

SAARLAND UNIVERSITY  
SAARLAND INFORMATICS CAMPUS

MASTER'S THESIS

---

FINE-GRAINED COMPLEXITY  
AND SHAVING LOG-FACTORS FOR  
REGULAR EXPRESSION PATTERN MATCHING  
AND MEMBERSHIP

---

**Author**

Philipp Johann Schepper

**Advisor**

Prof. Dr. Karl Bringmann

**Reviewers**


Prof. Dr. Karl Bringmann

Prof. Dr. Markus Bläser

Submitted: 22 January 2020



---



## Statement Erklärung

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

---



## Declaration of Consent Einverständniserklärung

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.


Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Saarbrücken, 22 January 2020

---

(PHILIPP J. SCHEPPER)





---


# Abstract

## Zusammenfassung

The classical  $\mathcal{O}(nm)$  time algorithms for regular expression pattern matching and membership can be improved by a factor of about  $\log^{3/2} n$ . Instead of focussing on general patterns we focus on homogeneous patterns of bounded depth in this thesis only. For them a classification splitting the types in easy (strongly sub-quadratic) and hard (essentially quadratic time under SETH) is known. We take a fine-grained look at the hard pattern types from this classification and show that few types allow super-poly-logarithmic improvements while the algorithms for the other pattern types can only be improved by a constant number of log-factors, assuming the FORMULA-SAT HYPOTHESIS.

Der klassische  $\mathcal{O}(nm)$ -Zeit Algorithmus für Pattern-Matching und Membership von regulären Ausdrücken kann um einen Faktor von  $\log^{3/2} n$  verbessert werden. In dieser Arbeit beschäftigen wir uns allerdings nicht mit allgemeinen Pattern sondern ausschließlich mit homogenen Pattern. Für diese ist eine Klassifikation in einfache (echt sub-quadratische Zeit) und schwere Typen (im Grunde quadratische Zeit unter SETH) bekannt. Wir führen eine fine-grained Analyse für diese harten Pattern-Typen durch und zeigen, dass manche Typen eine super-poly-logarithmische Laufzeitverbesserung erlauben, während die anderen Typen nur eine Verbesserung um eine konstante Zahl an log-Faktoren zulässt, vorausgesetzt die FORMULA-SAT HYPOTHESE stimmt.





---

## Acknowledgements

## Danksagung

First and foremost I want to thank my advisor Karl Bringmann for offering me this interesting topic as a master's thesis. Throughout the research he guided me with his experience and helped with new ideas when I was stuck or was heading into the wrong direction. Further, I thank Professor Bläser for his readiness to be the second reviewer of this thesis.

Additionally, I want to thank the IMPRS-CS that I could be a fellow during my whole master's studies. Its financial funding and the funding by the German Academic Scholarship Foundation (Studienstiftung des deutschen Volkes) gave me the possibility to completely focus on the studies and this thesis. Further, I want to thank my colleagues and friends at the MPI and Saarland University. I am deeply grateful for the uncountably many conversations and the time we could spend together during the last two years. But I also want to thank them for their ideas and useful comments about the problems I worked on and I hope my comments are helpful too.

Ebenso möchte ich natürlich aber auch all diejenigen danken, die mich außerhalb des Studiums unterstützt und mir auch mal eine Auszeit vom Studienalltag geboten haben. Ein besonderer Dank geht hier natürlich an die Kameraden bei der Feuerwehr und die Kollegen bei der Ostertalbahn, wo ich ja fast immer bin wenn nicht gerade an der Uni oder zu Hause.

Aber zu guter Letzt möchte ich natürlich meinen Eltern für ihre gesamte Unterstützung während meines Studiums danken. Ohne ihre Hilfe wäre es mir sicherlich nicht möglich gewesen, mich in diesem Maß auf das Studium und insbesondere auf diese Arbeit zu konzentrieren. Vielen Dank!







---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Our Results . . . . .	3
1.3	Structure of the Thesis . . . . .	4
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
2.1	Regular Expressions . . . . .	5
2.1.1	Basic Definitions . . . . .	5
2.1.2	Homogeneous Patterns . . . . .	6
2.1.3	The Size of Patterns . . . . .	7
2.2	Satisfiability Problems . . . . .	8
2.2.1	Definition of the Problems . . . . .	8
2.2.2	Hypotheses . . . . .	9
2.3	Known Results and Definitions . . . . .	11
<b>3</b>	<b>Upper Bounds by Faster Algorithms</b>	<b>15</b>
3.1	General Idea . . . . .	15
3.2	Small Patterns of Type $ \circ $ . . . . .	19
3.2.1	The Circuit . . . . .	19
3.2.2	The Polynomial . . . . .	20
3.2.3	Counting the Monomials . . . . .	21
3.3	Small Patterns of Type $ \circ^+$ . . . . .	23
3.3.1	Encoding Texts and Patterns . . . . .	23
3.3.2	The Circuit . . . . .	24
3.3.3	The Polynomial . . . . .	27
3.3.4	Counting the Monomials . . . . .	29
3.4	Derandomizing the Algorithm for $ \circ $ . . . . .	30
<b>4</b>	<b>Conditional Lower Bounds for Pattern Matching</b>	<b>33</b>
4.1	Results and General Idea . . . . .	34
4.2	Patterns of Type $\circ^+\circ$ . . . . .	36
4.2.1	Encoding the Formula . . . . .	36
4.2.2	Final Reduction . . . . .	38
4.3	Patterns of Type $\circ \circ$ . . . . .	40

4.4	Patterns of Type $\circ\star$ . . . . .	40
4.5	Patterns of Type $\circ+ $ . . . . .	43
4.6	Patterns of Type $\circ +$ . . . . .	44
4.7	Proof of Theorem 4.6 . . . . .	46
<b>5</b>	<b>Conditional Lower Bounds for Membership</b>	<b>49</b>
5.1	Reducing Pattern Matching to Membership . . . . .	50
5.1.1	Patterns of Type $\circ\star$ . . . . .	50
5.1.2	Patterns of Type $\circ+\circ$ . . . . .	50
5.1.3	Patterns of Type $\circ \circ$ . . . . .	51
5.1.4	Patterns of Type $\circ+ $ . . . . .	51
5.1.5	Patterns of Type $\circ +$ . . . . .	52
5.2	Patterns of Type $ + \circ$ . . . . .	52
5.2.1	Encoding the Formula . . . . .	53
5.2.2	Final Reduction . . . . .	56
<b>6</b>	<b>Conclusion</b>	<b>59</b>
6.1	The Results . . . . .	59
6.2	Open Problems and Further Questions . . . . .	61
<b>A</b>	<b>Homogeneous Patterns of Unbounded Depth</b>	<b>65</b>
	<b>Bibliography</b>	<b>67</b>
	<b>Abbreviations and Notations</b>	<b>73</b>

---

# Introduction

## 1.1 Motivation

**REGULAR EXPRESSION PROBLEMS** Regular expressions as introduced by Kleene in [Kle56] are present in many fields of computer science and other sciences. Since only a few commands or operations are needed (concatenation, alternative and Kleene star), it is very easy to write them down. But even with these three operations they offer a wide variety to combine these types to form patterns. Since these patterns consist of operations applied to other patterns and symbols, they are used in most cases to work on texts, with text as in the literal sense. They are appropriate to analyse texts pieces by a search tool or whole files containing certain strings by special dedicated tools such as `grep` and `egrep` for example. But they are also suitable for text manipulation as the unix command `sed` shows that searches for a pattern and replaces all occurrences of it by a specified text. Most programming languages also offer this possibility to perform searches and replacements based on regular expressions. However, they are not only suitable for text manipulation but also in biology when considering protein search in DNA sequences for example [Lan92; NR03]. Or they are used to analyse data networks as described in [Yu+06; JMR07]. Additionally one can query XML files using regular expressions as shown in [LM01; Mur01] or they help with human computer interaction [Kin+12]. All of these fields show that there are plenty of applications for regular expressions and they are not only an interesting topic in language theory of theoretical computer science. In this thesis we focus on regular expression pattern matching and membership. While for the first problem only a *substring* of the input text has to be matched, the *complete string* has to be matched for membership. We give a formal definition of the problems in Section 2.1.

**FINE-GRAINED ANALYSIS** A natural analysis of algorithms solving a specific problem involves an analysis of the running time. Using this and the design of faster algorithms we find upper bounds for the problem. While in “standard” complexity theory a running time of  $\mathcal{O}(n^c)$  for some constant  $c > 0$  is considered to be efficient, a running time of  $\mathcal{O}(n^2)$  still might be too slow for real data if the input size is very large. But despite on-going research often no major improvement of these bounds was found. Hence, one tries to prove lower bounds for the problems. Since this is very challenging one relaxed it to show *conditional* lower bounds instead (see discussion in Section 2.2.2). For these conditional bounds one assumes a popular conjecture and shows that under this

assumption the running time of a different problem cannot be improved. If the currently fastest algorithm has a running time of  $\mathcal{O}((nm)^c)$ , one is usually looking for conditional lower bounds ruling out algorithms with running time of  $\mathcal{O}((nm)^{c-\epsilon})$  for any  $\epsilon > 0$ .<sup>1</sup> These bounds imply that the algorithm cannot be improved too much, i.e. by a polynomial factor. But there still might be possibilities to improve the algorithm even further as the following example shows: The naive algorithm for ORTHOGONAL VECTORS ([CW16; Wil14a]) runs in time  $\mathcal{O}(n^2 d)$  for small dimensions  $d \in \Theta(\log^2 n)$ . A reduction from the SATISFIABILITY PROBLEM (SAT) shows that there is no algorithm with runtime  $\mathcal{O}(n^{2-\epsilon})$ , unless SAT has faster algorithms (cf. Hypothesis 1) [BK15]. But in [CW16] an algorithm with runtime  $\mathcal{O}(n^{2-1/\mathcal{O}(\log \log n)}) = \mathcal{O}(n^2/2^{\log n/\log \log n}) \subseteq \mathcal{O}(n^2/2^{\Theta(\sqrt{\log n})})$  is shown.

The current lower bounds for pattern matching and membership do not rule out such improvements but rather show that there are no sub-quadratic time algorithms ([BI16; BGL17]). Let  $n$  be the length of some input text and  $m$  be the size of the input pattern in the following. Despite intense research the running time was only improved from the classical  $\mathcal{O}(nm)$  approach as shown in [Tho68] first to  $\mathcal{O}(nm/\log n + (n+m)\log n)$  by Myers [Mye92] and then to  $\mathcal{O}(nm \text{ polylog } \log n / \log^{3/2} n)$  by Bille and Thorup [BT09]. Whether faster algorithms exist was still open until a recent result by Abboud and Bringmann [AB18]. They show that under a conjecture for a specialized version of SAT (Hypothesis 2) there is no algorithm for pattern matching and membership operating in time  $\mathcal{O}(nm/\log^7 n)$ . This result is quite astonishing since all previous lower bounds ruled out *polynomial* improvements and this results rules out *log-factor* improvements which is a bound on a different magnitude.

IDENTIFYING HARD PATTERNS Nevertheless, the result in [AB18] is for general patterns. This does not rule out that the “simpler” patterns considered in [BI16; BGL17] allow such improvements. These “simpler” patterns are *homogeneous* patterns of bounded *depth*. For now it suffices to see them as follows: When considering a pattern as tree, then all operators on the same level have to be equal. The type is the sequence of operators from the root to the leaf with highest depth and the depth of the pattern is the number of operators. A formal definition is given in the preliminaries in Section 2.1. From now on we only use homogeneous patterns of bounded depth unless stated otherwise. To not consider all possible combinations of operators the following lemma is quite helpful:

**Lemma 1.1** (Lemma 1 and Lemma 8 in the full version of [BGL17]). *For any type  $t$ , applying any of the following rules yields a type  $t'$  such that both are equivalent for pattern matching and membership under linear-time reductions, respectively:*

<i>For pattern matching</i>	<i>For membership</i>
<i>replace any substring <math>pp</math>, for any <math>p \in \{\circ,  , \star, +\}</math>, by <math>p</math></i>	<i>replace any substring <math>+ +</math> by <math>+</math></i>
<i>remove prefix <math>+</math></i>	<i>replace prefix <math>r\star</math> by <math>r+</math> for any <math>r \in \{+, \circ\}^*</math></i>
<i>replace prefix <math> +</math> by <math> </math></i>	

*We say that  $t$  simplifies if one of these rules applies. Applying these rules in any order will eventually lead to an unsimplifiable type.*

---

<sup>1</sup>A bound of  $o((nm)^c)$  would be fabulous.

Pattern matching	$\circ\star$	$\circ \circ$	$\circ +$	$\circ+\circ$	$\circ+ $	$  \circ  $	$  \circ +$	
Membership						$+  \circ  $	$+  \circ +$	$  +   \circ$

TABLE 1.1: Hard pattern types

While the modifications above simplified the pattern, the following modifications make the pattern more complicate to show that the hardness of some specific pattern types implies the hardness of other pattern types.

**Lemma 1.2** (Lemma 6 and Lemma 9 in the full version of [BGL17]). *For types  $t$  and  $t'$ , there is a linear-time reduction from  $t$ -pattern matching/membership to  $t'$ -pattern matching/membership if one of the following sufficient conditions holds:*

- $t$  is a prefix of  $t'$ ,
- we may obtain  $t'$  from  $t$  by replacing a  $\star$  by  $+\star$ ,
- we may obtain  $t'$  from  $t$  by inserting a  $|$  at any position,
- only for membership:  $t$  starts with  $\circ$  and we may obtain  $t'$  from  $t$  by prepending a  $+$  to  $t$ .

By these two lemmas and the previous results showing faster (i.e. almost and (near) linear time) algorithms for many patterns [KMP77; CH02; AC75; BI16; BGL17] it suffices to analyse the pattern types in Table 1.1 for the respective problem.

## 1.2 Our Results

As we have seen, there are five pattern types that are hard for pattern matching and for membership. For each type  $T$  of these five types we show tighter conditional lower bounds of the form  $\Omega(nm/\log^{c_T} n)$  with  $20 < c_T < 90$  based on the FORMULA-PAIR HYPOTHESIS (FPH) as stated in Hypothesis 3. We show the same lower bound for  $|+|\circ$ -membership by a similar reduction. We recall from [BT09] that the currently best algorithm for pattern matching and membership runs in time  $\mathcal{O}(nm \text{polyloglog } n / \log^{3/2} n)$ . By this and our improved bounds we show the following theorem:

**Theorem 1.3** (Main Theorem). *The currently best algorithm for  $\circ\star$ ,  $\circ|\circ$ ,  $\circ|+$ ,  $\circ+\circ$ , and  $\circ+|$ -pattern matching and membership and for  $|+|\circ$ -membership is optimal up to a constant number of log-factors, unless FPH is false.*

For the two pattern types  $| \circ |$  and  $| \circ +$  we also show tighter bounds. But instead of giving improved *lower* bounds we show *upper* bounds for pattern matching by using the polynomial method. We show that pattern matching with both types can be solved in time  $\mathcal{O}(nm/2^{\Theta(\sqrt{\log \min(n,m)})})$  when the alphabet is small. This improvement outperforms any number of log-factors that could be shaved off. Hence, there are no such conditional lower bounds as for the other pattern types ruling out log-factor improvements. But still the gap

between  $\mathcal{O}((nm)^{1-\epsilon})$  and the running time of these algorithms is not completely closed. As the polynomial method introduces randomness, these algorithms are randomized but still err only with small probability. Thus, we also present a derandomization for the first pattern type that is based on the same ideas as the derandomized algorithms for ALL-PAIRS SHORTEST PAIRS in [CW16].

We illustrate all of our results in the Figures 6.1, 6.3, 6.2, and 6.4. There we combine the previous results for specific pattern types with our new results. We also state the results for the other pattern types that obtain their hardness through Lemma 1.2 and we mark the patterns that first have to be simplified as mentioned in Lemma 1.1.

While we obtain a full dichotomy for the hard pattern types for pattern matching there is still some gap for membership. It remains an open question whether there are tighter lower bounds for patterns of type  $+|\circ|$  and  $+|\circ+$ . Likewise an improved algorithm could not be found yet. In the conclusion we give some ideas why it sounds reasonable to work on faster algorithms by using the polynomial method. But this has to be addressed in some further work.

### 1.3 Structure of the Thesis

In Chapter 2 we first give a formal definition of regular expressions and the pattern matching and membership problem. Then we introduce the satisfiability problems we use to reduce from and state the conjectures about their running times before introducing some already known results used for the reductions and the algorithm design. The faster algorithms for  $|\circ|$  and  $|\circ+$ -pattern matching are given in Chapter 3. In Chapter 4 we prove the conditional lower bounds for the hard patterns for pattern matching while the conditional lower bounds for membership are given in Chapter 5. We finally conclude the thesis in Chapter 6 by giving an overview of the previous results and the ones we showed in this thesis.

## Preliminaries

In the introduction we have already stated the problems we are considering in this thesis in an informal way. Hence, we give a formal definition of the pattern matching problem and the membership problem in Section 2.1. But first we define regular expressions and homogeneous patterns formally. In Section 2.2 we introduce the satisfiability problems FORMULA-SAT and FORMULA-PAIR. We also show relations between these problems and the corresponding hypotheses about their lower bounds. As a last part we state in Section 2.3 several already known techniques and methods which are used in a later part of the thesis. In the same section we also give a few definitions and notations we use throughout this thesis. An overview of widely used notation is given in the appendix.

### 2.1 Regular Expressions

#### 2.1.1 Basic Definitions

**Definition 2.1** (Regular Expressions and Languages). *Let  $\Sigma$  be a possibly infinite sized alphabet. We define the class of regular expressions over  $\Sigma$ ,  $\mathfrak{RE}_\Sigma$ , and the language of words accepted by a regular expression inductively.*

Base	Regular Expression	Language	Name
$\sigma \in \Sigma$	$\sigma \in \mathfrak{RE}_\Sigma$	$\mathcal{L}(\sigma) := \{\sigma\}$	Symbol
$p, q \in \mathfrak{RE}_\Sigma$	$(p \mid q) \in \mathfrak{RE}_\Sigma$	$\mathcal{L}(p \mid q) := \mathcal{L}(p) \cup \mathcal{L}(q)$	Alternative
$p, q \in \mathfrak{RE}_\Sigma$	$p \circ q \in \mathfrak{RE}_\Sigma$	$\mathcal{L}(p \circ q) := \{tu \mid t \in \mathcal{L}(p) \wedge u \in \mathcal{L}(q)\}$	Concatenation
$p \in \mathfrak{RE}_\Sigma$	$p^+ \in \mathfrak{RE}_\Sigma$	$\mathcal{L}(p^+) := \bigcup_{i=1}^{\infty} \mathcal{L}(\underbrace{p \circ \dots \circ p}_i)$	Kleene Plus <sup>1</sup>
$p \in \mathfrak{RE}_\Sigma$	$p^* \in \mathfrak{RE}_\Sigma$	$\mathcal{L}(p^*) := \{\varepsilon\} \cup \mathcal{L}(p^+)$	Kleene Star

We use  $\varepsilon$  to denote the empty string consisting of zero symbols.

We extend the definition of the alternative  $p \mid q$  and the concatenation  $p \circ q$  in the natural way to multiple sub-patterns. The symbol for the concatenation is usually omitted to simplify notation. In the following the size of the alphabet is bounded, strictly speaking

<sup>1</sup>Stephen Cole Kleene (1909–1994)

it is constant in most cases. Thus, we assume  $\varepsilon := |\Sigma|$  if not stated otherwise. We call regular expressions *patterns* in the following. We further omit the word “regular” since we only work with regular expressions and languages in this thesis. To simplify notation we write  $\Sigma^*$  for the language accepted by the pattern  $(\sigma_1 | \dots | \sigma_s)^*$ .

