs have been omitted from this version of the paper.

Related work: Our hardness results come on the heels of several recent works demonstrating quadratic hardness of sequence alignment problems assuming SETH or other conjectures. In particular, such results have been shown for Local Alignment [2], Fréchet distance [7], Edit Distance [5] and Longest Common Subsequence [1], [8]. As in our case, most of those results were achieved by a reduction from OVP, performed by concatenating appropriately constructed gadgets for the vectors in the input sets. The technical development in our paper is, however, quite different, since regular expression matching is not defined by a sequence similarity measure. Instead, our gadget constructions are tailored to the specific sets of operators and expression types defining the problem variants. Furthermore, we exploit the similarity between related expression types (such as "$\cdot+\cdot$" and "$\cdot|\cdot$") and show how to convert a hardness proof for one type into a hardness proof for the other type.

The reduction in Section III-A has been independently discovered by Kasper Larsen and Raphael Clifford (personal communication). Conditional lower bounds (via reductions from 3SUM) for certain classes of regular expressions have been investigated in [4]. Estimating the complexity of regular expression matching using *specific* algorithms has also been a focus of several papers. See e.g., [24] and the references therein.

II. Preliminaries

Orthogonal Vectors problem: Our reductions use the unbalanced version of the *Orthogonal Vectors Problem*, defined as follows: given two sets $A, B \subseteq \{0,1\}^d$ such that $|A| = M, |B| = N$, determine whether there exists $x \in A$ and $y \in B$ such that the dot product $x \cdot y = \sum_{j=1}^{d} x_j y_j$ (taken over the reals) is equal to 0. An equivalent formulation of this problem is: given two collections of subsets of $\{1, \ldots, d\}$, of sizes M and N, respectively, determine if there is a set in the first collection that is contained in a set from the second collection.

It is known that, for any $M = \Theta(N^\alpha)$ for some $\alpha \in (0, 1]$ and any constant $\delta > 0$, any algorithm for OVP problem with an $O(MN)^{1-\delta}$ running time would also yield a more efficient algorithm for SAT, violating the Strong Exponential Time Hypothesis, even in the setting when the dimension d is arbitrary $d = \omega(\log N)$. This was shown

in [25] for the balanced case $M = N$, and extended to the unbalanced case in [8]. Therefore, in this paper we show that a problem is SETH-hard by reducing unbalanced OVP to it.

Subset Matching problem: In the subset matching problem, we are given a pattern string p and a text string t where each pattern and text location is a set of symbols drawn from some alphabet. The pattern is said to occur at the text position i if the set p_j is a subset of the set t_{i+j} for all j. The goal of the problem is find all positions where p occurs in t. The problem can be solved in deterministic $O(N \log^2 N)$ time, where $N = \sum_i |t_i| + \sum_i |p_i|$ [10].

Superset Matching problem: This problem is analogous to subset matching except that we require that p_j is a *superset* of the set t_{i+j} for all j. The aforementioned algorithm of [10] applies to this problem as well.

Wildcard Matching problem: In the wildcard matching problem, we are given a pattern string p and a text string t where each pattern and text location is an element from $\Sigma \cup \{.\}$, where "." is the special wildcard symbol. The pattern is said to occur at the text position i if for all j we have that (i) one of the symbols p_j and t_{i+j} is equal to ".", or (ii) $p_j = t_{i+j}$. The goal of the problem is find all positions where p occurs in t. The problem can be solved in deterministic $O(n \log n)$ time [10].

Regular expressions: Regular expressions over a symbol set Σ and an operator set $O = \{\cdot, |, +, *\}$ are defined recursively as follows:

- a is a regular expression, for any $a \in \Sigma$;
- if R and S are regular expressions then so are $[R]|[S]$, $[R]\cdot[S]$, $[R]^+$ and $[R]^*$.

For the sake of simplicity, in the rest of this paper we will typically omit the concatenation operator $\cdot$, and also omit some of the parenthesis if the expression is clear from the context.

A regular expression p determines a language $L(p)$ over Σ. Specifically, for any regular expressions R, S and any $a \in \Sigma$, we have: $L(a) = \{a\}$; $L(R|S) = L(R) \cup L(S)$; $L(R\cdot S) = \{uv : u \in L(R), v \in L(S)\}$; $L(R^+) = \cup_{i \geq 1} \cdot_{j=1}^{i} L(R)$; and $L(R^*) = L(R^+) \cup \{\epsilon\}$, where ϵ denotes the empty word.

Regular expressions can be viewed as rooted labeled trees, with internal nodes labeled with operators from O and leaves labeled with symbols from Σ. Note that the number of children of an internal node is not fixed, and can range between 1 (for $+$ and $*$) and m. We say that a regular expression is *homogeneous* if all internal node labels at the same tree level are equal. Note that this definition does not preclude expressions such as aa^+ where not all leaves have the same depth. A homogeneous formula of depth k has *type* $o_1 o_2 \dots o_k$, $o_i \in O$, if for all levels i the operators at level i are equal to o_i. For example, aa^+ has type "$\cdot+$".

To state our results, it is convenient to identify depth-3 homogeneous regular expressions that are equivalent to some depth-2 regular expressions. In particular, in all types starting from $+$, the operator $+$ can be removed, since p^+ occurs in the text t if and only if p occurs in t. Similarly, in all types starting from $|+$, the operator $+$ can be removed, for the same reasons.

Notation: For symbol (or sequence) z and integer i, z^i denotes symbol (or sequence) z repeated i times. For an integer n, $[n]$ denotes $\{1, 2, 3, \dots n\}$. Given an integer d, 0_d (1_d, resp.) denotes the vector with all entries equal to 0 (1, resp.) in d dimensions. For sequences $s_1, s_2, \dots, s_k$, we write $\bigodot_{i=1}^{k} s_i$ to denote the concatenation $s_1\, s_2\, \dots\, s_k$ of the sequences.

Simplifying assumptions: To simplify our proofs, we will make several simplifying assumptions about the Orthogonal Vectors instance (different assumptions are used in different proofs). In what follows, we describe what kind of assumptions we make, and how to satisfy them:

1) The number of vectors M in set A is odd or even (depending on the proof): this can be achieved w.l.o.g. since we can add a vector to set A consisting only of 1s.
2) The dimensionality d of vectors from sets A and B is odd or even (depending on the proof): this can be achieved w.l.o.g. since we can add new coordinate to every vector and set this coordinate to 0.
3) Let $A = \{a^1, \dots, a^M\}$ and $B = \{b^1, \dots, b^N\}$. If there are i, j such that $a^i \cdot b^j = 0$ then there are i', j' such that $a^{i'} \cdot b^{j'} = 0$ and $i' \equiv j' (\mathrm{mod}\ 2)$: this can be assumed w.l.o.g. since we can define a set $A' = \{a^M, a^1, a^2, a^3, \dots, a^{M-1}\}$ and perform two reductions, one on the pair of sets A and B and another one on the pair of sets A' and B. If a pair of orthogonal vectors exists, one of these reductions will detect it.
4) The dimensionality d is greater than 100: we can assume this since otherwise Orthogonal Vectors problem can be solved in $O(2^d \cdot N) = O(N)$ time.
5) $b_1^t = b_d^t = 0$ for all $t \in [N]$ and d is odd: first, we make d odd as described above. Then we add two entries for every vector from A or B, one at the beginning and one at the end, and set both entries to 0.
6) The first vector a^1 from the set A is not orthogonal to all vectors from B: first, we can detect whether this is the case in $O(dN)$ time. If a^1 is orthogonal to a vector from B, we have found a pair of orthogonal vectors. Otherwise we proceed with the reduction.

III. Reductions for the Pattern Matching problem

We start this section by showing hardness for regular expressions of type "$|\cdot|$" and "$|\cdot+$". These hardness proofs are quite simple, and will help us introduce notation used in more complex reductions presented later. After that we present the hardness proof for regular expressions of type "$\cdot*$".