**Definition 2.2** (Membership Problem). *Let  $\Sigma$  be an alphabet.*

**Input:** *A text  $t \in \Sigma^*$  and a pattern  $p \in \mathfrak{RE}_\Sigma$ .*

**Task:** *Check whether  $t \in \mathcal{L}(p)$ .*

While for the membership problem the whole text must be matched, we mainly focus on the pattern matching problem. There we want to know whether there is a substring  $t'$  of  $t$  such that  $t' \in \mathcal{L}(p)$ . Thus, we define the matching-language of a pattern  $p$ :

**Definition 2.3** (Matching Language of a Pattern).  $\mathcal{M}(p) := \Sigma^* \mathcal{L}(p) \Sigma^*$

The following lemma shows that this definition agrees with our concept from above:

**Lemma 2.4.** *Let  $t = t_1 \dots t_n$  be a text of length  $n$  and  $p$  be a pattern, both over alphabet  $\Sigma$ . Then:*

$$t \in \mathcal{M}(p) \iff \exists 1 \leq i, i' \leq |n| : t_i \dots t_{i'} \in \mathcal{L}(p),$$

*with  $t_i \dots t_{i'} = \varepsilon$  if  $i' < i$ .*

This leads us to the definition of our main problem:

**Definition 2.5** (Pattern Matching Problem). *Let  $\Sigma$  be an alphabet.*

**Input:** *A text  $t \in \Sigma^*$  and a pattern  $p \in \mathfrak{RE}_\Sigma$ .*

**Task:** *Check whether  $t \in \mathcal{M}(p)$ .*

### 2.1.2 Homogeneous Patterns

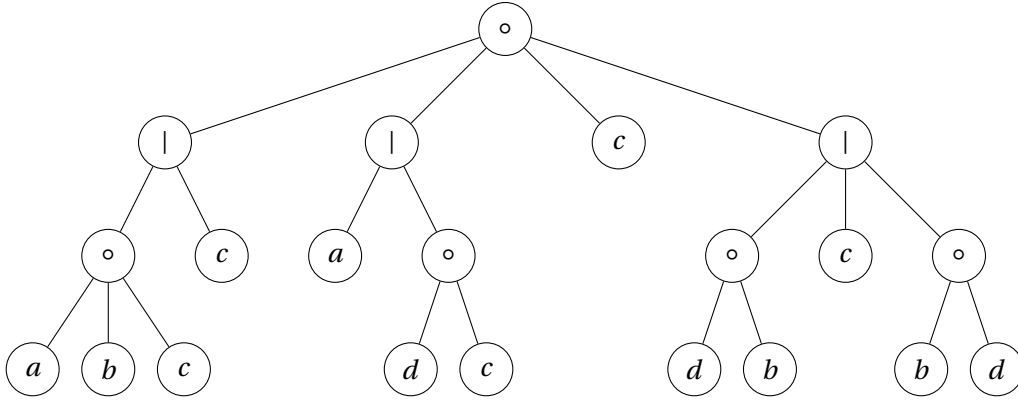
Analysing general patterns is quite difficult and finding faster algorithms or good lower bounds is even harder. Hence, we only work with so called *homogeneous* patterns as introduced by Backurs and Indyk in [BI16]. These patterns have a regular structure and are easier to analyse. Although faster algorithms for homogeneous patterns cannot be generalized for general patterns easily, the lower bounds also hold for the general patterns.

The intuitive definition is that we call a pattern *homogeneous* if the sub-patterns  $p$  and  $q$  as mentioned in Definition 2.1 are of the same type or they are symbols.

**Definition 2.6** (Homogeneous Patterns). *Treat the patterns as node-labeled trees, where the root of some subtree (i.e. an inner node) is labeled with the operation of the corresponding sub-pattern, and the leaves are labeled with symbols from the alphabet.*

- *Then a pattern is called homogeneous if for each level of the tree, the inner nodes are labeled with the same operation.*
- *The type of the pattern is the concatenation of these operations on the path from the root to the deepest inner node.*



FIGURE 2.1: The parse tree for the pattern  $(abc | c)(a | dc)c(db | c | bd)$ 

- The depth of a pattern is the depth of the tree, and thus the number of operations of the type (when collapsing equal operations into one).

We define the class of all homogeneous patterns of type  $T$  over alphabet  $\Sigma$  as  $\mathfrak{RE}_\Sigma(T)$ .

**Example 2.7.** The pattern  $(abc | c)(a | dc)c(db | c | bd)$  is homogeneous of depth 3 and has type  $o|o$ . See Figure 2.1 for the tree representation of the pattern.

We only consider homogeneous patterns in this thesis. Thus, we assume that patterns are homogeneous of the given type if not stated otherwise. As mentioned in the introduction it was shown in [BI16; BGL17] that the hardness of the patterns for the pattern matching problem reduces to pattern types of depth three and one pattern type of depth two (i.e.  $o\star$ ). Thus the depth is usually at most three. For the membership problem also few patterns of depth four have to be taken into account.

### 2.1.3 The Size of Patterns

When considering the running time of an algorithm it is important to know the size of the input. While it is easy to define the size for basic inputs as integers for example (use the length of the binary encoding of the actual number), it is not that easy for more complex inputs. Of course one could also use the bit-size of the input, but this is maybe rather complicated since the algorithm can process several bits at once. Furthermore, this type of measurement makes the analysis of the algorithm very tedious since every bit action has to be considered. Thus a good measurement should be defined such that it is still “close” to the bit-size but should also allow easier analyses of the algorithm.

**Definition 2.8** (Size of Patterns). *The size of a pattern is defined as the number of nodes (inner nodes plus leaves) in its parse tree.*

One can easily see that this definition only differs from the actual bit size by a factor of  $\mathcal{O}(\log s)$  since we need these many bits to encode the symbols. To encode the four operators and the possibility that the pattern is just a symbol we need three additional

bits for each node in the tree. But this is constant and thus exceeded by the multiplicative  $\mathcal{O}(\log s)$  factor.

We assume in the following that the size of the alphabet is constant. This assumption holds for the reductions in Chapter 4 and is used in the algorithms in Chapter 3. The application of the polynomial method for the upper bounds also works if the size is super constant but not too large. While for  $|\circ|$  we require that the size is rather small, i.e. in  $o(2^{\Theta(\sqrt{\log n})})$ , we can allow  $s \in \mathcal{O}(n)$  for  $|\circ+|$  and still get a runtime improvement.

## 2.2 Satisfiability Problems

### 2.2.1 Definition of the Problems

Instead of using the standard 3-SAT problem where one is given a Boolean formula in conjunctive normal form with clauses of three literals and one wants to find a satisfying assignment, we focus on two different satisfiability problems. We do not focus on this restricted class of formulas but on general formulas that are given as circuits. Such a formula is a node-labeled tree where each inner node computes a boolean function on at most two bits and the leaves are labeled with input variables or their negations. This especially implies that the output of each inner node, we call them gates from now on, is used only once since the out-degree is 1. The task is to check whether there is an assignment to the variables such that the formula evaluates to true. The *size* of a formula is defined as the number of *leaves* in the corresponding tree.

In the remaining part of this thesis we solely focus on *De Morgan formulas*.<sup>2</sup> Thus, a formula is De Morgan if not stated otherwise. These are formulas as described above but each inner gate is computing the Boolean AND, OR or NOT.<sup>3</sup> W.l.o.g. we can assume that no gate computes NOT because we can “push” these negations by De Morgans laws to the leaves of the formula. By this we still allow a leaf to be labeled with a variable or its negation. Since the out-degree of each gate is 1, the size of the formula does not change. We call a De Morgan formula *monotone* if the leaves are only labeled with variables, i.e. the negation of a variable is not allowed.

Let  $\mathbb{B} := \{\text{true}, \text{false}\}$  be the set of boolean values from now on. We identify true with 1 and false with 0 if necessary.

**Definition 2.9** (FORMULA-SAT ([AB18])).

**Input:** A (De Morgan) formula  $F$  of size  $s$  on  $n$  variables (also called inputs).

**Task:** Check whether there is an input in  $\mathbb{B}^n$  that makes  $F$  output true.

A problem closely related to FORMULA-SAT is the FORMULA-PAIR problem, as defined in Appendix B of the full version of [AB18]. This is the problem we base our reductions for the lower bounds on.

**Definition 2.10** (FORMULA-PAIR).

**Input:** A monotone De Morgan formula  $F = F(x_1, \dots, x_k, y_1, \dots, y_k)$  of size  $2k$  where each

---

<sup>2</sup>Augustus De Morgan (1806–1871)

<sup>3</sup>George Boole (1815–1864)

input is used exactly once and  $A, B \subseteq \mathbb{B}^k$  of size  $n$  and  $m$ , respectively.

**Task:** Check whether there are  $a \in A$  and  $b \in B$  such that

$$F(a, b) = F(a_1, \dots, a_k, b_1, \dots, b_k) = \text{true}.$$

The FORMULA-SAT and FORMULA-PAIR are closely related as the following lemma shows:

**Lemma 2.11** (Weak version of Lemma B.2 in the full version of [AB18]). *An instance of FORMULA-SAT on a De Morgan formula of size  $s$  over  $n$  inputs can be reduced to an instance of FORMULA-PAIR with two sets of size  $\mathcal{O}(2^{n/2})$  and a monotone De Morgan formula of size  $k = \mathcal{O}(s)$  in linear time.*

*Proof Idea.* Let  $F$  be the formula for FORMULA-SAT on  $n$  variables and size  $s$ . We define  $F'$  to be the same formula as  $F$  but each leaf is labeled with a different variable and we removes the negations from the leaves.

For all half-assignment  $x$  to the first half of inputs to  $F$  we construct a new half-assignment  $a_x$  for  $F'$  as follows. Let  $l$  be a leaf in  $F$  with a variable from the first half of inputs and let  $l'$  be the corresponding variable/leaf in  $F'$ . We set  $a_x[l'] = \text{true}$  if and only if  $l$  evaluates to true under  $x$ . We also construct the set  $B$  analogous for the second half of inputs of  $F$ .

Since  $F$  has  $n$  inputs this results in  $2^{n/2}$  assignments for  $A$  and  $B$ . □

## 2.2.2 Hypotheses

To prove *upper bounds* for problems it is sufficient to give an algorithm solving the problem in the stated runtime, even though it is of course not trivial to find these algorithms. But even for well studied problems like matrix multiplication, only very weak *lower bounds* are known. Ran Raz showed a lower bound of  $\Omega(n^2 \log n)$  [Raz02] while the best algorithms run in time  $\mathcal{O}(n^{2.38})$  as shown by Le Gall [Gal14]. To close this gap, one started to show *conditional lower bounds* such that under some assumption a better runtime cannot be achieved. But still these conditional lower bounds do not close the gap completely even if under these assumptions the gap may closes (up to “small” improvements). Nevertheless these bounds are conditional and may can be disproved, although this is very unlikely since widely-believed conjectured are used. The most famous conjecture is for the classical SATISFIABILITY PROBLEM:

**Hypothesis 1** (STRONG EXPONENTIAL TIME HYPOTHESIS (SETH) [Wil14a; CIP09; IP01]). *For every  $\delta < 1$ , there is a  $k \geq 3$  such that satisfiability of  $k$ -CNF formulas on  $n$  variables requires more than  $\mathcal{O}(2^{\delta n})$  time.*

See [BK15; Cyl+12; Bri14; AW14] for several examples of conditional lower bounds based on SETH. Furthermore, in [BI16; BGL17] conditional lower bounds based on SETH (or more precisely the implied conjecture for ORTHOGONAL VECTORS) for the pattern matching and membership problem are shown, ruling out faster than quadratic time algorithms for homogeneous patterns of bounded depth. But in this thesis we do not base our lower bounds on SETH but on an even stronger hypothesis that characterizes

the FORMULA-SAT problem and hence the FORMULA-PAIR problem. This is the same assumption as for the lower bounds shown in [Abb+16; AB18] which we use as a basis for our reductions.

WHY A DIFFERENT ASSUMPTION? For SETH based lower bounds usually an improvement by polynomial factors is ruled out. But the following hypothesis can be used to rule out not only these polynomial improvements but also lower order improvements. As already mentioned we show that the quadratic running time for pattern matching cannot be improved by more than a constant number of log-factors for several pattern types. This is a way tighter bound than the ones that are based on SETH.

**Hypothesis 2** (FORMULA-SAT HYPOTHESIS (FSH) [AB18]). *There is no algorithm that can solve FORMULA-SAT on De Morgan formulas of size  $s = n^{3+\Omega(1)}$  in  $\mathcal{O}(2^n / n^\epsilon)$  time, for some  $\epsilon > 0$ , in the Word-RAM model.*

But instead of directly using the FSH for the lower bounds, we define the corresponding hypothesis for FORMULA-PAIR and show that the later one is implied by FSH.

**Hypothesis 3** (FORMULA-PAIR HYPOTHESIS (FPH)). *For all  $k \geq 1$ , there is no algorithm that can solve FORMULA-PAIR for a monotone De Morgan formula  $F$  of size  $s$  and sets  $A, B \subseteq \mathbb{B}^{s/2}$  of size  $n$  and  $m$ , respectively, in time  $\mathcal{O}(nms^k / \log^{3k+2} n)$  in the Word-RAM model.*

**Lemma 2.12.** *FSH implies FPH.*

*Proof.* Assume FSH holds and FPH is false for some fixed  $k \geq 1$ .

Let  $F$  be a formula for FORMULA-SAT on  $N$  inputs and size  $s = N^{3+1/(4k)} \in N^{3+\Omega(1)}$ . By Lemma 2.11 we transform  $F$  into a monotone De Morgan formula  $F'$  of size  $s' = \mathcal{O}(s)$  and two sets with  $n, m \in \mathcal{O}(2^{N/2})$  assignments. We run the algorithm for FORMULA-PAIR now on this instance:

$$\begin{aligned} \mathcal{O}\left(\frac{n \cdot m \cdot s'^k}{\log^{3k+2} n} \log^{1+o(1)} 2^N\right) &\subseteq \mathcal{O}\left(\frac{2^{N/2} 2^{N/2} s^k N^{1.25}}{\log^{3k+2} 2^{N/2}}\right) \\ &= \mathcal{O}\left(2^N \frac{N^{3k+0.25+1.25}}{N^{3k+2} (1/2)^{3k+2}}\right) \\ &= \mathcal{O}\left(2^N \frac{N^{3k+1.5}}{N^{3k+2}}\right) \\ &= \mathcal{O}\left(\frac{2^N}{N^{0.5}}\right) \end{aligned}$$

But this contradicts the FSH.

The last factor  $N^{1+o(1)}$  comes from the change in the computation model we use. See the following paragraph for more information. We bounded the  $o(1)$  term by  $1/4$  which holds for sufficiently large  $N$ .  $\square$

**THE COMPUTATIONAL MODEL** Precisely as in [AB18] we use the *Word-RAM* model as our computational model. The word size of the machine will be fixed to  $\Theta(\log N)$  many bits for input size  $N$ . Similar as in the paper we assume several operations that can be performed in time  $\mathcal{O}(1)$  (e.g. AND, OR, NOT, addition, multiplication, ...).

While this is sufficient for our reductions, we need some more assumptions to state the FPH as above. There we also need that the operations are robust to a change of the word size: When considering a model with word size of  $\Theta(\log \log N)$  bit, we are able to split the original word into  $\Theta(\log N / \log \log N)$  chunks of size  $\Theta(\log \log N)$  and to perform the operations on these smaller chunks. Or to phrase it in the other way as mentioned by Abboud and Bringmann, we can simulate the operations on words of size  $\Theta(\log N)$  on a machine with word size  $\Theta(\log \log N)$  in time  $(\log N)^{1+o(1)}$ .

For the above reasoning that FPH is a reasonable assumption we saw that in the reduction from FORMULA-SAT to FORMULA-PAIR the size of the input increased from  $N$  to  $n = 2^N$ . To give a faster algorithm for FORMULA-SAT we cannot make use of a machine that has word size  $\log n = N$  but we have to use a word size of  $\log N$ . Thus, we have to simulate the operation on  $\log n = N$  bits on a machine with word size  $\log \log n = \log N$  bits. By this simulation the running time slows down by a factor of  $(\log n)^{1+o(1)} = N^{1+o(1)}$ .

## 2.3 Known Results and Definitions

During our reductions and for the algorithm design we make use of several useful facts and results. We state these (partly well known results) here in a precise way and derive corollaries which are used in the later reductions.

**FAST RECTANGULAR MATRIX MULTIPLICATION** While there are many improvements for the standard matrix multiplication, there is also a result about the multiplication of rectangular matrices. Coppersmith showed that in special cases these matrices can be multiplied faster than the standard approach, i.e. in almost quadratic time.

**Lemma 2.13** ([Cop82] (cf. Lemma 2.1 in [AWY15])). *For all sufficiently large  $N$ , the multiplication of a  $N \times N^{0.172}$  matrix with a  $N^{0.172} \times N$  matrix can be done in  $\mathcal{O}(N^2 \log^2 N)$  arithmetic operations.*

We can use this result to obtain a fast algorithm to evaluate polynomials on many inputs. We first show the balanced case of this result and then derive the general case from it:

**Lemma 2.14** (Lemma 2.2 in [AWY15]). *Given a polynomial  $Q(x_1, \dots, x_{d_1}, y_1, \dots, y_{d_2})$  over  $\mathbb{F}_2$  on at most  $n^{0.1}$  monomials. Let  $A \subseteq \mathbb{B}^{d_1}$  and  $B \subseteq \mathbb{B}^{d_2}$  consist of  $n$  elements each. We can evaluate  $Q$  on all pairs  $(a, b) \in A \times B$  in  $\mathcal{O}(n^2 \log^2 n + n^{1.1}(d_1 + d_2))$  time.*

*Remark.* The additional term  $n^{1.1}(d_1 + d_2)$  comes from the fact that we have to evaluate each monomial of  $Q$  on the values from  $A$  and  $B$ . Furthermore, the dimensions of  $A$  and  $B$  do not have to be equal.

**Lemma 2.15** (Fast Polynomial Evaluation). *Given a polynomial  $Q(x_1, \dots, x_{d_1}, y_1, \dots, y_{d_2})$  over  $\mathbb{F}_2$  on at most  $n^{0.1}$  monomials. Let  $A \subseteq \mathbb{B}^{d_1}$  and  $B \subseteq \mathbb{B}^{d_2}$  with  $n$  and  $m \geq n$  elements,*

respectively. We can evaluate  $Q$  on all pairs  $(a, b) \in A \times B$  in  $\mathcal{O}(nm \log^2 n + n^{0.1} m(d_1 + d_2))$  time.

*Proof.* We partition  $B$  as follows such that  $|B_i| = n$  for  $i \in [m/n - 1]$  and  $|B_{m/n}| \leq n$ :

$$B = B_1 \dot{\cup} B_2 \dot{\cup} \dots \dot{\cup} B_{\lceil m/n \rceil - 1} \dot{\cup} B_{\lceil m/n \rceil}$$

We can fill  $B_{\lceil m/n \rceil}$  with dummy values such that it always has size  $n$ , without affecting the overall running time. Then we can apply Lemma 2.14 with  $Q$  and  $A$  and for each  $B_i$ .

$$\begin{aligned} \mathcal{O}\left(\left\lceil \frac{m}{n} \right\rceil \left(n^2 \log^2 n + n^{1.1}(d_1 + d_2)\right)\right) &= \mathcal{O}\left(\frac{m}{n} \left(n^2 \log^2 n + n^{1.1}(d_1 + d_2)\right)\right) \\ &= \mathcal{O}\left(nm \log^2 n + n^{0.1} m(d_1 + d_2)\right) \quad \square \end{aligned}$$

**Definition 2.16** (Probabilistic Polynomial). *Let  $\mathcal{A}$  and  $\mathcal{B}$  be domains of values, and  $P$  be a predicate on values from  $\mathcal{A}$  and  $\mathcal{B}$ . Furthermore, let  $\psi : \mathcal{A} \rightarrow \mathbb{B}^a$  and  $\phi : \mathcal{B} \rightarrow \mathbb{B}^b$  be functions converting values from  $\mathcal{A}$  and  $\mathcal{B}$  to some binary representation with  $a$  and  $b$  bits, respectively. Let  $Q_1, \dots, Q_k$  be Boolean polynomials of degree at most  $a + b$  on Boolean variables  $\alpha_1, \dots, \alpha_a$  and  $\beta_1, \dots, \beta_b$ . Let  $q$  be a Boolean polynomial on  $k$  variables and  $R_1, \dots, R_k$  be Boolean random variables.*

We define

$$Q_{R_1, \dots, R_k}(\alpha_1, \dots, \alpha_a, \beta_1, \dots, \beta_b) := q(R_1 \cdot Q_1(\alpha_1, \dots, \beta_b), \dots, R_k \cdot Q_k(\alpha_1, \dots, \beta_b))$$

and say  $Q$  is a probabilistic polynomial that agrees with  $P$  with probability at least  $2/3$  if the following holds:

$$\forall A \in \mathcal{A}, B \in \mathcal{B} : \Pr_{r_1, \dots, r_k} [Q_{r_1, \dots, r_k}(\psi(A), \phi(B)) = \llbracket P(A, B) \rrbracket] \geq 2/3$$

We say  $Q$  has error probability at most  $1/3$ .

**POLYNOMIAL METHOD** We use the polynomial method together with the polynomial evaluation as defined above to show faster algorithms in Chapter 3. While the method was initially used to prove lower bounds for circuits, it was later adapted by Williams to improve the running time of algorithms [Wil14b]. The main idea is the transformation of an AND gate into a probabilistic polynomial that is correct with a certain probability. The original idea is due to Razborov [Raz87] and Smolensky [Smo87]:

**Definition 2.17.** *Let  $t$  and  $d$  be integers. Choose  $t \cdot d$  many random bits  $r_{i,j}$  uniformly at random with  $i \in [t], j \in [d]$ . Define*

$$A_t(y_1, \dots, y_d) := \prod_{i=1}^t \left( 1 \oplus \bigoplus_{j=1}^d r_{i,j} (1 \oplus y_j) \right)$$

**Lemma 2.18** (Lemma 2.3 in [AWY15]). *For every fixed  $y_1, \dots, y_d \in \mathbb{B}$ :*

$$\Pr_{r_{i,j}} \left[ A_t(y_1, \dots, y_d) = \bigwedge_{j=1}^d y_j \right] \geq 1 - 1/2^t$$

**DEPTH-REDUCTION TECHNIQUES** During the reductions for the lower bound we will see that the size of the output depends exponentially on the depth of the formula and polynomially on the size. To avoid this blow-up we make use of the following depth reduction technique:

**Lemma 2.19** (Theorem 4 in [BB94]). *Let  $C$  be a  $\{\wedge, \vee, \neg\}$ -formula of size  $m$ . Then for all  $k \geq 2$ , there is an equivalent  $\{\wedge, \vee, \neg\}$ -formula  $C'$  of depth  $d(C') \leq (3k \ln 2) \cdot \log m \leq 2.08k \log m$  such that  $\text{size}(C') \leq m^\alpha$ , where  $\alpha = 1 + (1 + \log(k - 1))^{-1}$ .*

Where the depth of a formula is defined as follows:

**Definition 2.20** (Depth of a tree). *Let  $T$  be a tree with root  $r$ . The depth of  $T$  is defined as the length of the longest path from  $r$  to any leaf:*

$$d(T) := \max_{u: \text{leaf in } T} |r \rightarrow_T u|$$

*Hence, every leaf has height 0.*





---

## Upper Bounds by Faster Algorithms

In this chapter we prove the following theorem by showing two fast algorithms solving the matching problem for patterns of type  $|\circ|$  and  $|\circ+$  with a super-poly-logarithmic runtime improvement.

**Theorem 3.1** (Faster Algorithms for  $|\circ|$  and  $|\circ+$ ). *Pattern matching on texts of length  $n$  and homogeneous patterns of size  $m$  and type  $|\circ|$  or  $|\circ+$  can be solved in time*

$$\mathcal{O}\left(\frac{nm}{2^{\Theta(\sqrt{\log \min(n,m)})}}\right)$$

*by a randomized algorithm with small error probability.*

The improvements of our algorithms are still sub-linear and thus not polynomial in the input size. This respects the quadratic time lower bounds which are shown for quite a couple of pattern matching problems, including these two pattern types. See [BI16; BGL17] for more information and the reductions.

Our result makes use of the polynomial method which was originally used in the field of circuit complexity. Recently it was applied in algorithm design to give faster algorithms for ALL-PAIRS SHORTEST PAIRS, ORTHOGONAL VECTORS and LONGEST COMMON SUBSTRING WITH DON'T CARES (see [AWY15; CW16; Wil14b] for further information).

In the following we first give the basic idea behind the algorithm and show the similar parts for both pattern types. In Section 3.2 we show the more involved part for patterns of type  $|\circ|$  while in Section 3.3 we show the corresponding result for patterns of type  $|\circ+$ .

### 3.1 General Idea

Recall, that for pattern matching we are given a text  $t$  of length  $n$  and a pattern  $p$  of size  $m$  (cf. Section 2.1.3). We first observe that it is very easy to split patterns of type  $|\circ|$  and  $|\circ+$  into smaller sub-patterns which can be handled independently from each other. The general idea of the procedure is as follows:

1. Define a threshold  $f \in 2^{\Theta(\sqrt{\log n})}$  depending on the pattern type.<sup>1</sup>

---

<sup>1</sup>Even though the improvement of the running time depends on the minimum of  $n$  and  $m$ , this threshold  $f$  only depends on  $n$ .

2. Split the pattern into *large* sub-patterns matching  $> f$  symbols and *small* sub-patterns matching  $\leq f$  symbols. These sub-pattern have type  $\circ|$  or  $\circ+$ .
3. Solve the problem for all large patterns sequentially, i.e. try all large sub-patterns and the text as a pattern matching instance.
4. For the small sub-patterns do the following:
  - Divide the text into pieces of size  $2f$ .
  - Design a circuit checking for a sub-pattern  $q$  and a text piece  $u$  whether there is a shift  $0 \leq i < f$  such that the  $j$ th part of the pattern is matched to the  $i + j$ th symbol of the text for all  $j \in [|q|] \subseteq [f]$ .
  - Transform this circuit into a probabilistic polynomial by the polynomial method introduced by Razborov and Smolensky [Raz87; Smo87].
  - Use the fast polynomial evaluation, based on the matrix multiplications by Coppersmith, from Lemma 2.15 to evaluate the polynomial on all inputs and get the required result. (This is possible due to a clever choice of the parameters.)
5. Repeat this last step, while the text pieces are shifted by  $f$  symbols, i.e. the first text piece starts at the  $f + 1$ th symbol of the text  $t$  and not the first as before.

*Remark.* We are only interested whether there is *some* substring that can be matched by the pattern. We do not ask for *all* possible substrings that can be matched by the pattern. Therefore, it only suffices to check whether there is a shift  $i$  as mentioned above. We use the same argument for the proof of Lemma 3.2 below.

If we are interested in finding all possible substring, then we have to change our algorithm (and circuit) and use a different approach. But then also the analysis of the running time becomes more involved since the algorithm has to output all substrings. Since there can be  $\mathcal{O}(n^2)$  many matched substring even for small patterns, this would change our running time significantly. But see the discussion about the running time in Chapter 5 in [AC75] for further information.

**LARGE SUB-PATTERNS** Since the large sub-patterns have to be of a certain size, there cannot be too many of them. Thus, we can test each sub-pattern against the whole text by the near-linear time algorithm for  $\circ|$  and  $\circ+$ -matching.

**Lemma 3.2** (Large Sub-Patterns). *Given a text  $t$  of length  $n$  and a pattern  $p$  of size  $m$ . Furthermore, let  $p$  be of type  $\circ|$  or  $\circ+$ . If all sub-patterns of  $p$  have size at least  $f$  and at most  $n$ , then we can check  $t \in \mathcal{M}(p)$  in time  $\mathcal{O}\left(f^{-1} nm \log^2 n\right)$ .*

*Proof.* From [CH02; BI16] we know that pattern matching of type  $\circ|$  and  $\circ+$  can be solved in time  $\mathcal{O}(n \log^2 m' + m')$  with patterns of size  $m'$ . We set  $m_i$  to be the length of the  $i$ th sub-pattern of  $p$ . Hence,  $m$  is the sum of the  $m_i$ 's plus 1. By assumption there are  $l \leq m/f$  many sub-patterns. This gives us an overall running time of:

$$\mathcal{O}\left(\sum_{i=1}^l n \log^2 m_i + m_i\right) \subseteq \mathcal{O}\left(l \cdot n \log^2 n + m\right) \subseteq \mathcal{O}\left(\frac{m}{f} n \log^2 n\right) = \mathcal{O}\left(\frac{nm \log^2 n}{f}\right) \quad \square$$

**SMALL SUB-PATTERNS** While this approach works for large sub-patterns we cannot use it for the small patterns because there can be too many of them. For these small patterns we use the polynomial method to check for all small sub-patterns whether they can be matched to some part of the text.

**Lemma 3.3** (Small Sub-Patterns). *Given a text  $t$  of length  $n$  and a pattern  $p$  of size  $m$  such that  $n \leq 2m/s$ . Furthermore, let  $p$  be of type  $|\circ|$  or  $|\circ+$ . If all sub-patterns have size at most  $f$  for  $f \in 2^{\Theta(\sqrt{\log n})}$ , then we can check whether  $t \in \mathcal{M}(p)$  in time  $\mathcal{O}(nm/2^{\Theta(\sqrt{\log n})})$  with high probability.*

Following the ideas above, it actually suffices to prove the following two lemmas. Then we can combine them with already known results to obtain the previously stated lemma.

**Definition 3.4** (Characteristic Vectors). *Let  $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_s\}$  be the alphabet with some (fixed) ordering of the symbols. We define  $\chi: \mathfrak{A}_{\Sigma}(|) \rightarrow \mathbb{B}^s$  as the characteristic function for patterns of type  $|$  (this includes single symbols):*

- $\chi(\sigma_i)[i] := 1$  and  $\chi(\sigma_i)[j] = 0$  for  $i \neq j \in [s]$
- $\chi(\sigma_{i_1}|\sigma_{i_2}|\dots|\sigma_{i_k})[j] = 1$  for all  $j \in \{i_1, i_2, \dots, i_k\}$  and  $\chi(\sigma_{i_1}|\sigma_{i_2}|\dots|\sigma_{i_k})[j] = 0$  else.

We extend this definition to patterns of type  $\circ|$  and thus also “normal” texts in the natural way by applying  $\chi$  to every sub-pattern (or symbol).

**Lemma 3.5.** *There is an  $f$  and a probabilistic polynomial (cf. Definition 2.16)  $Q$  on  $3fs$  variables with at most  $(n/2f)^{0.1}$  monomials.  $Q$  agrees with the following property with probability  $\geq 2/3$ :*

$$u \in \mathcal{M}(p) \iff Q(\chi(u), \chi(p)) = \text{true}$$

Where  $u$  is a text piece of length  $2f$  and  $p$  a pattern of type  $\circ|$  and size at most  $f$ .

For the characteristic vector  $\chi'$  used in the following lemma, we do not use a 1-hot encoding as above but we encode the index of the symbol in binary. A precise definition is given in the second paragraph of Section 3.3.

**Lemma 3.6.** *Let  $L := \lfloor \log \min(n, m+1) \rfloor + 1$ . The following exists:*

- An  $f$  such that we can define  $t := \log(24(L+1)f^2 \log(24f)) \leq \log(24(L+1)f^3)$ ,
- A probabilistic polynomial  $Q$  on  $(L+1)t$  variables with at most  $(n/2f)^{0.1}$  monomials,
- Functions  $T_{h,v,r}$  and  $P_{h,v,r}$  computable in time linear in the input size for  $v \in [t]$ ,  $h \in [L+1]$ .

$Q$  agrees with the following property with probability  $\geq 5/8$ :

$$u \in \mathcal{M}(p) \iff Q(T(\chi'(u)), P(\chi'(p))) = \text{true}$$

Where  $u$  is a text piece of length  $2f$  and  $p$  is a pattern of type  $\circ+$  and size at most  $f$ .

By combining these lemmas with Lemma 2.15 we can prove Lemma 3.3.

*Proof of Lemma 3.3.* Let  $Q$  be the polynomial and  $f$  the threshold from Lemma 3.5 and Lemma 3.6. Let  $p_1, \dots, p_l$  be the sub-patterns of the given pattern  $p$  matching  $\leq f$  symbols each. Assume  $n = k2f + f$  for some arbitrary  $k \in \mathbb{N}$ . Otherwise, append the text with fresh symbols not occurring in the pattern. Set  $n$  to be the length of the new text (this only differs by at most  $2f - 1$  from the old length). Now split the text  $t = t_1 \cdots t_n$  into  $a_i := t_{i2f-2f+1} \cdots t_{i2f}$  and  $b_i := t_{i2f-f+1} \cdots t_{i2f+f}$  for  $1 \leq i \leq \lfloor n/2f \rfloor$ .

$Q$  has at most  $(n/2f)^{0.1}$  monomials. We know that we can transform the  $a_i$ 's and  $p_i$ 's into two sets  $V$  and  $W$  of boolean vectors by the functions stated in the lemmas, i.e. by  $\chi$  for  $|\circ|$  and by  $\chi', T$ , and  $P$  for  $|\circ+$ . We also repeat the following procedure with the  $b_i$ 's instead of the  $a_i$ 's. We remark that  $|V| = n/2f$  and  $m/sf \leq |W| \leq m$  and hence by assumption  $|V| \leq |W|$ . Now we apply Lemma 2.15 with the dimensions set appropriately for each pattern type and get a running time of

$$\mathcal{O}\left(\frac{n}{2f} m \log^2 \frac{n}{2f}\right) \subseteq \mathcal{O}\left(\frac{nm \log^2 n}{2^{\Theta(\sqrt{\log n})}}\right) \subseteq \mathcal{O}\left(\frac{nm}{2^{\Theta(\sqrt{\log n})}}\right)$$

To reduce the error probability of the polynomial we can use the standard boosting technique by sampling  $\Theta(\log n)$  many probabilistic polynomials. For each sampled polynomial we check whether all entries are false. Since the polynomial does not produce false negatives, we continue with the  $b_i$ 's in this case.<sup>2</sup> By using a Chernoff bound one can show that the error is at most  $\text{poly } n^{-1}$  (see Section 7.4.1 in [AB09] for details).

Because the  $p_i$  can match at most  $f$  symbols and the texts have length  $2f$ , we recognize if the pattern matches a text within this text piece. But we also have to check if the text matched by a pattern is overlapping two  $a_i$ 's. Therefore, we shift the text chunks by  $f$  symbols in the second round, i.e. we use the  $b_i$ 's instead of the  $a_i$ 's.  $\square$

**PUTTING EVERYTHING TOGETHER** Using the above lemmas we can show our main theorem of this chapter by giving fast algorithms for the two pattern matching cases:

*Proof of Theorem 3.1.* Let  $t$  be the text of length  $n$  and  $p$  be the pattern of size  $m$ . Choose  $f$  as in the corresponding lemma above according to the type of the pattern, i.e.  $f \in 2^{\Theta(\sqrt{\log n})}$ .

Let  $p_{\leq}$  and  $p_{>}$  be the partitioning of  $p$  depending on the number of symbols/tuples the sub-patterns can match (i.e. they can match  $\leq f$  or  $> f$  symbols/tuples). Let  $m_{\leq}$  and  $m_{>}$  be the size of the corresponding patterns. We handle both patterns independently from each other:

$p_{>}$  We first observe that we can ignore all sub-patterns of  $p_{>}$  with size  $> \mathcal{O}(n)$  since this means that more than  $n$  symbols have to be matched which is not possible. We apply Lemma 3.2 to solve the problem in the following time:

$$\mathcal{O}\left(\frac{nm_{>} \log^2 n}{f}\right)$$

<sup>2</sup>This follows from the definition of the polynomial  $A_t$  in Definition 2.17.

$p_{\leq}$  We distinguish between the case  $n \leq 2m_{\leq}/s$  and  $n > 2m_{\leq}/s$ .

$n \leq 2m_{\leq}/s$  We directly apply Lemma 3.3 and get a running time of

$$\mathcal{O}\left(\frac{nm_{\leq}}{2^{\Theta(\sqrt{\log n})}}\right)$$

$n > 2m_{\leq}/s$  We partition the text into  $sn/2m_{\leq}$  pieces of size  $n' = 2m_{\leq}/s$  and pad the text with fresh symbols if  $2m_{\leq}/s$  does not divide  $n$ . In a second run we shift the text pieces by  $m_{\leq}/s$  symbols, similarly as in the proof of Lemma 3.3. We use the algorithm from that lemma to get a total running time of

$$\mathcal{O}\left(\frac{sn}{2m_{\leq}} \cdot \frac{n' \cdot m_{\leq}}{2^{\Theta(\sqrt{\log n'})}}\right) = \mathcal{O}\left(\frac{nm_{\leq}}{2^{\Theta(\sqrt{\log m_{\leq}})}}\right)$$

for this case. Note that this is an improvement not depending on  $n$  but on  $m$ .

By these case distinctions we get an overall running time of

$$\mathcal{O}\left(\frac{nm}{2^{\Theta(\sqrt{\log \min(n,m)})}}\right)$$

where we assumed in all steps that the size of the alphabet is constant.  $\square$

*Remark.* The improvement in Theorem 3.1 is not depending on  $n$  but on the minimum of  $n$  and  $m$ . Why this is still satisfying, can be seen by the following observation: Assume we have a large text and a small pattern of possibly constant size. If the algorithm would have an improvement only depending on  $n$ , we could achieve a sub-linear time algorithm. But this is not possible since we have to read the input at least once and thus we have a linear lower bound. The analogous argument hold for the other direction. Hence, the improvement must depend on the smaller of the two parameters specifying the input size.

Now it remains to prove Lemma 3.5 and 3.6. We show each of them in a separate section in the following.

## 3.2 Small Patterns of Type $|\circ|$

Let  $u = u_1 \cdots u_{2f}$  be a text piece and  $q = q_1 \cdots q_f$  be a pattern of size  $f$  and type  $|\circ|$ . Recall, we assume the size of the alphabet  $\Sigma$  is constant and thus the size of each  $q_i$  is in  $\mathcal{O}(1)$ .

### 3.2.1 The Circuit

**MATCHING SYMBOLS** The following definition of the circuit  $C_{\varepsilon}$  on  $2s$  inputs satisfies the following property for all symbols  $a$  and patterns  $r$  of type  $|\circ|$

$$C_{\varepsilon}(\chi(a), \chi(r)) := \bigvee_{k=1}^s \chi(a)[k] \chi(r)[k]$$

$$C_{\varepsilon}(\chi(a), \chi(r)) = 1 \iff a \in \mathcal{L}(r)$$

The correctness of the circuit follows immediately by its construction and the definition of  $\chi$ , Definition 3.4.

**MATCHING WORDS** Recall, that the function  $\chi$  was not only defined for patterns of type  $|$  but it can easily be extended to patterns of type  $\circ|$  and thus texts of arbitrary size and length, respectively.

To check whether  $q$  matches  $u$ , we have to check whether there is a shift  $i$  of  $u$  such that the first alternative of the pattern matches the  $i + 1$ th symbol of the text. Then we check for the following  $f - 1$  symbols whether they match. This leads us to the definition of the circuit  $C_M$ :

$$C_M(\chi(u), \chi(q)) := \bigvee_{i=0}^{f-1} \bigwedge_{j=1}^f C_\epsilon(\chi(u_{i+j}), \chi(q_j)) = \bigvee_{i=0}^{f-1} \bigwedge_{j=1}^f \bigvee_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k]$$

### 3.2.2 The Polynomial

**DEFINITION** To obtain a polynomial over  $\mathbb{F}_2$  we have to replace the AND and OR by products and binary additions (i.e. addition modulo 2). For this we first notice that at most one product in  $C_\epsilon$  is true for all valid inputs. This is because we used one bit for each symbol and the characteristic vector of a text symbol contains exactly one 1. Therefore, we can replace the OR by an XOR:

$$C'_\epsilon(\chi(a), \chi(r)) := \bigoplus_{k=1}^s \chi(a)[k] \chi(r)[k] \equiv C_\epsilon(\chi(a), \chi(r))$$

By De Morgans laws we replace the outer OR in  $C_M$  by a NOT-AND-NOT and transform all  $\neg\phi$  into  $1 \oplus \phi$  for any formula  $\phi$ .

$$C_M(u, q) \equiv 1 \oplus \bigwedge_{i=0}^{f-1} \left( 1 \oplus \bigwedge_{j=1}^f \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k] \right)$$

Now we replace the outer AND by a polynomial  $A_2$  as defined in Definition 2.17 (using the results from Razborov and Smolensky [Raz87; Smo87]):

$$\begin{aligned} & 1 \oplus \prod_{i'=1}^2 \left( 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \left( 1 \oplus 1 \oplus \bigwedge_{j=1}^f \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k] \right) \right) \\ &= 1 \oplus \prod_{i'=1}^2 \left( 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \bigwedge_{j=1}^f \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k] \right) \end{aligned}$$

Finally we replace the inner AND by  $2f$  probabilistic polynomials  $A_{3\log f}$ :

$$\begin{aligned}
 & 1 \oplus \prod_{i'=1}^2 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \prod_{j'=1}^{3\log f} \left( 1 \oplus \bigoplus_{j=1}^f r_{i',i,j',j} \left( 1 \oplus \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k] \right) \right) \\
 &= 1 \oplus \prod_{i'=1}^2 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \prod_{j'=1}^{3\log f} \left( 1 \oplus \underbrace{\left( \bigoplus_{j=1}^f r_{i',i,j',j} \cdot 1 \right)}_{=: r_{i',i,j' \in \{0,1\}} (*)} \oplus \bigoplus_{j=1}^f \left( r_{i',i,j',j} \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k] \right) \right) \\
 &= 1 \oplus \prod_{i'=1}^2 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \prod_{j'=1}^{3\log f} \left( r_{i',i,j'} \oplus \bigoplus_{j=1}^f \left( r_{i',i,j',j} \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k] \right) \right) =: Q_r(u, q)
 \end{aligned}$$

Whether (\*) is equal to 0 or 1 only depends on the choice of the random bits. We can see the sum as some value  $r_{i',i,j'}$  that is specific for each  $i', i, j'$  and depends only on the corresponding bits  $r_{i',i,j',j}$ .