A. Hardness for type "$|\cdot|$"

Theorem 1. *Given sets $A = \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B = \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression p and a sequence of symbols t, in $O(Nd)$ time, such that a substring of t can be derived from p if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, p has type "$|\cdot|$", $|p| \leq O(Md)$ and $|t| \leq O(Nd)$.*

Proof: First, we will construct our pattern p. For an integer $v \in \{0,1\}$, we construct the following pattern coordinate gadget

$$CG(v) := \begin{cases} [0|1] & \text{if } v = 0; \\ [0] & \text{if } v = 1. \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define a pattern vector gadget

$$VG(a) := CG(a_1)\, CG(a_2)\, CG(a_3)\, \ldots\, CG(a_d).$$

Our pattern p is then defined as "$|$" of all pattern vector gadgets:

$$\begin{aligned} p :=& VG\left(a^1\right) | VG\left(a^2\right) | VG\left(a^3\right) | VG\left(a^4\right) | \\ & \ldots | VG\left(a^{M-1}\right) | VG\left(a^M\right). \end{aligned}$$

Now we construct the text t. First, for a vector $b \in \{0,1\}^d$, we define text vector gadget as concatenation of all entries of b: $VG'(b) := b_1\, b_2\, b_3\, b_4\, \ldots\, b_d$. Note that we can derive $VG'(b)$ from $VG(a)$ if and only if $a \cdot b = 0$. Our text t is defined as a concatenation of all text vector gadgets with a symbol 2 in between any two neighbouring vector gadgets:

$$\begin{aligned} t :=& VG'\left(b^1\right) 2\, VG'\left(b^2\right) 2\, VG'\left(b^3\right) 2\, VG'\left(b^4\right) 2 \\ & \ldots 2\, VG'\left(b^{N-1}\right) 2\, VG'\left(b^N\right). \end{aligned}$$

We need to show that we can derive a substring of t from p if and only if there are two orthogonal vectors in A and B. This follows from lemmas 1 and 2 below. ∎

Lemma 1. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of t can be derived from p.*

Proof: Suppose that $a^i \cdot b^j = 0$ for some $i \in [M]$, $j \in [N]$. We choose a pattern vector gadget $VG(a^i)$ from the pattern p and transform it into a text vector gadget $VG'(b^j)$. This is possible because of the orthogonality and the construction of the vector gadgets. ∎

Lemma 2. *If a substring of t can be derived from p, then there are two orthogonal vectors.*

Proof: By the construction of pattern p, we have to choose one pattern vector gadget, say, $VG(a^i)$, that is transformed into a binary substring of t of length d. The text t has the property that it is a concatenation of binary strings of length d separated by symbols 2. This means that $VG(a^i)$ will be transformed into binary string $VG'(b^j)$ for some j. This implies that $a^i \cdot b^j = 0$ by the construction of the vector gadgets. ∎

B. Hardness for type "$|\cdot+$"

Theorem 2. *Given sets $A = \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B = \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression p and a sequence of symbols t, in $O(Nd)$ time, such that a substring of t can be derived from p iff there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, p has type "$|\cdot+$", $|p| \leq O(Md)$ and $|t| \leq O(Nd)$.*

Proof: First, we will construct our pattern. For an integer $v \in \{0,1\}$, we construct the following pattern coordinate gadget

$$CG(v) := \begin{cases} x^+ & \text{if } v = 0; \\ x^+x^+ & \text{if } v = 1. \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define a pattern vector gadget as concatenation of coordinate gadgets for all coordinates with the symbol y in between every two neighbouring coordinate gadgets:

$$VG(a) := CG(a_1)\, y\, CG(a_2)\, y\, CG(a_3)\, y\, \ldots\, y\, CG(a_d).$$

Our pattern p is then defined as an OR ("$|$") of all the pattern vector gadgets:

$$\begin{aligned} p :=& VG\left(a^1\right) | VG\left(a^2\right) | VG\left(a^3\right) | VG\left(a^4\right) | \\ & \ldots | VG\left(a^{M-1}\right) | VG\left(a^M\right). \end{aligned}$$