**ERROR BOUND** According to Definition 2.16 we want to show that for every input  $(u, q)$  the polynomial  $Q_r$  agrees with  $C_M$  with probability  $\geq 2/3$  or that the error probability is bounded by  $1/3$ . From Lemma 2.18 we know that the outer polynomial  $A_2$  is wrong with probability at most  $1/2^2 = 1/4$ . Each of the inner  $A_{3\log f}$  is wrong with probability at most  $1/2^{3\log f} = 1/f^3$ . This gives an overall error probability of  $\leq 1/4 + 2f/f^3$ . One can easily check that this is smaller than  $1/3$  for  $f \geq 5$ .

### 3.2.3 Counting the Monomials

it remains to bound the number of monomials of  $Q_r$ . For this we convert  $Q_r$  into sums of products by flipping the order of the operations.

**FIRST FLIP** We first flip the inner most sum and the middle product.

▷ **Claim 3.2.1.** For all  $i', i$ :

$$\prod_{j'=1}^{3\log f} r_{i',i,j'} \oplus \bigoplus_{j=1}^f \left( r_{i',i,j',j} \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k] \right)$$

has  $< 2 \binom{f+1}{3\log f} s^{3\log f}$  monomials for  $s \geq 2$ .

*Proof.* The main idea is to make use of the distributivity law. To keep the number of monomials small we use the same facts and ideas as in [AWY15]:

1.  $x^2 = x$  in  $\mathbb{F}_2$
2.  $3\log f \leq (f+1)/2$  for sufficiently large  $f$
3. Each of the  $3\log f$  sums is the sum of elements from  $\{1, \gamma_1, \dots, \gamma_f\}$  with

$$\gamma_j := \bigoplus_{k=1}^s \chi(u_{i+j})[k] \chi(q_j)[k]$$

Let us first treat the  $\gamma_j$  as variables in the following. Then, when applying the distributivity law, the number of monomials is bounded by  $\sum_{w=1}^{3\log f} \binom{f+1}{w}$ . This is because when expanding the product we have one summand for each  $j'$ . But these summands are from the set defined above and when we have the same summands (and thus factors) for two  $j'$ 's they reduce to one because we are in the binary setting (observation 1). By this we can use the number of subsets from  $\{1, \gamma_1, \dots, \gamma_f\}$  of size at most  $3\log f$  since there can be at most that many factors in each product.

But since the  $\gamma_j$  are no variables, we have to expand them, too. For each subset of size  $w$ , there are at most  $w$  many  $\gamma_j$ 's in each monomial. Since each  $\gamma_j$  has  $\mathfrak{s}$  summands, this results in at most  $\mathfrak{s}^w$  monomials for each monomial consisting of  $w$  many  $\gamma_j$ 's. Thus, the number of monomials is bounded by

$$\sum_{w=1}^{3\log f} \binom{f+1}{w} \mathfrak{s}^w \leq \sum_{w=1}^{3\log f} \binom{f+1}{3\log f} \mathfrak{s}^w \leq \binom{f+1}{3\log f} \mathfrak{s}^{\frac{3\log f - 1}{\mathfrak{s} - 1}} \leq 2 \binom{f+1}{3\log f} \mathfrak{s}^{3\log f - 1}$$

for sufficiently large  $f$  and observation 2.  $\square$

**SECOND FLIP** We observe that the initial “ $1\oplus$ ” can be ignored because we can interpret an output of 0 as true and 1 as false.

$\triangleright$  **Claim 3.2.2.** Let  $P_{i',i}$  be polynomials, each consisting of at most  $k$  monomials. Then,

$$\prod_{i'=1}^2 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} P_{i',i}$$

has  $\leq (fk+1)^2$  monomials.

*Proof.* By applying the distributivity law we get the claimed bound.  $\square$

The final polynomial has at most  $M = \left(2f \binom{f+1}{3\log f} \mathfrak{s}^{3\log f}\right)^2$  monomials by the two claims above.

**BOUNDING THE NUMBER OF MONOMIALS** It remains to show that  $M$  is smaller than  $(n/2f)^{0.1}$ . For this we set  $f := 2^{\sqrt{1/20 \cdot \log n / (2+3\log \mathfrak{s})} - 1}$ , i.e.  $f = 2^{\Theta(\sqrt{\log n})}$  when  $\mathfrak{s}$  is constant.

$$\begin{aligned} M \leq (n/2f)^{0.1} &\iff \sqrt{M} \leq (n/2f)^{1/20} \\ &\iff 2f \binom{f+1}{3\log f} \mathfrak{s}^{3\log f} \leq (n/2f)^{1/20} \\ &\iff 1 + \log f + \log \binom{f+1}{3\log f} + 3\log f \log \mathfrak{s} \leq 1/20 \cdot (\log n - \log(2f)) \\ &\iff \Gamma := 1 + \log f + \log \binom{f+1}{3\log f} + 3\log f \log \mathfrak{s} + 1/20 \cdot \log(2f) \leq 1/20 \cdot \log n \end{aligned}$$



We show the last inequality:

$$\begin{aligned}
 \Gamma &= 1 + \log f + \log \binom{f+1}{3 \log f} + 3 \log f \log \mathfrak{s} + \frac{\log(2f)}{20} \\
 &\leq \log(f+1) + 3 \log f \log(f+1) + 3 \log(f+1) \log \mathfrak{s} + \log f \\
 &\leq (2 + 3 \log \mathfrak{s}) \log(f+1) + 3 \log^2(f+1) \\
 &\leq (2 + 3 \log \mathfrak{s}) \frac{\sqrt{\frac{1}{20} \cdot \log n}}{2 + 3 \log \mathfrak{s}} + 3 \frac{\frac{1}{20} \cdot \log n}{(2 + 3 \log \mathfrak{s})^2} \\
 &\leq \sqrt{\frac{1}{20} \cdot \log n} + \frac{1}{2} \frac{1}{20} \cdot \log n \\
 &\leq \frac{1}{20} \log n
 \end{aligned}$$

For the first inequality we used that  $1 + \frac{\log(2f)}{20} \leq \log f$  and  $\binom{a}{b} \leq a^b$ . These two and all other bounds hold for sufficiently large  $n$  and thus  $f$ . ■

*Remark.* We still get the same asymptotic bound if  $\mathfrak{s}$  is not constant but also not too large. Strictly speaking, we require  $\mathfrak{s} \in o(2^{\Theta(\sqrt{\log n})})$  for the proof to work. For larger sizes the improvement cannot be achieved by this method anymore.

### 3.3 Small Patterns of Type $\circ+$

#### 3.3.1 Encoding Texts and Patterns

**PARTIAL INEQUALITIES** Recall the definitions of the size of patterns in Section 2.1.3. But instead of using this definition for patterns of type  $\circ+^3$  and hence  $\circ+$ , we define a compressed version of the patterns. By this the size of the patterns does not increase but may decrease and the information stays the same. Thus, we have an alternative way of representing the pattern.

We do this by merging consecutive sub-patterns of type  $+$  over the same symbol together. We know that a single symbol matches exactly one symbol while a Kleene Plus matches at least one symbol. Thus, we can sum these minimal numbers of repetitions that a symbol has to be matched. By this we represent patterns of type  $\circ+$  as lists of tuples with a symbol and a partial inequality.

**Example 3.7.** A pattern such as  $aaa^+b^+bc$  requires that  $\geq 3$  symbols  $a$ , then  $\geq 2$  symbols  $b$ , and finally  $= 1$  symbol  $c$  are matched. We represent this by the following tuple sequence  $(a, 3 \leq)(b, 1 \leq)(c, 1 =)$ .

**Definition 3.8** (Partial Inequality). Let  $\square \in \{\geq, =, \leq\}$ . We define a partial inequality  $c \square$  to be a function  $f: \mathbb{N} \rightarrow \mathbb{B}$  with  $n \mapsto \llbracket c \square n \rrbracket$ .

<sup>3</sup>The same procedure can be applied directly to patterns of type  $\circ*$ .

Since texts can be seen as patterns of type  $\circ+$  and hence  $\circ+$ , we can use the same compression but replacing a partial inequalities of the form  $c =$  by the integer  $c$ . Using this compression, we can reformulate the membership problem: A text  $t = (\sigma_1, l_1) \cdots (\sigma_k, l_n)$  is matched by a pattern  $p = (\tau_1, k_1) \cdots (\tau_k, k_n)$  if and only if  $\sigma_i = \tau_i$  and  $k_i(l_i) = \text{true}$  for all  $i \in [n]$ . Now it remains to show how the compression of the patterns and the texts can be encoded.

**ENCODING TEXTS AND PATTERNS** We identify the symbols with their index in  $\Sigma$ . To encode the indices in binary we need  $\lfloor \log s \rfloor + 1$  bits. Later we require that every binary encoding has at least one 1 and thus we cannot use index 0. Hence, we need  $S := \lfloor \log(s + 1) \rfloor + 1$  bits for this encoding.

We first find an upper bound for the number encoding the repetitions in the tuples of the text and the pattern. Recall, that every pattern of type  $\circ+$  with size  $m$  has  $\leq m$  leaves. Thus, it can match at most  $m$  symbols when not considering the  $+$ . So, every number that could appear in a partial equation is  $m$  at most. Now consider a text tuple  $(\sigma, l)$  with  $l > m$ . By the above observation this text has to make use of at least one  $+$  in the pattern, otherwise it could not be matched. But all of the  $l - m$  repetitions of the latter can be mapped to this specific  $+$ . Thus, we can reduce every number strictly larger than  $m$  to  $m + 1$ . We also observe that the number can never be larger than  $n$ , where  $n$  is the length of the text. Because if a pattern has to match  $> n$  repetitions of the same symbols, it requires a string longer than the text and thus it can be ignored. Therefore, we can use the minimum of  $n$  and  $m + 1$  as a bound for the number in the tuple. By the same observation we can assume that all sub-patterns of type  $\circ+$  match at most  $n$  tuples and symbols. Thus, we need  $L = \lfloor \log \min(n, m + 1) \rfloor + 1$  bits for this.

To encode the type of the partial equation  $c \square$  it suffices to consider the case  $\square \in \{\leq, =\}$ . We use a single bit  $e$  for this that is set to 0 for equality and to 1 for inequalities. To simplify notation we will treat  $e$  as an additional bit for the binary encoding of the number of repetitions at some point in the proof. Even though the text tuple does not encode a partial inequality, we also add this bit. Later we see why we set it to 1 for all tuples.

By these observations we encode each tuple of the text and the pattern by  $S + 1 + L$  bits.

*Remark.* We can also apply this type of compression when the sub-patterns of the Kleene Plus or Star are more complex. But in these cases it is not very likely that the sub-patterns are repeated in the concatenation more than once. Furthermore, since we are working with homogeneous patterns, these patterns are in general not be homogeneous anymore and can therefore not occur in our setting.

Let  $u = u_1 \cdots u_{2f}$  be a text piece and  $q = q_1 \cdots q_f$  be the pattern of size  $f$  and type  $\circ+$  after the above compression in the following.

### 3.3.2 The Circuit

**MATCHING TUPLES** Let  $v = (\sigma, l)$  be a text tuple and  $r = (d, \tau, e, l')$  be a pattern tuple in the following. Where the bit  $d$  is a new don't-care bit. We use it when the number of pattern tuples is not as large as the circuit we defined it for. Then we add new tuples with all bits set to 0 except for the bit  $d$  which is set to 1.

We define the circuit  $C_\epsilon((\sigma, l), (d, \tau, e, l'))$  such that it outputs true if one of the following conditions holds and false otherwise:

1.  $d = 1$
2.  $\sigma = \tau$ , and  $l = l'$  if  $e = 0$  and  $l \geq l'$  if  $e = 1$

We get:

$$\begin{aligned} C_\epsilon((\sigma, l), (d, \tau, e, l')) &= d \vee (\llbracket \sigma = \tau \rrbracket \wedge (e \vee \llbracket l = l' \rrbracket) \wedge (\neg e \vee \llbracket l \geq l' \rrbracket)) \\ &= d \vee (\llbracket \sigma = \tau \rrbracket \wedge (\llbracket l = l' \rrbracket \vee (e \wedge \llbracket l > l' \rrbracket))) \\ &= d \oplus \left( \bigwedge_{h=1}^S 1 \oplus \langle \sigma \rangle_h \oplus \langle \tau \rangle_h \wedge \right. \\ &\quad \left. \left( \bigwedge_{k=1}^L 1 \oplus \langle l \rangle_k \oplus \langle l' \rangle_k \vee \left( e \wedge \bigvee_{k=1}^L (1 \oplus \langle l' \rangle_k) \wedge \langle l \rangle_k \wedge \bigwedge_{k'=k+1}^L 1 \oplus \langle l \rangle_{k'} \oplus \langle l' \rangle_{k'} \right) \right) \right) \end{aligned}$$

The  $d \oplus$  can be used since we have set all other bits to 0 if  $d = 1$ . Then we also know that there is exactly one  $h \in [S]$  such that  $\langle \sigma \rangle_h = 1$ . Then the first AND of the right hand side of the outer XOR evaluates to false since all pattern bits are 0 by assumption.

To bring this equation into a nicer form, we treat bit  $e$  as the most significant bit of the number  $l'$ . Likewise we can add this additional bit to the encoding of  $l$  and set it there to 1, as already mentioned above. We refer to the bit by  $\langle l' \rangle_{L+1}$  and likewise for  $l$ . Now we rewrite the OR such that it changes to a OR of  $L + 1$  ANDs of  $L + 2$  XORs of three variables.

▷ **Claim 3.3.1.** There are  $\gamma_{h,k}, \alpha_{h,k}, \beta_{h,k} \in \mathbb{B}$ , and  $i_{h,k} \in [L + 1]$  such that

$$C_\epsilon((\sigma, l), (d, \tau, e, l')) = d \oplus \left( \bigwedge_{h=1}^S 1 \oplus \langle \sigma \rangle_h \oplus \langle \tau \rangle_h \wedge \bigvee_{h=1}^{L+1} \bigwedge_{k=1}^{L+2} \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}} \right)$$

*Proof.* This proof is inspired by the procedure from Chapter 5 in [BK]. It obviously suffices to show that these  $\gamma_{h,k}, \alpha_{h,k}, \beta_{h,k} \in \mathbb{B}$ , and  $i_{h,k} \in [L + 1]$  exist such that

$$\begin{aligned} \left( \bigwedge_{k=1}^L 1 \oplus \langle l \rangle_k \oplus \langle l' \rangle_k \right) \vee \left( e \wedge \bigvee_{h=1}^L (1 \oplus \langle l' \rangle_h) \wedge \langle l \rangle_h \wedge \bigwedge_{k=h+1}^L 1 \oplus \langle l \rangle_k \oplus \langle l' \rangle_k \right) \\ = \bigvee_{h=1}^{L+1} \bigwedge_{k=1}^{L+2} \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}} \end{aligned}$$

We first observe that the left side of the outer OR is a special case of the AND of the right side. Hence, we are going to merge these cases in a later step together. But we first modify the right side by moving the  $e$  into the OR and see it, as described above, as  $L + 1$ th bit of the input. Then we change the right side by making the  $1 \oplus \langle l' \rangle_h$  a part of the AND:

$$\bigvee_{h=1}^L (1 \oplus \langle l' \rangle_h) \wedge \langle l \rangle_h \wedge \bigwedge_{k=h+1}^{L+1} 1 \oplus \langle l \rangle_k \oplus \langle l' \rangle_k = \bigvee_{h=1}^L \langle l \rangle_h \wedge \bigwedge_{k=h}^{L+1} \gamma_{h,k} \oplus \langle l \rangle_k^{\alpha_{h,k}} \oplus \langle l' \rangle_k^{\beta_{h,k}}$$

### 3. UPPER BOUNDS BY FASTER ALGORITHMS

---

Where we set:

$$\begin{aligned} \gamma_{h,k} = \alpha_{h,k} = \beta_{h,k} = 1 & \quad \forall h \in [L], h < k \in [L+1] \\ \gamma_{h,h} = 1 & \quad \forall h \in [L] \\ \alpha_{h,h} = 0 & \quad \forall h \in [L] \\ \beta_{h,h} = 1 & \quad \forall h \in [L] \end{aligned}$$

Now we extend the AND by a  $L+2$ th term such that we only have an OR of an AND:

$$\bigvee_{h=1}^L \langle l \rangle_h \wedge \bigwedge_{k=h}^{L+1} \gamma_{h,k} \oplus \langle l \rangle_k^{\alpha_{h,k}} \oplus \langle l' \rangle_k^{\beta_{h,k}} = \bigvee_{h=1}^L \bigwedge_{k=h}^{L+2} \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}}$$

We define the  $i_{h,k}$  as follows and extend the  $\alpha_{h,k}$ ,  $\beta_{h,k}$ , and  $\gamma_{h,k}$ :

$$\begin{aligned} i_{h,k} = k & \quad \forall h \in [L], h \leq k \in [L+1] \\ i_{h,L+2} = h & \quad \forall h \in [L] \\ \gamma_{h,L+2} = 0 & \quad \forall h \in [L] \\ \alpha_{h,L+2} = 1 & \quad \forall h \in [L] \\ \beta_{h,L+2} = 0 & \quad \forall h \in [L] \end{aligned}$$

By setting  $\gamma_{h,k} = 1$ ,  $\alpha_{h,k} = \beta_{h,k} = 0$  for all  $k < h \in [L]$  we can generalize the inner OR. Furthermore, we add the left part of the original OR as the  $L+1$ th term of the outer OR:

$$\left( \bigwedge_{k=1}^L 1 \oplus \langle l \rangle_k \oplus \langle l' \rangle_k \right) \vee \left( \bigvee_{h=1}^L \bigwedge_{k=1}^{L+2} \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}} \right) = \bigvee_{h=1}^{L+1} \bigwedge_{k=1}^{L+2} \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}}$$

We extend the definition of the values as follows:

$$\begin{aligned} i_{L+1,k} = k & \quad \forall k \in [L] \\ \gamma_{L+1,k} = 1 & \quad \forall k \in [L+2] \\ \alpha_{L+1,k} = 1 & \quad \forall k \in [L] \\ \alpha_{L+1,L+1} = \alpha_{L+1,L+2} = 0 & \\ \beta_{L+1,k} = 1 & \quad \forall k \in [L] \\ \beta_{L+1,L+1} = \beta_{L+1,L+2} = 0 & \end{aligned}$$

These values only depend on  $L$  and can thus be precomputed as soon as  $L$  is known.  $\square$

To simplify the circuit even further we combine the inner AND with the comparison of  $\sigma$  and  $\tau$  and add  $S$  more constraints to the inner AND.