Now we proceed with the construction of our text t. For an integer $v \in \{0,1\}$, we define the following text coordinate gadget

$$CG'(v) := \begin{cases} xx & \text{if } v = 0; \\ x & \text{if } v = 1. \end{cases}$$

For vector $b \in \{0,1\}^d$, we define the text vector gadget as

$$\begin{aligned} VG'(b) :=& CG'(b_1, 1)\, y\, CG'(b_2, 2)\, y\, CG'(b_3, 3)\, y \\ & \ldots\, y\, CG'(b_d, d). \end{aligned}$$

Note that we can derive $VG'(b)$ from $VG(a)$ iff $a \cdot b = 0$. Our text t is defined as a concatenation of all text vector gadgets with the symbol z in between any two neighbouring vector gadgets:

$$\begin{aligned} t :=& VG'(b^1)\, z\, VG'(b^2)\, z\, VG'(b^3)\, z\, VG'(b^4)\, z \\ & \ldots\, z\, VG'(b^{N-1})\, z\, VG'(b^N). \end{aligned}$$

We need to show that we can derive a substring of t from p iff there are two orthogonal vectors. This follows from lemmas 3 and 4 below. ∎

Lemma 3. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of t can be derived from p.*

Proof: Suppose that $a^i \cdot b^j = 0$ for some $i \in [M]$, $j \in [N]$. We choose a pattern vector gadget $VG(a^i)$ from the pattern p and transform it into a text vector gadget $VG'(b^j)$. This is possible because of the orthogonality and the construction of the vector gadgets. ∎

Lemma 4. *If a substring of t can be derived from p, then there are two orthogonal vectors.*

Proof: We call a sequence of symbols *nice* iff it can be derived from the regular expression $x^+yx^+yx^+y \dots yx^+$, where "x^+" appears d times.

By the construction of pattern p, we have to choose one pattern vector gadget, say, $VG(a^i)$, that is transformed into a nice sequence. The text t has the property that it is a concatenation of nice sequences separated by symbols z. This means that $VG(a^i)$ will be transformed into a sequence $VG'(b^j)$ for some j. This implies that $a^i \cdot b^j = 0$ by the construction of vector gadgets. ∎

C. Hardness for type "$\cdot*$"

Theorem 3. *Given sets $A = \{a^1, \dots, a^M\} \subseteq \{0,1\}^d$ and $B = \{b^1, \dots, b^N\} \subseteq \{0,1\}^d$ with $M \leq N$, we can construct the regular expression p and the text t in time $O(Nd)$, such that a substring of t can be derived from p iff there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, p is of type "$\cdot*$", $|p| \leq O(Md)$ and $|t| \leq O(Nd)$.*

Proof: W.l.o.g., we can assume that $M \equiv 1 (\text{mod } 2)$ and $d \equiv 1 (\text{mod } 2)$, $d \geq 100$. Also, if there are $i \in [M]$, $j \in [N]$ such that $a^i \cdot b^j = 0$, then there are $i' \in [M]$, $j' \in [N]$ such that $a^{i'} \cdot b^{j'} = 0$ and $i' \equiv j' (\text{mod } 2)$. Furthermore, we assume $b^j_1 = b^j_d = 0$ for all $j \in [N]$ and that a^1 is not orthogonal to any vector b^j.

First, we will construct our pattern. For an integer $v \in \{0,1\}$ and an integer $i \in [d]$, we construct the following pattern coordinate gadget

$$CG(v,i) := \begin{cases} yy^* & \text{if } v = 0 \text{ and } i \equiv 1 (\text{mod } 2); \\ yyyy^* & \text{if } v = 1 \text{ and } i \equiv 1 (\text{mod } 2); \\ xx^* & \text{if } v = 0 \text{ and } i \equiv 0 (\text{mod } 2); \\ xxxx^* & \text{if } v = 1 \text{ and } i \equiv 0 (\text{mod } 2). \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define a pattern vector gadget

$$VG(a) := CG(a_1, 1)\, CG(a_2, 2)\, CG(a_3, 3)\, \dots\, CG(a_d, d).$$

We also need another pattern vector gadget $VG_0 := (y^*\, x^*)^{d+10}\, y^*$.

Our pattern is then defined as follows:

$$p := y^6 \bigodot_{j \in [M-1]} \left(x^{10}\, VG(a^j)\, x^{10}\, VG_0\right)\, x^{10}\, VG(a^M)\, x^{10}\, y^6.$$

Now we proceed with the construction of our text. For integers $v \in \{0,1\}, i \in [d]$, we define the following text coordinate gadget

$$CG'(v,i) := \begin{cases} yyy & \text{if } v = 0 \text{ and } i \equiv 1 (\text{mod } 2); \\ y & \text{if } v = 1 \text{ and } i \equiv 1 (\text{mod } 2); \\ xxx & \text{if } v = 0 \text{ and } i \equiv 0 (\text{mod } 2); \\ x & \text{if } v = 1 \text{ and } i \equiv 0 (\text{mod } 2). \end{cases}$$

For a vector $b \in \{0,1\}^d$ and an integer $j \equiv 1 (\text{mod } 2)$, we define the text vector gadget as

$$VG'(b,j) := CG'(b_1, 1)\, CG'(b_2, 2)\, CG'(b_3, 3)\, \dots\, CG'(b_d, d).$$

We also define $VG'(b,j)$, when $j \equiv 0 (\text{mod } 2)$. In this case, $VG'(b,j)$ is equal to $VG'(b,1)$ except that we replace every occurrence of the substring y^3 with the substring y^6.

One can verify that for any *vectors* $a, b \in \{0,1\}^d$ and any integer i, $VG'(b,i)$ can be derived from $VG(a)$ iff $a \cdot b = 0$.

We will also need an additional text vector gadget

$$VG'_0 := y^3\, (x^3\, y^3)^{(d-1)/2}.$$

Our text is then defined as follows:

$$t := \bigodot_{j=-2N}^{3N} \left(x^{10}\, VG'_0\, x^{10}\, VG'(b^j, j)\right),$$

where we assume $b^j := 011111 \dots 111110$ for $j \notin [N]$.

We have to show that we can derive a substring of t from p iff there are two orthogonal vectors. This follows from lemmas 5 and 6 below. ∎

Lemma 5. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of t can be derived from p.*

Proof: W.l.o.g. we have that, $a^k \cdot b^k = 0$ for some $k \in [M]$. The proof for the case when $a^k \cdot b^r = 0$, $k \in [M]$, $r \in [N]$, $k \equiv r (\text{mod } 2)$ is analogous.

The pattern p starts with y^6. We transform it into $CG'(b^0_d, d)$ appearing in $VG'(b^0, 0)$. We can do this since $b^0_d = 0$.

For $j = 1, 2, \dots, k-2$, we transform $x^{10}\, VG(a^j)\, x^{10}\, VG_0$ into $x^{10}\, VG'_0\, x^{10}\, VG'(b^j, j)$ by transforming $VG(a^j)$ into VG'_0 and VG_0 into $VG'(b^j, j)$.

Next we transform

$$x^{10}\, VG(a^{k-1})\, x^{10}\, VG_0\, x^{10}\, VG(a^k)\, x^{10}\, VG_0$$

into

$$x^{10}\, VG'_0\, x^{10}\, VG'(b^{k-1}, k-1)\, x^{10}\, VG'_0\, x^{10}$$
$$VG'(b^k, k)\, x^{10}\, VG'_0\, x^{10}\, VG'(b^{k+1}, k+1)$$

Notice that we use the fact that $k \geq 2$ (we assumed that a^1 is not orthogonal to any vector from B). Note that $VG'(b^{k+1}, k+1)$ appears in the text t even if $k = N$. This is because in the definition of the text t, integer j ranges from $-2N$ up to $3N$.