▷ **Claim 3.3.2.** We can extend the  $\gamma_{h,k}, \alpha_{h,k}, \beta_{h,k} \in \mathbb{B}$ , and  $i_{h,k} \in [L+1]$  from the previous claim and there are  $\delta_{h,k} \in \mathbb{B}$  and  $j_{h,k} \in [S]$  such that:

$$C_{\in}((\sigma, l), (d, \tau, e, l')) = d \oplus \bigvee_{h=1}^{L+1} \bigwedge_{k=1}^{L+2+S} \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}} \oplus \langle \sigma \rangle_{j_{h,k}}^{\delta_{h,k}} \oplus \langle \tau \rangle_{j_{h,k}}^{\delta_{h,k}}$$

*Proof.* We set the values to the following:

$$\begin{aligned}
 \gamma_{h,L+2+k} &= 1 & \forall h \in [L+1], k \in [S] \\
 \alpha_{h,L+2+k} &= 0 & \forall h \in [L+1], k \in [S] \\
 \beta_{h,L+2+k} &= 0 & \forall h \in [L+1], k \in [S] \\
 \delta_{h,L+2+k} &= 1 & \forall h \in [L+1], k \in [S] \\
 \delta_{h,k} &= 0 & \forall h \in [L+1], k \in [L+2] \\
 j_{h,L+2+k} &= k & \forall h \in [L+1], k \in [S]
 \end{aligned}$$

The value of  $i_{h,k}$  for all  $h \in [L+1]$  and  $k \in [L+3, L+2+S]$  can be chosen arbitrarily since the term vanishes since  $\alpha_{h,k} = \beta_{h,k} = 0$ . The analog holds for  $j_{h,k}$  with  $k < L+3$ .  $\square$

By this we can later group the parts involving  $(\sigma, l)$  and  $(d, \tau, e, l')$  together to reduce the complexity of the circuit. Since at most one AND can be true at any time, we can replace the OR by an XOR. To simplify notation we define

$$C_\epsilon((\sigma, l), (d, \tau, e, l')) := d \oplus \bigoplus_{h=1}^{L+1} D_h(\sigma, l, \tau, e, l')$$

Where:

$$D_h(\sigma, l, \tau, e, l') := \bigwedge_{k=1}^{L+2+S} \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}} \oplus \langle \sigma \rangle_{j_{h,k}}^{\delta_{h,k}} \oplus \langle \tau \rangle_{j_{h,k}}^{\delta_{h,k}}$$

**MATCHING WORDS** We extend the above circuit to texts and patterns of non trivial size. As for the  $|\circ|$ -matching, we define the following circuit to test whether there is a shift  $i$  such that  $u_{i+1}$  is matched by  $q_1$  and likewise for the following  $f-1$  tuples:

$$C_M(u, q) := \bigvee_{i=0}^{f-1} \bigwedge_{j=1}^f C_\epsilon((\sigma_{i+j}, l_{i+j}), (d_j, \tau_j, e_j, l'_j)) = \bigvee_{i=0}^{f-1} \bigwedge_{j=1}^f d_j \oplus \bigoplus_{h=1}^{L+1} D_h(\sigma_{i+j}, l_{i+j}, \tau_j, e_j, l'_j)$$

Using De Morgans laws we transform the outer OR into a NOT-AND-NOT and replace NOTs by  $1 \oplus$ . The outer  $1 \oplus$  can be omitted since we can swap the result at any point of time during the further calculations. Thus, the circuit changes to

$$C_M(u, q) \equiv \bigwedge_{i=0}^{f-1} 1 \oplus \bigwedge_{j=1}^f d_j \oplus \bigoplus_{h=1}^{L+1} D_h(\sigma_{i+j}, l_{i+j}, \tau_j, e_j, l'_j)$$

### 3.3.3 The Polynomial

**DEFINITION** As in the previous section, we make use of the result due to Razborov and Smolensky as shown in Lemma 2.18 to transform this circuit into a polynomial over  $\mathbb{F}_2$ . We first transform the function  $D_h$  into a probabilistic polynomial  $D_{h,r}$  by replacing the

AND-gate by  $A_t$  as defined in Definition 2.17 with  $t$  to be chosen later.

$$\begin{aligned}
 D_{h,r} &:= \prod_{v=1}^t 1 \oplus \bigoplus_{k=1}^{L+2+S} r_{h,v,k} \left( 1 \oplus \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}} \oplus \langle \sigma \rangle_{j_{h,k}}^{\delta_{h,k}} \oplus \langle \tau \rangle_{j_{h,k}}^{\delta_{h,k}} \right) \\
 &= \prod_{v=1}^t T_{h,v,r}(\sigma, l) \oplus P_{h,v,r}(\tau, e, l') \\
 &= \bigoplus_{z \in \mathbb{B}^t} \prod_{v=1}^t T_{h,v,r}(\sigma, l)^{z_v} P_{h,v,r}(\tau, e, l')^{1-z_v}
 \end{aligned}$$

Where we define:

$$\begin{aligned}
 T_{h,v,r}(\sigma, l) &:= \bigoplus_{k=1}^{L+2+S} r_{h,v,k} \left( 1 \oplus \gamma_{h,k} \oplus \langle l \rangle_{i_{h,k}}^{\alpha_{h,k}} \oplus \langle \sigma \rangle_{j_{h,k}}^{\delta_{h,k}} \right) \oplus 1 \\
 P_{h,v,r}(\tau, e, l') &:= \bigoplus_{k=1}^{L+2+S} r_{h,v,k} \left( \langle l' \rangle_{i_{h,k}}^{\beta_{h,k}} \oplus \langle \tau \rangle_{j_{h,k}}^{\delta_{h,k}} \right)
 \end{aligned}$$

We remark that  $e$  is treated as part of  $\langle l' \rangle$  in the formula. For fixed  $r$ ,  $(\sigma, l)$ , and  $(\tau, e, l')$  we can see the  $T$ 's and  $P$ 's as variables indexed by  $h$  and  $v$ . This is because the definition of the  $\gamma$ ,  $\alpha$ ,  $\beta$ , and  $\delta$  only depends on the number of bits (i.e.  $L$  and  $S$ ) and can thus be precomputed.

By the change from  $D_h$  to  $D_{h,r}$  the polynomial has the following form:

$$\bigwedge_{i=0}^{f-1} 1 \oplus \bigwedge_{j=1}^f d_j \oplus \bigoplus_{h=1}^{L+1} \bigoplus_{z \in \mathbb{B}^t} X_{i,j,h,z}$$

Where we defined  $X_{i,j,h,z} := \prod_{v=1}^t T_{h,v,r}(\sigma_{i+j}, l_{i+j})^{z_v} P_{h,v,r}(\tau_j, e_j, l'_j)^{1-z_v}$ . For now, we see these values as monomials in the  $T$ 's and  $P$ 's. This is reasonable since the  $X_{i,j,h,z}$  are products of  $T$ 's and  $P$ 's.

Finally, we replace the two outer ANDs by probabilistic polynomials  $A_3$  and  $A_{t'}$ , where  $t'$  is chosen later.

$$\begin{aligned}
 &\prod_{i'=1}^3 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \left( \prod_{j'=1}^{t'} 1 \oplus \bigoplus_{j=1}^f r_{i',i,j',j} \left( 1 \oplus d_j \oplus \bigoplus_{h=1}^{L+1} \bigoplus_{z \in \mathbb{B}^t} X_{i,j,h,z} \right) \right) \\
 &= \prod_{i'=1}^3 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \left( \underbrace{\prod_{j'=1}^{t'} 1 \oplus \bigoplus_{j=1}^f r_{i',i,j',j}}_{r_{i',i,j':j}} \bigoplus_{j=1}^f r_{i',i,j',j} \left( d_j \oplus \bigoplus_{h=1}^{L+1} \bigoplus_{z \in \mathbb{B}^t} X_{i,j,h,z} \right) \right) \\
 &= \prod_{i'=1}^3 1 \oplus \bigoplus_{i=0}^{f-1} r_{i',i} \left( \prod_{j'=1}^{t'} r_{i',i,j'} \bigoplus_{j=1}^f r_{i',i,j',j} \left( d_j \oplus \bigoplus_{h=1}^{L+1} \bigoplus_{z \in \mathbb{B}^t} X_{i,j,h,z} \right) \right) \\
 &=: Q_r(u, q)
 \end{aligned}$$

**ERROR BOUND** We show that  $Q_r$  agrees with  $C_M$  with probability at least  $5/8$  and thus has error probability at most  $3/8$ . Even though this error is larger than  $1/3$  it does not affect the success probability of our algorithm since the standard boosting technique still applies. We choose  $t := \log(24(L+1)f^2 \log(24f))$ , and  $t' := \log(24f)$ . From Lemma 2.18 we know that the outer polynomial errs with probability at most  $1/2^3$ . The middle one with probability at most  $1/2^{t'}$ , but it is used  $3f$  times. The inner one is used  $3ft'f(L+1)$  times with error of at most  $1/2^t$ . Using the union bound, we get a total error probability of at most

$$\frac{1}{8} + \frac{3f}{2^{t'}} + \frac{3ft'f(L+1)}{2^t} = \frac{1}{8} + \frac{3f}{24f} + \frac{3f \log(24f)f(L+1)}{24(L+1)f^2 \log(24f)} \leq 3/8$$

### 3.3.4 Counting the Monomials

To count the number of monomials of  $Q_r$  we convert it to a polynomial that is a sum (XOR) of products. Therefore, we have to change the order of the quantifiers.

**FIRST FLIP** Change the order of the inner sums with the inner product: Similar as for  $|\circ|$  we count the number of subsets from  $\{1, \eta_1, \dots, \eta_f\}$  with  $\eta_j = d_j \oplus \bigoplus_{h=1}^{L+1} \bigoplus_{z \in \mathbb{B}^t} X_{i,j,h,z}$ . But we only consider subsets of size at most  $t'$ . This gives us a total number of  $\sum_{w=1}^{t'} \binom{f+1}{w}$  monomials with  $\eta_j$  as variables and degree  $t'$ . Since each  $\eta_j$  is a sum of  $(L+1)2^t + 1$  many monomials, the final number of monomials in this inner product is

$$\sum_{w=1}^{t'} \binom{f+1}{w} ((L+1)2^t + 1)^w.$$

**SECOND FLIP** Applying the distributivity law for the outer product we get that  $Q_r$  has at most  $(f \sum_{w=1}^{t'} \binom{f+1}{w} ((L+1)2^t + 1)^w + 1)^3$  monomials of degree at most  $3t'$  in the  $X$ 's and thus of  $3t't$  in the  $T$ 's and  $P$ 's. By bounding the sum the number of monomials is at most  $M := (2f \binom{f+1}{t'} ((L+1)2^t + 1)^{t'})^3$ .

**BOUNDING THE NUMBER OF MONOMIALS** Recall, we defined  $t' = \log(24f)$  and  $t \leq \log(24(L+1)f^3)$ . We prove  $M \leq (n/2f)^{0.1}$  which is equivalent to show  $\sqrt[3]{M} \leq (n/2f)^{1/30}$ . This holds if  $\Gamma := \log(\sqrt[3]{M}) + 1/30 \cdot \log(2f) \leq 1/30 \cdot \log n$ :

$$\begin{aligned} \Gamma &\leq 1 + \log f + \log \binom{f+1}{t'} + t' \log((L+1)2^t + 1) + 1/30 \cdot \log(2f) \\ &\leq 2 \log f + t' \log(f+1) + t' \log((L+1)24(L+1)f^3 + 1) \\ &\leq 2 \log f + \log(24f) \log(f+1) + \log(24f) \log(24(L+1)^2 f^3 + 1) \\ &\leq \log(24f) \left( 2 + \log(f+1) + \log(24(L+1)^2 f^3 + 1) \right) \\ &\leq 2 \log^2(24(L+1)^2 f^3 + 1) \end{aligned}$$

When setting  $f$  as follows we get  $\Gamma \leq 1/30 \cdot \log n$  and thus the claimed bound holds. Additionally,  $f \in 2^{\Theta(\sqrt{\log n})} / L^{2/3}$  and hence  $f \in 2^{\Theta(\sqrt{\log n})}$  since  $L \in \mathcal{O}(\log n)$ .

$$f := \sqrt[3]{\frac{2\sqrt{1/60 \cdot \log n} - 1}{24(L+1)^2}}$$

*Remark.* We observe that the size of the alphabet  $\mathfrak{s}$  does not occur in this final computation. This is due to the fact, that we used the  $T_{h,q,r}$  and  $P_{h,q,r}$  to preprocess our input data. Thus,  $\mathfrak{s}$  only affects the dimensions for our polynomial evaluation. For this still to work we need  $d_1 + d_2 \in \mathcal{O}(n^{0.9})$  as one can see in Lemma 2.15. But since the dimensions are poly-logarithmic in  $n$ ,  $m$ , and  $\mathfrak{s}$  the method works for any  $\mathfrak{s} \in \mathcal{O}(n)$ . ■

### 3.4 Derandomizing the Algorithm for $|\circ|$

**DERANDOMIZATIONS** In the last sections we have seen randomized algorithms to solve the pattern matching problem for patterns of type  $|\circ|$  and  $|\circ+$ . These algorithms and constructions are, as already mentioned, inspired by the method of Williams, who showed the first super-logarithmic improvement for ALL-PAIRS SHORTEST PAIRS in [Wil14b]. Both of our algorithms are randomized and therefore only correct with high probability. Even though Williams showed a derandomization in the original paper, this deterministic algorithm has a worse running time than the randomized approach. It took a few year until Chan and Williams [CW16] showed that the problem can be solved *deterministically* and hence without an error in the asymptotically same time as for the randomized case. They made use of so called *small-bias sets* and *modulus-amplifying polynomials*. We use these techniques to present a deterministic version of the algorithm for  $|\circ|$ -matching. This derandomization is based on the fact that one can simulate the SUM of ORs by a small polynomial (for a sufficient definition of small). We use this polynomial to simulate that part of our circuit that was converted into a probabilistic polynomial by the method of Razborov and Smolensky.

**Definition 3.9** (SUM-OR (cf. Section 2 in [CW16])). *Let  $d_1$  and  $d_2$  be positive integers. Define the function  $\text{SUM-OR}_{d_1,d_2} : \mathbb{B}^{d_1 \cdot d_2} \rightarrow \{0, \dots, d_1\}$  as follows:*

$$\text{SUM-OR}_{d_1,d_2}(x_{1,1}, \dots, x_{1,d_2}, \dots, x_{d_1,1}, \dots, x_{d_1,d_2}) := \sum_{i=1}^{d_1} \bigvee_{j=1}^{d_2} x_{i,j}$$

Chan and Williams proved the following lemma combining small-bias sets ([NN93; Alo+92]) and modulus-amplifying polynomials ([BT94]):

**Lemma 3.10** (Theorem 2.1 in [CW16]). *Let  $d_1$  and  $d_2$  be positive integers such that  $10 \log(d_1 \cdot d_2) < d_2$ . There are integers  $l = 5 \log(d_1 \cdot d_2)$ ,  $N \leq \text{poly}(d_1, d_2)$ , and a polynomial  $P_{d_1,d_2}$  in  $d_1 \cdot d_2$  variables over  $\mathbb{Z}$  with  $m \leq \binom{d_2}{2l} \cdot d_1^3 d_2^2$  monomials, such that for all  $\vec{x} \in \mathbb{B}^{d_1 \cdot d_2}$ ,  $\text{SUM-OR}_{d_1,d_2}(\vec{x})$  equals the nearest integer to  $(P_{d_1,d_2}(\vec{x}) \bmod 2^l) / N$ . Furthermore,  $P_{d_1,d_2}$  can be constructed in  $\text{poly}(d_1) \cdot \binom{d_2+1}{l}^2$  time.*

Even though the degree of the polynomial is not mentioned in the stated lemma, the following fact is contained in its proof. The degree is solely due to the degree of the modulus-amplifying polynomial.

**Lemma 3.11.**  *$P_{d_1,d_2}$  has degree  $2l - 1$ .*



THE DERANDOMIZATION As in Section 3.2, let  $u = u_1 \dots u_{2f}$  be a text of length  $2f$  and  $q = q_1 \dots q_f$  be a pattern of size  $f$  and type  $\circ|$ .  $f$  is determined later. Recall, that the circuit to check whether a substring of length  $f$  of  $u$  is matched by  $q$  was defined as follows:

$$\begin{aligned} C_M(\chi(u), \chi(q)) &:= \bigvee_{i=0}^{f-1} \bigwedge_{j=1}^f \bigvee_{k=1}^{\mathfrak{s}} \chi(u_{i+j})[k] \chi(q_j)[k] \\ &= \neg \bigwedge_{i=0}^{f-1} \bigvee_{j=1}^f \neg \bigvee_{k=1}^{\mathfrak{s}} \chi(u_{i+j})[k] \chi(q_j)[k] \end{aligned}$$

By the definition of  $\chi$  there is at most one  $k$  such that the inner product is true. Hence, we replace the inner OR by XOR as we did for the randomized case. Furthermore, we replace the NOT by  $1 \oplus$ . Additionally, we can assume a  $\mathfrak{s} + 1$ th bit for the characteristic vector that is always set to true to remove the inner  $1 \oplus$ . By this the circuit changes to:

$$C_M(\chi(u), \chi(q)) = 1 \oplus \bigwedge_{i=0}^{f-1} \bigvee_{j=1}^f \bigoplus_{k=1}^{\mathfrak{s}+1} \chi(u_{i+j})[k] \chi(q_j)[k]$$

Before applying Lemma 3.10 we first observe the following:

$$\begin{aligned} C_M(\chi(u), \chi(q)) = 1 &\iff C_M(\chi(u), \chi(q)) \oplus 1 = 0 \\ &\iff \bigwedge_{i=0}^{f-1} \bigvee_{j=1}^f \bigoplus_{k=1}^{\mathfrak{s}+1} \chi(u_{i+j})[k] \chi(q_j)[k] = 0 \\ &\iff \sum_{i=0}^{f-1} \bigvee_{j=1}^f \bigoplus_{k=1}^{\mathfrak{s}+1} \chi(u_{i+j})[k] \chi(q_j)[k] < f \end{aligned}$$

Where the outer sum is over the integers and the inner sum, the  $\bigoplus$ , is over the booleans (i.e. integer addition modulo 2).

Now rewrite the SUM of ORs by a polynomial  $P_{f,f}$  that is guaranteed by Lemma 3.10. By your choice of  $d_1 = d_2 = f$ , we get  $l = 5 \log(f^2) = 10 \log f$  and  $m \leq \binom{f}{2l} f^3 f^2 = \binom{f}{20 \log f} f^5$  monomials. Furthermore, the polynomial can be constructed in time  $\text{poly}(f) \binom{f+1}{10 \log f}^2$  and has degree  $D = 2l - 1 = 20 \log f - 1$  when considering the  $\bigoplus_{k=1}^{\mathfrak{s}+1} \chi(u_{i+j})[k] \chi(q_j)[k]$  as variables  $X_{i,j}$ . But since these  $X_{i,j}$  are no variables, the number of monomials has to be multiplied by  $(\mathfrak{s} + 1)^D = (\mathfrak{s} + 1)^{20 \log f - 1}$  for expanding the XOR. Since these expansions can be written down in a straightforward way for each  $i, j$ , the running time increases by this factor only. The degree of the polynomial stays the same after this expansion.

To make use of the fast matrix multiplication, we prove the following inequality:

$$\begin{aligned} M &:= m \cdot (\mathfrak{s} + 1)^{20 \log f - 1} \leq (n/2f)^{0.1} \\ \iff \log M + 0.1 \log 2f &= \log m + (20 \log f - 1) \log(\mathfrak{s} + 1) + 0.1 \log 2f \leq 0.1 \log n \end{aligned}$$

### 3. UPPER BOUNDS BY FASTER ALGORITHMS

---

Now it remains to prove this bound:

$$\begin{aligned}\log M + 0.1 \log 2f &= \log m + (20 \log f - 1) \log(s + 1) + 0.1 \log 2f \\ &\leq \log \binom{f}{20 \log f} + 5 \log f + 20 \log f \log(s + 1) + 0.1 \log 2f \\ &\leq (20 \log f) \log f + 26 \log f \log(s + 1) \\ &\leq 46 \log(s + 1) \log^2 f\end{aligned}$$

When choosing  $f$  as follows, the claimed bound holds and  $f \in 2^{\Theta(\sqrt{\log n})}$  as for the randomized case if the size of the alphabet is constant:

$$f := 2^{\sqrt{\log n / 460 \log(s+1)}}$$

■

---

## Conditional Lower Bounds for Pattern Matching

In [BI16; BGL17] conditional lower bounds based on SETH are shown ruling out sub-quadratic time algorithms for pattern matching (and membership) with certain *homogeneous* patterns of *bounded* depth. Since the two patterns from the previous chapter are classified to be hard (i.e. no sub-quadratic time algorithm), we cannot hope for much faster algorithms than the ones given. Furthermore, this implies that there are no sub-quadratic time algorithms for pattern matching and the membership problem in general.