We perform the transformation by performing the following steps:

1) transform $VG(a^{k-1})$ into VG'_0;
2) transform VG_0 into $VG'(b^{k-1}, k-1)\, x^{10}\, VG'_0$;
3) transform $VG(a^k)$ into $VG'(b^k, k)$ (we can do this since $a^k \cdot b^k = 0$);
4) transform VG_0 into $VG'_0\, x^{10}\, VG'(b^{k+1}, k+1)$.

Now, for $j = k+1, \ldots, M-1$ transform $x^{10}\, VG(a^j)\, x^{10}\, VG_0$ into $x^{10}\, VG'_0\, x^{10}\, VG'(b^{j+1}, j+1)$ similarly as before. Next, transform $x^{10}\, VG(a^M)\, x^{10}$ into $x^{10}\, VG'_0\, x^{10}$. Finally, transform y^6 into $CG'(b_1^{M+1})$ appearing in $VG'(b^{M+1}, M+1)$. We can do this since $b_1^{M+1} = 0$. ∎

Lemma 6. *If a substring of t can be derived from p, then there are two orthogonal vectors.*

Proof: By the construction, every substring x^{10} from p must be mapped to a unique substring x^{10} in t (there are no substrings of t that have more than 10 symbols x). Because of this, every $VG(a^i)$ must be mapped to VG'_0 or $VG'(b^j, j)$ for some j. If the latter case occurs, the corresponding vectors are orthogonal and we are done. It remains to consider the case that *all* vector gadgets $VG(a^i)$ get mapped to VG'_0. Consider any vector gadget VG_0 in p. To the left of it we have the sequence x^{10} and to the right of it we have the sequence x^{10}. Each one of these two sequences x^{10} in p gets mapped to a unique sequence x^{10} in t. We call the vector gadget VG_0 *nice* if the two unique sequences x^{10} are neighbouring in t, that is, there is no other sequence x^{10} in t between the two unique sequences. We consider two cases below.

Case 1: There is a vector gadget VG_0 in p that is *not* nice. Take any vector gadget VG_0 that is not nice and denote it by v. The gadget v is immediately to the right of the expression $VG(a^{i'})\, x^{10}$ in p for some $i' \in [M]$. $VG(a^{i'})$ is mapped to VG'_0 (otherwise, we have found an orthogonal pair of vectors, as per the discussion above) and this VG'_0 is to the left of substring $x^{10}\, VG'(b^{j''}, j'')$ in t for some j''. Because v is *not* nice, a prefix of it must map to $VG'(b^{j''}, j'')\, x^{10}\, VG'_0$. We claim that *entire* v gets mapped to $VG'(b^{j''}, j'')\, x^{10}\, VG'_0$. If this is not the case, then a prefix of v must be mapped to $VG'(b^{j''}, j'')\, x^{10}\, VG'_0\, x^{10}\, VG'(b^{j''+1}, j''+1)$ (since sequence x^{10} in p to the right of v must be mapped to x^{10}). A prefix of v can't be mapped to $VG'(b^{j''}, j'')\, x^{10}\, VG'_0\, x^{10}\, VG'(b^{j''+1}, j''+1)$ since $v = (y^*\, x^*)^{d+10} y^*$ can produce sequence with at most $d+11$ substrings of maximal length consisting entirely of symbols y but sequence $VG'(b^{j''}, j'')\, x^{10}\, VG'_0\, x^{10}\, VG'(b^{j''+1}, j''+1)$ has $3\frac{d+1}{2} > d+11$ (if $d \geq 100$) subsequences of maximal length consisting entirely of y. Therefore, we are left with the case that entire v is mapped to $VG'(b^{j''}, j'')\, x^{10}\, VG'_0$. The gadget v is to the left of vector gadget $VG(a^{i'+1})$ and $VG'(b^{j''}, j'')\, x^{10}\, VG'_0$ is to the left of vector gadget $VG'(b^{j''+1}, j''+1)$. We conclude that $VG(a^{i'+1})$ must be mapped to $VG'(b^{j''+1}, j''+1)$. This implies that $a^{i'+1} \cdot b^{j''+1} = 0$ and we are done.