Using reductions from FORMULA-PAIR and thus from FORMULA-SAT an almost tight conditional lower bound for pattern matching in general was shown. This bound rules out an improvement of more than 7 log-factors compared to the quadratic time approach (see [AB18; Abb+16] for more information).

But in contrast to the lower bounds shown in [BI16; BGL17], this (almost) tight bound in [AB18] is for *non-homogeneous* patterns of *unbounded* depth. Since this bound does not generalize to patterns of *bounded* depth and *homogeneous* type, the question rises whether we can adopt the result from [AB18] to *homogeneous* patterns of *bounded* depth.

In this chapter we show almost tight conditional lower bounds for the pattern matching problem with homogeneous patterns of depth two and three that do not have strongly sub-quadratic time algorithms, except for  $|\circ|$  and  $|\circ+$ . By reducing from FORMULA-PAIR we show that one cannot shave off an arbitrary number of log-factors from the quadratic running time for these patterns. Hence, the application of methods yielding super-polylogarithmic improvements is not possible (e.g. the polynomial method).

We base all conditional lower bounds on the FORMULA-PAIR HYPOTHESIS (FPH) (Hypothesis 3). This hypothesis is the implication of the FORMULA-SAT HYPOTHESIS for the FORMULA-PAIR problem which is used for the results in [AB18; Abb+16]. Thus, assuming FPH the currently best algorithm for pattern matching running in time  $\mathcal{O}(nm/\log^{3/2} n \cdot \text{polyloglog } n)$  [BT09] is optimal up to a constant number of log-factors for these pattern types.

**STRUCTURE** We first state our results in a formal way in Section 4.1. There we also explain the main ideas of the reductions. Then we give the blue print of all further reductions in Section 4.2 for patterns of type  $\circ+\circ$ . In the following sections we show variations for each pattern type. As a last step we show the proof of one theorem in Section 4.7 which is used in all reductions. In Appendix A we show that one can modify the result in [AB18] such that it applies to *homogenous* patterns of *unbounded* depth. Even though a weaker version

of the bound can be derived from the lower bound for  $\circ\star$ , it shows that for patterns of unbounded depth even tighter bounds can be proven.

## 4.1 Results and General Idea

**RESULTS** By the observations from [BI16; BGL17] which we mentioned in the introduction it suffices to show the following theorem. The hardness of the other pattern types is obtained from these types by Lemma 1.1 and Lemma 1.2. We prove the theorem in the remaining sections for each pattern type.

**Theorem 4.1.** *Pattern matching with the following homogenous types cannot be solved in the stated running time even for constant sized alphabets, unless FPH as in Hypothesis 3 is false:*

$$\begin{array}{c|c|c|c|c} \circ\star & \circ+\circ & \circ|\circ & \circ+| & \circ|+ \\ \hline \mathcal{O}\left(\frac{nm}{\log^{76} n}\right) & \mathcal{O}\left(\frac{nm}{\log^{72} n}\right) & \mathcal{O}\left(\frac{nm}{\log^{81} n}\right) & \mathcal{O}\left(\frac{nm}{\log^{27} n}\right) & \mathcal{O}\left(\frac{nm}{\log^{72} n}\right) \end{array}$$

For homogenous patterns of unbounded depth (of type  $\circ\star\circ\star\dots$ ) there is no  $\mathcal{O}(nm/\log^7 n)$  time algorithm.

For the lower bounds it suffices to prove the following reductions from FORMULA-PAIR to pattern matching with the given types and size bounds:

**Lemma 4.2.** *Given a FORMULA-PAIR instance with a formula  $F$  of size  $s$  and sets  $A$  and  $B$  with  $n$  half-assignments each. We can reduce this to an instance of pattern matching with the stated type, the stated bounds for the text length  $N$  and pattern size  $M$ , constant sized alphabet, and in time linear in the size of the output:*

Type	$\circ\star$	$\circ+\circ$	$\circ \circ$	$\circ+ $	$\circ +$
$N$	$ns^{6\ln 5+2} \log s$	$ns^{6\ln 5+2} \log s$	$ns^{6\ln 5+2}$	$ns^{6\ln 2+2} \log s$	$ns^{6\ln 5+2} \log s$
$M$	$ns^{6\ln 6+2} \log s$	$ns^{6\ln 5+2} \log s$	$ns^{6\ln 8+2}$	$ns^2 \log s$	$ns^{6\ln 5+2} \log s$

Note: The size bound indicates the magnitude, not the actual value ( $\mathcal{O}$ -notation).

*Proof of Theorem 4.1.* The theorem directly follows from the lemma since the claimed running time would directly disprove FPH which is conjectured to be true. For the case of unbounded depth, see Appendix A.  $\square$

**GENERAL IDEA** When analyzing the proof of Theorem E.1 in the full version of [AB18] one sees that the INPUT and AND gates do not increase the depth of the pattern. The increase of the pattern depth (and the non-homogeneity) is only caused by the OR gates (and the outer OR). While it is relatively easy to transform the pattern into a homogenous one, it is hard to “flatten” it such that the depth becomes constant (and especially  $\leq 3$ ). Our main idea is that we do not only design one text  $t_g(a)$  and one pattern  $p_g(b)$  for pairs  $a, b$  of half-assignment and gate  $g$  but also a universal text  $u_g$  and a universal pattern  $q_g$ . These patterns are designed such that the universal text  $u_g$  can be matched by  $p_g(b)$  independently from the half-assignment  $b$ . The analog holds for the text  $t_g(a)$  and the universal pattern  $q_g$ . But additionally we also require that  $u_g$  can be matched by  $q_g$ .

All of our reductions use a small ( $\leq 6$ ) constant sized alphabet but not a binary one! By using more symbols we can simplify the reduction since several symbols are used as separators (e.g. 2 and  $\gamma$ ) while the symbols 0, 1 encode the actual information. It is unclear and remains an open questions whether the reductions can be done with a smaller alphabet.

We always start with some De Morgan formula and two half-assignments for two distinct sets of variables, we also call them assignments in the following. We first define a separator gadget over the alphabet  $\{0, 1, 2\}$  that is essentially the binary encoding of the number of the gate we consider. This separator and the encoding of the INPUT and AND gate is the same for all reductions. In the case of the AND gate we have that for both children of the current gate the pattern has to match the corresponding text. This is done by using the separator gadget enforcing correct alignment. This works because the binary representation is unique and thus the gadget does not appear somewhere else in the text. For the OR gate we need some more involved techniques that are (slightly) different for every pattern type. In general the idea is that we allow the pattern to be aligned to the text in (at most) two different ways. For the first case the text and the pattern of the left child of the gate should match and for the second case vice-versa. Using this approach it remains to design the left most and right most part of the pattern and text such that the pattern can match the whole text. This is the more involved part since the additional parts have to be matched, too. For this we use the approach that we define some pattern that can match some text  $T$  and  $TT$ . This is different for all pattern types and heavily exploits the available operators of the pattern. As a last step we have to design the outer OR such that we can pick one assignment from the first set and one from the other set. But still the whole pattern has to match some text. We do this by defining a framework that requires five additional helper gadgets with certain properties. Since these helper gadgets are in general quite small, one can easily show these requirements.

The reductions we give in the following sections depend not only on the size of the formula but also exponentially on the depth. Since the depth and the size can be of same magnitude, we need some techniques to reduce the depth. For this we use the already known depth-reduction technique as shown in [BB94]. These techniques generate an equivalent formula whose depth is logarithmic in its size. By this the exponential dependence on the depth becomes a polynomial dependence on the size. Hence, we can reduce the proof of Lemma 4.2 to the following lemma:

**Lemma 4.3.** *Given a FORMULA-PAIR instance with a formula  $F$  of size  $s$ , depth  $d$ , and sets  $A$  and  $B$  with  $n$  and  $m \leq n$  assignments, respectively. We can reduce this to pattern matching of the stated type, with the stated bounds for the text length  $N$  and pattern size  $M$ , constant sized alphabet, and in time linear in the size of the output:*

Type	$\circ\star$	$\circ+\circ$	$\circ \circ$	$\circ+ $	$\circ +$
$N$	$\mathcal{O}(n5^d s \log s)$	$\mathcal{O}(n5^d s \log s)$	$\mathcal{O}(n5^d s \log s)$	$\mathcal{O}(n2^d s \log s)$	$\mathcal{O}(n5^d s \log s)$
$M$	$\mathcal{O}(m6^d s \log s)$	$\mathcal{O}(m5^d s \log s)$	$\mathcal{O}(m8^d s \log s)$	$\mathcal{O}(ms \log s)$	$\mathcal{O}(m5^d s \log s)$
Proof	Section 4.4	Section 4.2	Section 4.3	Section 4.5	Section 4.6

*Proof of Lemma 4.2.* We show the result only for patterns of type  $\circ\star$ , the proof for the other types is analogous. First apply the depth-reduction technique of Bonet and Buss

with  $k = 2$  as stated in Lemma 2.19 (cf. [BB94]) to  $F$ . This yields an equivalent formula  $F'$  of depth  $d' \leq 6 \ln 2 \log s$  and size  $s' \leq s^2$ . Now apply the reduction from Lemma 4.3. Hence, the text length is  $\mathcal{O}(n5^{6 \ln 2 \log s} s^2 \log s)$  and the pattern size  $\mathcal{O}(n6^{6 \ln 2 \log s} s^2 \log s)$ . Since  $F'$  is equivalent to  $F$  this proves the claimed bound.  $\square$

NOTATION IN THE FOLLOWING SECTIONS If not stated otherwise, we assume the following:

- $a$  and  $b$  are two assignments for the two sets of variables. We omit them in the first section for the sake of easier notation.
- $r$  is the root of  $F$ ,  $s$  the size, and  $d$  the depth.
- For any gate  $g$  of  $F$ , let  $F_g$  be the formula restricted to the sub-formula rooted at  $g$ .
- Then considering the text  $t_r(a)$  we omit the index  $r$  to simplify notation, the same holds for patterns  $p_r(b)$ .
- When we consider a gate  $g$ , we let  $g_1$  and  $g_2$  be the left and right child, respectively. We index texts and patterns corresponding to a specific child by the index of the child.

## 4.2 Patterns of Type $\circ + \circ$

### 4.2.1 Encoding the Formula

We first observe that a formula of size  $s$  (that is the number of leaves) has exactly  $s - 1$  inner gates and thus  $2s - 1$  gates in total. We associate a unique integer in  $[2s]$  with every gate of the formula and call it the ID of a gate in the following.<sup>1</sup> To simplify notation we identify the gate with its ID and write  $\langle g \rangle$  for the binary encoding of the ID of gate  $g$ .

To encode the ID of any gate, i.e. an integer in  $[2s]$ , we need  $\lfloor \log(2s) \rfloor + 1 = \lfloor \log s \rfloor + 2$  bits. Hence, we can assume  $\langle g \rangle$  always outputs a sequence of  $\lfloor \log s \rfloor + 2 = \Theta(\log s)$  bits padded with zeros if necessary. Using this we define the separator gadget  $G$  when considering a fixed gate  $g$  where the symbol “2” is used to separate the encoding from other gadgets:

$$G := 2\langle g \rangle 2$$

**INPUT Gate** Let the gate  $g$  read variable  $x_i$  (i.e. it comes from the first set). Then the gate is true if and only if  $a_i = 1$ . Thus, we define

$$t_g := 0a_i 1 \quad p_g := 0^+ 1 1^+$$

If the gate  $g$  reads variable  $y_i$ , we define

$$t_g := 0 1 1 \quad p_g := 0^+ b_i 1^+$$

We define the universal text and pattern as follows:

$$u_g := 0011 \quad q_g := 0^+ 1^+$$

---

<sup>1</sup>We use  $[2s]$  instead of  $[2s - 1]$  just to keep notation simple.

**AND Gates** We define the texts and the patterns as mentioned in the introduction to this chapter:

$$\begin{aligned} t_g &:= t_1 G t_2 & u_g &:= u_1 G u_2 \\ p_g &:= p_1 G p_2 & q_g &:= q_1 G q_2 \end{aligned}$$

**OR Gates** The texts and the patterns for gate  $g$  are defined as follows:

$$\begin{aligned} t_g &:= (u_1 G G u_2) G (u_1 G G u_2) G (t_1 G G t_2) G (u_1 G G u_2) G (u_1 G G u_2) \\ u_g &:= (u_1 G G u_2) G (u_1 G G u_2) G (u_1 G G u_2) G (u_1 G G u_2) G (u_1 G G u_2) \\ q_g &:= (u_1 G G u_2) G (u_1 G G u_2) G (q_1 G G q_2) G (u_1 G G u_2) G (u_1 G G u_2) \\ p_g &:= (u_1 G G u_2 G)^+ (q_1 G G p_2) G (p_1 G G q_2) (G u_1 G G u_2)^+ \end{aligned}$$

One can easily see, the definitions of  $u_g$ ,  $t_g$ , and  $q_g$  only differ at two positions from each other: That is where we replace  $t_1$  and  $t_2$  by  $u_1$  and  $u_2$  for  $u_g$  and by  $q_1$  and  $q_2$  for  $q_g$ , respectively. The parenthesis in this and all following encodings are not part of the text or pattern but they are used for grouping reasons and to make reading easier.

#### CORRECTNESS AND SIZE BOUND

**Lemma 4.4** (Correctness of the Construction). *For all assignments  $a, b$  and gates  $g$ :*

- $F_g(a, b) = \text{true} \iff t_g(a) \in \mathcal{L}(p_g(b))$
- $t_g(a) \in \mathcal{L}(q_g)$
- $u_g \in \mathcal{L}(q_g) \cap \mathcal{L}(p_g(b))$

*Proof.* The proofs of the second and third claim follow inductively from the encoding of the gates because of our encoding of the INPUT gate. For the first claim we do a structural induction on the output gate of the formula.

**INPUT Gate “ $\Rightarrow$ ”** One can easily see if the gate (i.e. the variable) is satisfied, then the text matches the pattern. Both are equal up to occurrences of repetitions in the pattern.

**INPUT Gate “ $\Leftarrow$ ”** If the gate is not satisfied and a variable  $x_i$  is read, the text changes to 001 but the pattern  $0^+11^+$  has to match at least two 1s. Hence, they cannot match. For the case when a variable  $y_i$  is read, the pattern changes to  $0^+01^+$  but the text stays 011 and again no match is possible.

**AND Gate “ $\Rightarrow$ ”**  $F_g(a, b) = \text{true} = F_{g_1}(a, b) \wedge F_{g_2}(a, b) \Rightarrow F_{g_1}(a, b) = \text{true} \wedge F_{g_2}(a, b) = \text{true}$ . This implies that the pattern matches the text.

**AND Gate “ $\Leftarrow$ ”** If  $t_g \in \mathcal{L}(p_g)$ , then the  $G$  in the middle of the pattern has to match the corresponding part in the text because the gadget  $G$  does not appear somewhere else in the text. Furthermore, we know that the complete text is matched. We get  $t_1 \in \mathcal{L}(p_1)$  and  $t_2 \in \mathcal{L}(p_2)$ . By assumption this shows  $F_{g_1}(a, b) = \text{true}$  and  $F_{g_2}(a, b) = \text{true}$  and hence  $F_g(a, b) = F_{g_1}(a, b) \wedge F_{g_2}(a, b) = \text{true}$ .

**OR Gate “ $\Rightarrow$ ”**  $F_g(a, b) = F_{g_1}(a, b) \vee F_{g_2}(a, b) = \text{true}$ . Assume w.l.o.g. that  $F_{g_1}(a, b) = \text{true}$ , the other case is symmetric. By assumption  $t_1 \in \mathcal{L}(p_1)$ . The first part of the pattern, i.e.  $(u_1GGu_2G)^+$ , is only repeated once. Because of this, we can transform  $q_1GGp_2$  into the second  $u_1GGu_2$  by our third claim of the lemma. Now the  $p_1GGq_2$  matches  $t_1GGt_2$  by our assumption and the second claim. Finally, we match  $Gu_1GGu_2Gu_1GGu_2$  by two repetitions of  $(Gu_1GGu_2)^+$ .

**OR Gate “ $\Leftarrow$ ”** Assume  $t_g \in \mathcal{L}(p_g)$ . This implies that the  $G$ s in the text have to be matched. Furthermore, since the complete text is matched, the two repetitions can only be repeated once or twice each. Otherwise we had to match  $t_1$  to  $u_1$  and  $t_2$  to  $u_2$ . This is impossible since  $u_1$  and  $u_2$  are strings and hence do not allow to match any other string.

Assume the first repetition is only repeated once. Then, the following  $q_1GGp_2$  has to be matched to the second  $u_1GGu_2$  in the text. This follows from the fact that the  $G$ s in the pattern have to match the  $G$ s in the text since they do not occur somewhere else in the text. But then it must be the case that  $p_1$  is transformed into  $t_1$  showing that the gate has to be satisfied by the inductive hypothesis.

Precisely this argument holds for the second repetition, too. Thus, we can assume that the first and second repetition are repeated twice each. Then  $t_1GGt_2$  must be matched to  $(q_1GGp_2)G(p_1GGq_2)$ , i.e. the remaining part of the pattern. But since the text contains two  $G$ s and the pattern has to match five  $G$ s, this leads to a contradiction and hence this case cannot occur.  $\square$

**LENGTH OF THE TEXT AND THE PATTERN** By a close observation we can see that the construction of the texts and the patterns can be done straight forward. For this we only have to recurse on the formula. All texts and patterns can be constructed by the same recursive call since we can write down the texts and patterns explicitly when we have the texts and patterns for the two sub-formulas given. By this the running time of the encoding takes time linear in the size of the output by using space that is also linear in this. Thus, it remains to analyse the length of the texts and the size of the patterns:

**Lemma 4.5.**  $|u_r|, |t_r|, |p_r|, |q_r| \in \mathcal{O}(5^d s \log s)$ .

*Proof.*  $p_g$  is obviously smaller than  $u_g$ . Hence, it suffices to analyse the length of  $u_g$  since the sizes of  $u_g$ ,  $t_g$ , and  $q_g$  are asymptotically equal:

$$|u_g| \leq 5|u_1| + 5|u_2| + \mathcal{O}(\log s)$$

Inductively over the  $d(F_g)$  levels of  $F_g$  this yields  $|u_g| \leq \mathcal{O}(5^{d(F_g)} s \log s)$ . The factor of  $s \log s$  is because there are  $\mathcal{O}(s)$  inner gates and each introduces  $\mathcal{O}(\log s)$  symbols.  $\square$

#### 4.2.2 Final Reduction

Instead of giving the direct construction and proof for this and every other pattern type, we show a generalized version that makes use of helper gadgets. The final text and pattern depend on the texts and patterns we defined above and on new helper gadgets for which



we require some properties. As soon as we define these gadgets for this and the other pattern types, we directly get the final text and pattern by the following theorem. The two texts  $S$  and  $S'$  and the three patterns  $\eta$ ,  $U_2$ , and  $U_3$  have to satisfy the following properties:

- $S$  is a proper substring of  $S'$  and  $S, S' \in \mathcal{L}(\eta)$ .
- $S$ ,  $S'$ , and  $\eta$  only consist of fresh symbols.
- $S'$  has a non-empty substring that cannot be matched by  $U_2$ .
- $|S'|, |U_2|, |U_3|, |\eta| \in \mathcal{O}(|u_r|)$ .
- $\{Su_r S, Su_r Su_r S\} \in \mathcal{L}(U_2)$
- $\{S, S'\} \circ \{u_r S, u_r Su_r S, u_r Su_r Su_r S\} \in \mathcal{L}(U_3)$
- For all  $t$ :  $Su_r S' t \notin \mathcal{L}(U_3)$
- Either (1)  $\mathcal{L}(U_3) \subseteq \mathcal{L}(\eta(u_r S)^+)$  or (2) for any word  $t \in \mathcal{L}(U_3)$  there are at most four disjoint intervals that consist only of symbols that occur in  $S$  or  $S'$ . The last interval ends at the end of  $t$ .

Analogously for three intervals for  $U_2$ .

**Theorem 4.6.** *Given the following:*

- A monotone De Morgan formula  $F$  of size  $s$ .
- Sets  $A$  and  $B$  of  $n$  and  $m \leq n$  half-assignments to  $s/2$  variables, respectively.
- A text gadget  $t(a)$  and a pattern gadget  $p(b)$  for  $a \in A$  and  $b \in B$ , such that  $t(a) \in \mathcal{L}(p(b))$  if and only if  $F(a, b) = \text{true}$ .
- A universal text gadget  $u_r$  and a universal pattern gadget  $q_r$  such that  $u_r, t(a) \in \mathcal{L}(q_r)$  for all  $a \in A$  and  $u_r \in \mathcal{L}(p(b))$  for all  $b \in B$ .
- Helper gadgets  $S, S', \eta, U_2$ , and  $U_3$  as described above.

Then we can construct a text  $t$  and a pattern  $p$  such that  $t \in \mathcal{M}(p)$  if and only if there is an  $a \in A$  and a  $b \in B$  such that  $F(a, b) = \text{true}$ . Furthermore,  $|t| = \mathcal{O}(n(|u_r| + |t(0)|))$ ,  $|p| = \mathcal{O}(m(|u_r| + |p(0)| + |q_r|))$  and  $t$  and  $p$  are concatenations of gadgets.

We prove the theorem in Section 4.7. One can easily see, that the first four assumptions are satisfied by Lemma 4.4. Thus it only remains to define the helper gadgets with the claimed properties:

$$\begin{aligned} S &:= \gamma \\ S' &:= \gamma\gamma \\ \eta &:= \gamma^+ \\ U_2 &:= \gamma(u_r\gamma)^+ \\ U_3 &:= \gamma^+(u_r\gamma)^+ \end{aligned}$$

The correctness of these gadgets follows directly from the fact that  $\gamma$  is a fresh symbol that has not occurred so far. ■

### 4.3 Patterns of Type $\circ|\circ$

First observe that for the encoding of the formula in the reduction for  $\circ+\circ$  each repetition was taken at most twice. Hence, we can hardcode these repetitions in our reduction by replacing  $\alpha^+$  by  $(\alpha | \alpha\alpha)$ . Thus, we only show the changes in the reduction compared to the results given in Section 4.2.

For the encoding of the INPUT gate we replace every  $1^+$  by  $(1 | 11)$  and  $0^+$  by  $(0 | 00)$ . While the encoding of the AND gate stays the same, we change the definition of  $p_g$  for the OR gate to the following while  $u_g, q_g$ , and  $t_g$  remain unchanged:

$$p_g := (u_1 GGu_2 Gu_1 GGu_2 | u_1 GGu_2) G(q_1 GGp_2) G(p_1 GGq_2) G(u_1 GGu_2 Gu_1 GGu_2 | u_1 GGu_2)$$

The correctness proof for  $\circ+\circ$  directly shows the correctness of this encoding, too. Only the size bound for the pattern  $p_r$  changes.

**Lemma 4.7.**  $|u_r|, |t_r|, |q_r| \in \mathcal{O}(5^d s \log s)$  and  $|p_r| \in \mathcal{O}(8^d s \log s)$ .

*Proof.* Since we only changed the definition of  $p_g$  we get  $|u_g|, |t_g|, |q_g| \in \mathcal{O}(|p_g|)$  and thus only have to analyse the size of  $p_g$ :

$$\begin{aligned} |p_g| &\leq 6|u_1| + 6|u_2| + |q_1| + |q_2| + |p_1| + |p_2| + \mathcal{O}(\log s) \leq 8|p_1| + 8|p_2| \\ &\leq 8^{d(F_g)} \mathcal{O}(\log s + s \log s) = \mathcal{O}(8^{d(F_g)} s \log s) \end{aligned}$$

With the same argument as in the proof of Lemma 4.5. □

For the helper gadgets required for the application of Theorem 4.6, we also write down the number of repetitions explicitly, as for the encoding of the formula:

$$\begin{aligned} S &:= \gamma \\ S' &:= \gamma\gamma \\ \eta &:= (\gamma | \gamma\gamma) \\ U_2 &:= \gamma(u_r | u_r\gamma u_r)\gamma \\ U_3 &:= (\gamma | \gamma\gamma)(u_r | u_r\gamma u_r | u_r\gamma u_r\gamma u_r)\gamma \end{aligned}$$

The correctness of these gadgets follows directly from the correctness of the gadgets for  $\circ+\circ$ . ■

### 4.4 Patterns of Type $\circ\star$

**ENCODING THE FORMULA** In the two previous sections we have seen the reductions for  $\circ+\circ$  and  $\circ|\circ$ . Both reductions are related very closely. Now we extend these techniques to patterns of type  $\circ\star$ . For this we first observe that a pattern as  $\sigma^+$  can be seen as short-hand for  $\sigma\sigma^*$ . Thus, the encoding of the INPUT and AND gates stays the same as in Section 4.2 since this definition is quite general.

Only the encoding of the OR gate has to be modified since we cannot transfer it directly such that it applies to this reduction. But it suffices to simulate the  $(u_1 GGu_2 G)^+$  by a pattern of type  $\circ\star$ . For this we use a similar approach as for the Kleene Plus of a single symbol but we use the starred version of a text where every symbol is starred:

**Definition 4.8** (Starred Version of a Text). *Let  $t = t_1 \dots t_n$  be a text of length  $n$ . The starred version  $t^\star$  is a pattern of type  $\circ\star$  and defined as follows:*

$$t^\star := t_1^\star t_2^\star \dots t_n^\star$$

By this it suffices to change the definition of  $p_g$  as follows such that we can reuse the  $t_g$ ,  $u_g$ , and  $q_g$  from the reduction for  $\circ+\circ$ :

$$p_g := (u_1^\star G G u_2^\star) G (u_1 G G u_2) G (q_1 G G p_2) G (p_1 G G q_2) G (u_1 G G u_2) G (u_1^\star G G u_2^\star)$$

**Lemma 4.9** (Correctness of the Construction). *For all assignments  $a, b$  and gates  $g$ :*

- $F_g(a, b) = \text{true} \iff t_g(a) \in \mathcal{L}(p_g(b))$
- $t_g(a) \in \mathcal{L}(q_g)$
- $u_g \in \mathcal{L}(q_g) \cap \mathcal{L}(p_g(b))$

*Proof.* Again the proof of the last two claims follows directly from the encoding of the gates. Since the definition of the INPUT and AND gate is the same as for  $\circ+\circ$ , we only show the inductive step for the OR gate. Recall, that the text is defined as

$$t_g := (u_1 G G u_2) G (u_1 G G u_2) G (t_1 G G t_2) G (u_1 G G u_2) G (u_1 G G u_2).$$

“ $\Rightarrow$ ”  $F_g(a, b) = F_{g_1}(a, b) \vee F_{g_2}(a, b) = \text{true}$ . Assume w.l.o.g. that  $F_{g_1}(a, b) = \text{true}$ , the other case is symmetric. By assumption  $t_1 \in \mathcal{L}(p_1)$ . We match the first sequence of starred symbols to the empty string  $\varepsilon$ . Then we match  $u_1 G G u_2 G$  to each other. By the third claim above we can match  $u_1 G G u_2$  to  $q_1 G G p_2$ . Then by the assumption and the second claim, we match  $t_1 G G t_2$  to  $p_1 G G q_2$ . The remaining part of the text is matched in the canonical way to the pattern while the starred sequence matches the original text.

“ $\Leftarrow$ ” Assume  $t_g \in \mathcal{L}(p_g)$ . We first observe, that the  $G$ s in the pattern have to be matched to these strings in the text and that the text was matched completely. Since the first non-starred  $GG$  in the pattern has to be matched to a  $GG$  in the text, one can easily see that the first starred sequence has to be matched to the empty string or to  $u_1 G G u_2 G$ . The same argument holds for the other starred sequence. Thus it remains to check four different cases:

- The first sequence produced the empty string and the last sequence did not. The other way is symmetric. Then the remaining part of the pattern  $p'$  has to match the remaining part of the text  $t'$ :

$$\begin{aligned} t' &= (u_1 G G u_2) G (u_1 G G u_2) G (t_1 G G t_2) G (u_1 G G u_2) \\ p' &= (u_1 G G u_2) G (q_1 G G p_2) G (p_1 G G q_2) G (u_1 G G u_2) \end{aligned}$$

Since the  $G$ s in the pattern have to match  $G$ s in the text, we get  $t_1 \in \mathcal{L}(p_1)$  and thus a satisfying assignment.

- Both starred sequences produce a non-empty string, i.e. their non-starred version. Then the remaining text in between contains three  $GG$ s to which the four  $GG$ s of the remaining pattern have to be matched. A contradiction.
- Both starred sequences produce the empty string. Since  $u_1$  and  $u_2$  are strings the following text  $t'$  has to be matched by the remaining pattern  $p'$ :

$$\begin{aligned} t' &= (u_1 GG u_2) G (t_1 GG t_2) G (u_1 GG u_2) \\ p' &= (q_1 GG p_2) G (p_1 GG q_2). \end{aligned}$$

When taking a closer look at the definition of the universal pattern  $q_h$  and the universal text  $u_h$ , one can see that they only differ at the definition of the INPUT gates. Thus, we cannot match  $q_h$  to something different than  $u_h$  in the above setting. These observations imply  $u_2 G t_1 GG t_2 G u_1 \in \mathcal{L}(p_2 G p_1)$ . We use the following claim to obtain a contradiction for this case: The number of symbol changes for every word in  $\mathcal{L}(p_2 G p_1)$  is bounded by  $A(p_1) + A(p_2) + \ell + 2$  with  $\ell = A(G)$ . But the text has  $2A(u_1) + 2A(u_2) + 4\ell + 7$  symbol changes and thus cannot be matched by the pattern by Claim 4.4.1.  $\square$

**Definition 4.10** (Symbol Changes). *We define  $A(t)$  to be the number of symbol changes in the text  $t$ : Define  $A(\sigma) := 0$  for any symbol  $\sigma$  and  $A(t_1 \dots t_{n-1} t_n) := A(t_1 \dots t_{n-1}) + \llbracket t_{n-1} \neq t_n \rrbracket$ . For patterns  $p$  we define  $A(p) = \max_{t \in \mathcal{L}(p)} A(t)$ .*

$\triangleright$  Claim 4.4.1.  $A(u_g) = A(t_g) = A(q_g)$  and  $2A(u_g) > A(p_g)$ .

*Proof.* We first observe  $A(u_g) = A(t_g) = A(q_g)$  since their definitions only differ for the INPUT gate for which the claim holds. We show the main claim by a structural induction on gate  $g$ .

**INPUT Gate** We have  $A(u_g) = A(p_g) = 1$  and thus the claim holds.

**AND Gate**  $A(u_g) = A(u_1) + A(u_2) + A(G) + 2$  since all texts and patterns start with 0 and end with 1. Thus,  $A(p_g) = A(p_1) + A(p_2) + A(G) + 2$  for the same reason and hence the claim follows inductively.

**OR Gate** From  $A(u_g) = 5A(u_1) + 5A(u_2) + 14A(G) + 18$  we get

$$\begin{aligned} 2A(u_g) &= 10A(u_1) + 10A(u_2) + 28A(G) + 36 \\ &= 5A(u_1) + 5A(u_2) + 5A(u_1) + 5A(u_2) + 28A(G) + 36 \\ &\stackrel{\text{IH}}{>} 5A(u_1) + 5A(u_2) + 2.5A(p_1) + 2.5A(p_2) + 28A(G) + 36 \\ &> 5A(u_1) + 5A(u_2) + A(p_1) + A(p_2) + 17A(G) + 22 = A(p_g) \end{aligned} \quad \square$$

**Lemma 4.11.**  $|u_r|, |t_r|, |q_r| \in \mathcal{O}(5^d s \log s)$  and  $|p_r| \in \mathcal{O}(6^d s \log s)$ .

*Proof.* As for the modified encodings in Section 4.3 we have  $|u_g|, |t_g|, |q_g| \in \mathcal{O}(|p_g|)$ . Due to the changed definition of  $p_g$  we get:

$$\begin{aligned} |p_g| &\leq 4|u_1| + 4|u_2| + |q_1| + |q_2| + |p_1| + |p_2| + \mathcal{O}(\log s) \leq 6|p_1| + 6|p_2| + \mathcal{O}(\log s) \\ &\leq 6^{d(F_g)} \mathcal{O}(\log s + s \log s) = \mathcal{O}(6^{d(F_g)} s \log s) \end{aligned} \quad \square$$

FINAL REDUCTION Again we make use of Theorem 4.6 by defining the helper gadgets as follows:

$$\begin{aligned} S &:= \gamma\gamma\gamma \\ S' &:= \delta\gamma\gamma\gamma \\ \eta &:= \delta^*\gamma^+ \\ U_2 &:= \gamma^+ u_r^* \gamma^+ u_r \gamma^+ \\ U_3 &:= \delta^* \gamma^+ u_r^* \gamma^+ u_r \gamma^+ u_r \gamma^+ \end{aligned}$$

Recall, that  $\gamma^+$  is syntactic sugar for  $\gamma\gamma^*$ . ■

## 4.5 Patterns of Type $\circ+|$

ENCODING THE FORMULA For the encoding of the formula we again reuse the encoding of the INPUT and AND gate from the reduction for patterns of type  $\circ+\circ$ . This is because only the encoding of the OR gate introduces the operator of depth three. Hence, we have to use a different approach to encode the OR. Since patterns of type  $\circ+|$  can match arbitrary string, we have to adopt the encodings of the OR gate a bit while the main idea still stays the same:

$$\begin{aligned} t_g &:= 0G(t_1GGu_2)G(u_1GGt_2)G1 \\ u_g &:= 0G(u_1GGu_2)G(u_1GGu_2)G1 \\ q_g &:= 0G(q_1GGu_2)G(u_1GGq_2)G1 \\ p_g &:= (0|1|2)^+G(p_1GGp_2)G(0|1|2)^+ \end{aligned}$$

**Lemma 4.12** (Correctness of the construction). *For all assignments  $a, b$  and gates  $g$ :*

- $F_g(a, b) = \text{true} \iff t_g(a) \in \mathcal{L}(p_g(b))$
- $t_g(a) \in \mathcal{L}(q_g)$
- $u_g \in \mathcal{L}(q_g) \cap \mathcal{L}(p_g(b))$

*Proof.* Again we only show the inductive step for the OR case of the first claim.

“ $\Rightarrow$ ”  $F_g(a, b) = F_{g_1}(a, b) \vee F_{g_2}(a, b) = \text{true}$ . Assume w.l.o.g. that  $F_{g_1}(a, b) = \text{true}$ , the other case is symmetric. The first repetition is transformed into the initial 0. Then we match  $p_1GGp_2$  to  $t_1GGu_2$  by our third claim of the lemma and the assumption that  $t_1 \in \mathcal{L}(p_1)$ . Since the text only consists of symbols from  $\{0, 1, 2\}$ , the remaining part  $u_1GGt_2G1$  can be matched by the second repetition.

“ $\Leftarrow$ ” Assume  $t_g \in \mathcal{L}(p_g)$ . Since the GGs in the text have to match one of the two GG in the text there are only two possible ways, how the text was matched by the pattern. If the GG of the pattern matched the first GG of the text, then also the first G of the text and the first G of the pattern matched each other. Hence,  $Gt_1GG \in \mathcal{L}(Gp_1GG)$  and this

implies  $t_1 \in \mathcal{L}(p_1)$ . The induction hypothesis guarantees a satisfying assignment. The analog arguments hold for the second case when the  $GG$  in the pattern matches the second  $GG$  in the text. Then the last  $G$  of the text and the pattern have to match and this implies  $t_2 \in \mathcal{L}(p_2)$ .  $\square$

**Lemma 4.13.**  $|t_r|, |u_r|, |q_r| \in \mathcal{O}(2^d s \log s)$  and  $|p_r| \in \mathcal{O}(s \log s)$ .

*Proof.* Again we have  $\mathcal{O}(|u_g|) = \mathcal{O}(|t_g|) = \mathcal{O}(|q_g|)$ . Hence, we only analyse  $u_g$ :

$$|u_g| \leq 2|u_1| + 2|u_2| + \mathcal{O}(\log s) \leq 2^{d(F_g)} \mathcal{O}(\log s + s \log s) = \mathcal{O}(2^{d(F_g)} s \log s)$$

With the same argument as for the previous size bounds. For the size of the pattern we get  $|p_g| \leq |p_1| + |p_2| + \mathcal{O}(\log s) \leq \mathcal{O}(s \log s)$ .  $\square$

**FINAL REDUCTION** As for the patterns before, we use Theorem 4.6 to conclude the proof of Lemma 4.3 for this pattern type.

$$\begin{aligned} S &:= \gamma \alpha \gamma \alpha \gamma \alpha \\ S' &:= \delta \gamma \alpha \gamma \alpha \gamma \alpha \\ \eta &:= (\alpha | \gamma | \delta)^+ \\ U_2 &:= (\alpha | \gamma)^+ (\alpha | 0 | 1 | 2)^+ (\alpha | \gamma)^+ \alpha u_r (\alpha | \gamma)^+ \\ U_3 &:= (\alpha | \gamma | \delta)^+ (\alpha | 0 | 1 | 2)^+ (\alpha | \gamma)^+ (\alpha | 0 | 1 | 2)^+ (\alpha | \gamma)^+ \alpha u_r (\alpha | \gamma)^+ \end{aligned}$$

The correctness follows directly by the definition of these gadgets.  $\blacksquare$

## 4.6 Patterns of Type $\circ|+$

**ENCODING THE FORMULA** To reuse the definitions from the previous sections we have to allow unary alternatives. By this we can see the pattern  $\sigma^+$  as a pattern of type  $|+$  for some symbol  $\sigma \in \Sigma$ . This is reasonable since we can replace  $\sigma^+$  by  $(\sigma | \sigma^+)$  which represents exactly the same language as just  $\sigma^+$ . One could also use a fresh symbol  $\alpha$  which will never appear in the text and replace  $\sigma^+$  by  $(\alpha | \sigma^+)$ .

In the previous encodings of the OR gate we made use of the fact that we can define a pattern that can match  $T$  and  $TT$  for some text  $T$ . But for patterns of type  $\circ|+$  this is not that easily possible since we cannot omit symbols and every sub-pattern of the concatenation has to produce at least one symbol. For this we introduce the barred version of a text that transforms it into a pattern such that we can match the whole concatenation to the repetition of the same symbol and still to the original text.

**Definition 4.14** (Barred Version of a Text). *Let  $\tau$  be a symbol and  $t = t_1 \dots t_n$  be a text of length  $n$ . We define the barred version of  $t$  as a pattern of type  $\circ|$  as follows:*

$$\bar{t}^\tau := (t_1 | \tau)(t_2 | \tau) \dots (t_n | \tau)$$

Using this we change the encoding of the OR gate to the following:

$$\begin{aligned}
t_g &:= 0^{|u_1 GGu_2 G|+1} (u_1 GGu_2) G(t_1 GGt_2) G(u_1 GGu_2) 1^{|Gu_1 GGu_2|+1} \\
u_g &:= 0^{|u_1 GGu_2 G|+1} (u_1 GGu_2) G(u_1 GGu_2) G(u_1 GGu_2) 1^{|Gu_1 GGu_2|+1} \\
q_g &:= 0^{|u_1 GGu_2 G|+1} (u_1 GGu_2) G(q_1 GGq_2) G(u_1 GGu_2) 1^{|Gu_1 GGu_2|+1} \\
p_g &:= 0^+ \overline{u_1 GGu_2 G}^0 (q_1 GGp_2) G(p_1 GGq_2) \overline{Gu_1 GGu_2}^1 1^+
\end{aligned}$$

**Lemma 4.15** (Correctness of the construction). *For all assignments  $a, b$  and gates  $g$ :*

- $F_g(a, b) = \text{true} \iff t_g(a) \in \mathcal{L}(p_g(b))$
- $t_g(a) \in \mathcal{L}(q_g)$
- $u_g \in \mathcal{L}(q_g) \cap \mathcal{L}(p_g(b))$

*Proof.* Again we only show the proof for the OR gate in the first claim.