Case 2: All vector gadgets VG_0 in p are nice. We start p with y^6 followed immediately by x^{10}. This means that y^6 is mapped to $CG'(b_d^{j'}, j')$ for some *even* j' (by the construction of coordinate gadgets CG'). Consider vector gadgets in p from left to the right. We must have that $VG(a^1)$ is mapped to VG'_0, that VG_0 is mapped to $VG'(b^1, 1)$ (since every VG_0 in p is nice), that $VG(a^2)$ is mapped to VG'_0, that VG_0 is mapped to $VG'(b^2, 2)$ (since every VG_0 in p is nice) and so forth. Since M is odd, we have that $VG(a^M)$ is mapped to VG'_0 and that this vector gadget VG'_0 is followed by $x^{10}\, VG'(b^{j'+M}, j'+M)$. $VG(a^M)$ is followed by $x^{10} y^6$ and this means that y^6 is mapped to the beginning of $VG'(b^{j'+M}, j'+M)$. This is impossible since $VG'(b^{j'+M}, j'+M)$ does not contain a substring of length 6 or more consisting of symbols y (observe that $j'+M$ is odd and see the construction of vector gadget VG'). We get that Case 2 can't happen. ∎

ACKNOWLEDGMENTS

We thank Ludwig Schmidt and the reviewers for many helpful comments. This work was supported by grants from the NSF, the MADALGO center, and the Simons Investigator award.

REFERENCES

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and other Sequence Similarity Measures. *FOCS*, 2015.

[2] Amir Abboud, V. Vassilevska Williams, and Oren Weimann. Consequences of faster sequence alignment. *ICALP*, 2014.

[3] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[4] Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B Riva Shalom. Mind the gap. *arXiv preprint arXiv:1503.07563*, 2015.

[5] Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). *STOC*, 2015.

[6] Philip Bille and Mikkel Thorup. Faster regular expression matching. In *Automata, Languages and Programming*, pages 171–182. Springer, 2009.

[7] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. *FOCS*, 2014.

[8] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 79–97. IEEE, 2015.

[9] Richard Cole and Ramesh Hariharan. Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 66–75. ACM, 1997.

[10] Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the Thirty-fourth annual ACM symposium on Theory of computing*, pages 592–601. ACM, 2002.

[11] Michael J Fischer and Michael S Paterson. String-Matching and Other Products. Technical report, DTIC Document, 1974.

[12] Zvi Galil. Open problems in stringology. In *Combinatorial Algorithms on Words*, pages 1–8. Springer, 1985.

[13] Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, pages 7–10, 1999.

[14] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[15] Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 166–173. IEEE, 1998.

[16] Adam Kalai. Efficient pattern-matching with don't cares. In *SODA*, volume 2, pages 655–656, 2002.

[17] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton: multitouch gestures as regular expressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2885–2894. ACM, 2012.

[18] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[19] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. *ACM SIGCOMM Computer Communication Review*, 36(4):339–350, 2006.

[20] LeetCode. Problem 139. Word Break. *https://leetcode.com/problems/word-break/*.

[21] Gene Myers. A four russians algorithm for regular expression pattern matching. *Journal of the ACM (JACM)*, 39(2):432–448, 1992.

[22] Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *Journal of Computational Biology*, 10(6):903–923, 2003.

[23] Daniel Tunkelang. Retiring a Great Interview Problem. *http://thenoisychannel.com/2011/08/08/retiring-a-great-interview-problem*, 2011.

[24] Nicolaas Weideman, Brink van der Merwe, Martin Berglund, and Bruce Watson. Analyzing matching time behavior of backtracking regular expression matchers by using ambiguity of nfa. In *International Conference on Implementation and Application of Automata*, pages 322–334. Springer, 2016.

[25] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.