“ $\Rightarrow$ ”  $F_g(a, b) = F_{g_1}(a, b) \vee F_{g_2}(a, b) = \text{true}$ . Assume w.l.o.g. that  $F_{g_1}(a, b) = \text{true}$ , the other case is symmetric. We match the first barred text to a repetition of 0s which is possible by the definition of the barred text. Then  $q_1 GGp_2$  matches  $u_1 GGu_2$  by the third claim of the lemma. Now  $p_1$  matches  $t_1$  by the induction hypothesis and  $t_2$  is matched to  $q_2$  by the second claim of the lemma. Then the second barred pattern matches its original text while the repetition of 1s is matched by  $1^+$ .

“ $\Leftarrow$ ” Assume  $t_g \in \mathcal{L}(p_g)$ . Since the whole text has to be matched and the Gs in the pattern have to match Gs in the text, there are three possibilities how the GGs of the pattern can be matched to the GGs in the text:

- The first GG of the pattern matches the first GG of the text and the second of the text is matched by the second of the pattern. But this implies that  $u_2 Gt_1 \in \mathcal{L}(p_2 Gp_1)$  and since the G can only match itself, we get  $t_1 \in \mathcal{L}(p_1)$  which gives us a satisfying assignment by the induction hypothesis.
- The first GG of the pattern matches the first GG of the text and the second GG of the pattern matches the third GG of the text. We get  $u_2 Gt_1 GGt_2 Gu_1 \in \mathcal{L}(p_2 Gp_1)$ . Here we can use the same argument as for  $\circ\star$  and show that the number of symbol changes for every word in  $\mathcal{L}(p_2 Gp_1)$  is at most  $A(p_1) + A(p_2) + A(G) + 2$  while the text has  $2A(u_1) + 2A(u_2) + 4A(G) + 6$  symbol changes. Analogous to Claim 4.4.1 we can show that this case cannot occur since  $2A(u_h) > A(p_h)$ .
- The first GG of the pattern matches the second GG of the text and the third of the text is matched to the second of the pattern. This case is symmetric to the first case and implies  $t_2 \in \mathcal{L}(p_2)$ .  $\square$

**Lemma 4.16.**  $|u_r|, |t_r|, |q_r|, |p_r| \in \mathcal{O}(5^d s \log s)$ .

*Proof.* We first observe  $\mathcal{O}(|u_g|) = \mathcal{O}(|t_g|) = \mathcal{O}(|q_g|)$  and  $|p_g| \in \mathcal{O}(|u_g|)$ . Hence, we only analyse the size of  $u_g$ :

$$|u_g| \leq 5|u_1| + 5|u_2| + \mathcal{O}(\log s)$$

The claim follows by the same argument as before.  $\square$

**FINAL REDUCTION** For the definition of the helper gadgets we set  $\ell := |u_r|$  to simplify notation. Let  $\gamma$  and  $\alpha$  be fresh symbols.

$$\begin{aligned} S &:= \gamma \alpha^{\ell+5} \gamma \alpha^{\ell+5} \gamma \alpha^5 \\ S' &:= \gamma S \\ \eta &:= \gamma^+ \alpha^+ \gamma \alpha^+ \gamma \alpha^+ \\ U_2 &:= \gamma \alpha^+ (\alpha | \gamma) \alpha^+ (\alpha | \gamma) \alpha^+ \bar{u}_r^{-\alpha} \gamma \alpha^+ (\alpha | \gamma) \alpha^+ (\alpha | \gamma) \alpha^+ u_r S \\ U_3 &:= \gamma^+ \alpha^+ (\alpha | \gamma) \alpha^+ (\alpha | \gamma) \alpha^+ \bar{u}_r^{-\alpha} U_2 \end{aligned}$$

With Theorem 4.6 this concludes the proof of the corresponding part of Lemma 4.3.  $\blacksquare$

## 4.7 Proof of Theorem 4.6

We assume the following properties of the helper gadgets:

- $S$  is a proper substring of  $S'$  and  $S, S' \in \mathcal{L}(\eta)$ .
- $S, S'$ , and  $\eta$  only consist of fresh symbols.
- $S'$  has a non-empty substring that cannot be matched by  $U_2$ .
- $|S'|, |U_2|, |U_3|, |\eta| \in \mathcal{O}(|u_r|)$ .
- $\{Su_r S, Su_r Su_r S\} \in \mathcal{L}(U_2)$
- $\{S, S'\} \circ \{u_r S, u_r Su_r S, u_r Su_r Su_r S\} \in \mathcal{L}(U_3)$
- For all  $t$ :  $Su_r S' t \notin \mathcal{L}(U_3)$
- Either (1)  $\mathcal{L}(U_3) \subseteq \mathcal{L}(\eta(u_r S)^+)$  or (2) for any word  $t \in \mathcal{L}(U_3)$  there are at most four disjoint intervals that consist only of symbols that occur in  $S$  or  $S'$ . The last interval ends at the end of  $t$ . Analogously for three intervals for  $U_2$ .

Let  $A = \{a^{(1)}, \dots, a^{(n)}\}$  be the first set and  $B = \{b^{(1)}, \dots, b^{(m)}\}$  be the second set of half-assignments. Inspired by the reduction in Section 3.4 in the full version of [BI16] we define the final text and pattern as follows:

$$\begin{aligned} t &:= \bigodot_{i=1}^{3n} \left( S' u_r Su_r Su_r S t(a^{(i)}) Su_r Su_r Su_r Su_r \right) \\ p &:= Su_r Su_r Su_r Su_r \bigodot_{j=1}^m \left( U_3 u_r \eta q_r S p(b^{(j)}) U_2 q_r \right) Su_r Su_r Su_r Su_r \end{aligned}$$

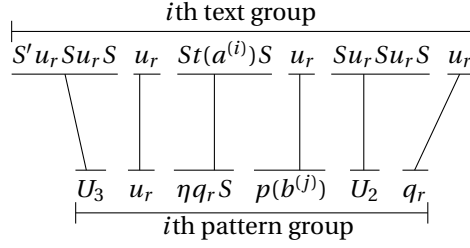
Where we set  $a^{(j)} = a^{(j \bmod n)}$  for  $j \in [n+1, 3n]$ . We call the concatenations in  $t$  and  $p$  for each  $i$  and  $j$  the  $i$ th text group and the  $j$ th pattern group, respectively.



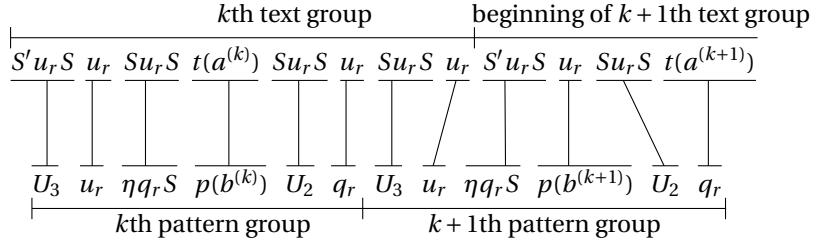
CORRECTNESS AND SIZE BOUND We split the proof of the theorem in three lemmas, about the completeness, correctness and the size bound.

**Lemma 4.17.** *If there are  $a^{(k)}$  and  $b^{(l)}$  such that  $F(a^{(k)}, b^{(l)}) = \text{true}$ , then  $t \in \mathcal{M}(p)$ .*

*Proof.* Assume w.l.o.g.  $a^{(k)}$  and  $b^{(k)}$  satisfy  $F$ . Otherwise we have to shift the indices for the text and the pattern in the proof accordingly. We match the prefix of  $p$  to the corresponding suffix of the  $n$ th group of the text. Then we match the  $i$ th group of the text to the  $i$ th group of the pattern for  $i = 1, \dots, k-1$  as follows using the assumptions:



Then we match the  $k$ th and  $k+1$ th group of the pattern to the  $k$ th group of the text and a part of the  $k+1$ th group of the text as follows, which is again possible by the assumptions:



Now we shift the remaining part  $t'$  of the text such that it becomes easier to show which part the remaining pattern matches:

$$\begin{aligned} t' &= Su_r Su_r Su_r Su_r \bigcirc_{i=n+k+2}^{3n} \left( S'u_r Su_r Su_r St(a^{(i)}) Su_r Su_r Su_r Su_r \right) \\ &= \bigcirc_{i=n+k+2}^{3n} \left( Su_r Su_r Su_r Su_r S'u_r Su_r Su_r St(a^{(i)}) \right) Su_r Su_r Su_r Su_r \end{aligned}$$

Now we match the  $i$ th group of the pattern to the  $i$ th group of the text  $t'$  for  $i = n+k+2, \dots, m$ . We do this by matching  $Su_r Su_r Su_r S$  to  $U_3$  and  $Su_r S$  to  $U_2$ . The other parts are matched in a straightforward way. Finally, the suffix of the pattern is matched to the prefix of the  $m+1$ th group of the text in the canonical way.  $\square$

**Lemma 4.18.** *If  $t \in \mathcal{M}(p)$ , then there are  $a^{(k)}$  and  $b^{(l)}$  such that  $F(a^{(k)}, b^{(l)}) = \text{true}$ .*

*Proof.* By the design of the pattern and the text, there must be a  $j \leq n$  such that the prefix of the pattern is matched to the suffix of the  $j-1$ th group of the text. Furthermore, we know that the suffix of the pattern has to match the suffix of some text group because nowhere else the four  $Su_r$  could be matched. By this we know that not all groups of the

pattern and the text match each other precisely. Meaning, there is a text group  $k$  and a pattern group  $l$  such that the pattern group does not match the whole text group or matches more than this group. We choose the first of these groups (i.e. the pair with smallest  $k$  and  $l$ ).

Since all prior groups have been matched precisely,  $U_3$  has to match (some prefix of)  $S'u_rSu_rSu_rS$ . Because the  $S$  have to be aligned to each other and we know by our assumption that either (1) all substrings between  $S$  or  $S'$  are equal and thus  $t(a^{(l)})$  cannot be matched, or (2) for all matched word there are at most four disjoint intervals of text positions where  $S$  and  $S'$  could be located. We observe further that  $U_3$  can only match (some prefix of)  $S'u_rSu_rS$  because otherwise the following  $u_r$  in the pattern has to match  $t(a^{(i)})$ .

- If  $U_3$  matches  $S'u_rS$ , then the following  $u_r$  in the pattern and text match each other. Hence,  $\eta q_rS$  has to match  $Su_rS$  because the  $S$  have to be aligned. This implies that  $p(b^{(j)})$  matches  $t(a^{(i)})$  and by our assumptions this gives us a satisfying assignment for  $F$ .
- Now assume for the sake of a contradiction that  $U_3$  matches  $S'u_rSu_rS$ . Then the following  $u_r$  match each other such that  $q_r$  matches  $t(a^{(i)})$  and  $p^{(j)}$  matches  $u_r$ . Observe that  $U_2$  can only match  $Su_rSu_rS$ . Assume it matches only  $Su_rS$ , then  $q_r$  matches the second last  $u_r$  and  $U_3$  has to match a word with  $Su_rS'$  as prefix. A contradiction to one of the assumptions.  $U_2$  can also not match more than  $Su_rSu_rS$ , because then it has to match  $S'$  which is not possible since  $S'$  contains a substring that cannot be matched by  $U_2$  by assumption. But then we have that  $q_r$  matches the last  $u_r$  of the text group, and the text and the pattern group match each other. This is a contradiction to our assumption that these groups do not match.  $\square$

**Lemma 4.19.** *The final text has length  $\mathcal{O}(n(|u_r|+|t(0)|))$  and the pattern has size  $\mathcal{O}(m(|u_r|+|p(0)|+|q_r|))$ .*

By this we conclude the proof of Theorem 4.6.  $\blacksquare$

## Conditional Lower Bounds for Membership

In the previous Chapter 4 we have seen conditional lower bounds for *pattern matching*. In this chapter we show lower bounds for the *membership* problem. Recall that for the membership problem the whole string has to be matched while for the matching problem only a substring has to be matched.

**Theorem 5.1.** *Membership with homogenous patterns of the following types cannot be solved in the stated running time even for constant sized alphabet, unless FPH as stated in Hypothesis 3 is false:*

$\circ\star$	$\circ+\circ$	$\circ \circ$	$\circ+ $	$\circ +$	$ +\circ$
$\mathcal{O}\left(\frac{nm}{\log^{76} n}\right)$	$\mathcal{O}\left(\frac{nm}{\log^{72} n}\right)$	$\mathcal{O}\left(\frac{nm}{\log^{81} n}\right)$	$\mathcal{O}\left(\frac{nm}{\log^{27} n}\right)$	$\mathcal{O}\left(\frac{nm}{\log^{72} n}\right)$	$\mathcal{O}\left(\frac{nm}{\log^{20} n}\right)$

Instead of showing the lower bounds and thus reductions from scratch, we reduce pattern matching to membership. By this we can reuse the results from the previous chapter. Although our reductions change the size of the pattern and the text, we can use them since they do not change the asymptotic size of the text and pattern as defined in the previous chapter.

**Lemma 5.2** (Reducing Pattern Matching to Membership). *Let  $t$  be a text and  $p$  be a pattern of type  $T$ . For the mentioned pattern types we can construct a text  $t'$  and a pattern  $p'$  such that*

$$t \in \mathcal{L}(p) \iff t' \in \mathcal{M}(p')$$

with the following size bounds for  $t'$  and  $p'$ :

	$\circ\star$	$\circ+\circ$	$\circ \circ$	$\circ+ $	$\circ +$
$ t' $	$\mathcal{O}( t )$	$\mathcal{O}( \Sigma  t )$	$\mathcal{O}( t )$	$\mathcal{O}( t )$	$\mathcal{O}( t )$
$ p' $	$\mathcal{O}( t  +  p )$	$\mathcal{O}( \Sigma ( t  +  p ))$	$\mathcal{O}( \Sigma  t  +  p )$	$\mathcal{O}( \Sigma  +  p )$	$\mathcal{O}( t  +  p )$
<i>Proof</i>	Section 5.1.1	Section 5.1.2	Section 5.1.3	Section 5.1.4	Section 5.1.5

*Proof of Theorem 5.1.* From Lemma 4.3 we know that we can reduce FORMULA-PAIR to pattern matching. With the above lemma, we can reduce this pattern matching instance to membership without changing the size. Thus the running time transfers and the theorem follows directly from Theorem 4.1. We show the proof for  $|+\circ$  in Section 5.2.  $\square$

## 5.1 Reducing Pattern Matching to Membership

We assume that  $t$  is the text and  $p$  is the pattern as in Lemma 5.2. Let w.l.o.g.  $\Sigma = \{1, \dots, s\}$  be the alphabet of the text and the pattern. When using  $\Sigma$  as a pattern we mean  $(1 \mid \dots \mid s)$ . Likewise for  $\Sigma^+$  which is shorthand for  $(1 \mid \dots \mid s)^+$ . We extend this in the natural way to other sets of symbols.

### 5.1.1 Patterns of Type $\circ\star$

Recall the starred version of a text as in Definition 4.8 introduced in Section 4.4. With this we define the text  $t'$  and the pattern  $p'$  as follows:

$$\begin{aligned} t' &:= t \\ p' &:= t^* p t^* \end{aligned}$$

Then we obviously have:

$$t \in \mathcal{M}(p) \iff t' \in \mathcal{L}(p')$$

Because the pattern  $p$  has to match some substring of  $t' = t$ .

### 5.1.2 Patterns of Type $\circ+\circ$

We define the reduction in two steps. First we encode every symbol in a special way such that we can define a universal pattern  $U$  of type  $\circ+$  in step two matching any symbol. For this we define a function  $f: \Sigma \rightarrow \Sigma^{s+1}$  with  $x \mapsto 1 \cdots (x-1)xx(x+1) \cdots s$ . We extend this definition in the natural way to texts by applying it to every symbol. We also extend this definition to patterns of type  $\circ+\circ$  such that we apply  $f$  to every symbol of the pattern. Observe that this does not change the type of the pattern. This encoding preserves the membership property for the matching language:

$$t \in \mathcal{M}(p) \iff f(t) \in \mathcal{M}(f(p))$$

In the second step we change from pattern matching to membership. For this we set  $U := 1^+ 2^+ \cdots s^+$  and  $R := 12 \cdots s$ . From the definition we directly get the following properties for all  $\sigma \in \Sigma$ :

$$\begin{aligned} R &\in \mathcal{L}(U) \\ f(\sigma) &\in \mathcal{L}(U) \\ R &\notin \mathcal{L}(f(\sigma)) \end{aligned}$$

Using this we define the final text and pattern as follows:

$$\begin{aligned} t' &:= R^{|t|+1} f(t) R^{|t|+1} \\ p' &:= R^+ U^{|t|} f(p) U^{|t|} R^+ \end{aligned}$$

If  $t \in \mathcal{M}(p)$ , then there is a substring  $\hat{t}$  of  $t$  to which  $p$  is matched. By the above observations we can match  $f(p)$  to  $f(\hat{t})$ , which is a substring of  $f(t)$ . Then we use the

$U^{|t|}$  to match the not matched suffix and prefix of  $t$  and a part of  $R^{|t|+1}$ . The remaining repetitions of  $R$  are matched by the  $R^+$  in the beginning and the end.

If  $t' \in \mathcal{L}(p')$ , then  $f(p)$  has to match some substring of  $f(t)$  because  $f$  duplicates the original symbol and thus  $R$  cannot be matched, since there are no repetitions.

### 5.1.3 Patterns of Type $\circ|\circ$

Using  $L := 2^{\lceil \log |t| \rceil}$  we define the text and the pattern as follows with  $\alpha$  as a new symbol:

$$t' := \alpha^{3L-1} t \alpha^{3L-1}$$

$$p' := \bigcirc_{i=0}^{\log L} (\alpha^{2^i} | \alpha^{2^{i+1}}) (\Sigma \cup \{\alpha\})^L p (\Sigma \cup \{\alpha\})^L \bigcirc_{i=0}^{\log L} (\alpha^{2^i} | \alpha^{2^{i+1}})$$

If  $t' \in \mathcal{L}(p')$ , it is clear that there must be some substring of  $t$  which is matched by  $p$  because  $p$  cannot match strings containing  $\alpha$ . The other direction is not completely obvious. Observe that the first and last concatenation of  $p'$  can match  $\alpha^z$  for all  $z \in [2L-1, 4L-2]$ : We know that the binary encoding of a number  $k \in [0, 2L-1]$  has  $\leq \log L + 1$  bits. We use the alternatives in the concatenation to represent this binary encoding by choosing the left option  $\alpha^{2^i}$  if the  $i$ th bit is 0 and the right option  $\alpha^{2^{i+1}}$  if the bit is 1. But we can also see the options as choosing  $\varepsilon$  for the left side and  $\alpha^{2^i}$  for the right side but always appending  $\alpha^{2^i}$  independent from the value of the  $i$ th bit:

$$\bigcirc_{i=0}^{\log L} (\alpha^{2^i} | \alpha^{2^{i+1}}) \equiv \bigcirc_{i=0}^{\log L} \alpha^{2^i} (\varepsilon | \alpha^{2^i})$$

By the correctness of the binary encoding there are  $2L-1+k$  repetitions of  $\alpha$  produced by this procedure.

Now if  $t \in \mathcal{M}(p)$ , we know there is a substring of  $t$  that is matched to  $p$ . The not matched suffix and prefix of  $t$  are matched by the alternatives of length  $L$  directly before and after  $p$  in  $p'$ . The sub-patterns of these alternatives which do not match a part of the text  $t$  match some prefix and suffix of  $\alpha^{3L-1}$ . The remaining part is of length at least  $2L-1$  and at most  $3L-1 \leq 4L-2$  and can thus be matched by the above observations.

We observe  $L \leq 2|t|$  because of the rounding. Furthermore, the size of the pattern is increased by  $\mathcal{O}(\varepsilon \cdot L)$  for the alternatives of symbols and by

$$\sum_{i=0}^{\log L} 2^i + 2^{i+1} = 2^{\log L+1} - 1 + 2(2^{\log L+1} - 1) = \mathcal{O}(L)$$

for the concatenation of alternatives.

### 5.1.4 Patterns of Type $\circ+|$

We define:

$$t' := 1t1$$

$$p' := \Sigma^+ p \Sigma^+$$

If  $t' \in \mathcal{L}(p')$  we know that the first and last repetition of  $p'$  match at least the first and last symbol of the text. Thus, the pattern  $p$  has to match some substring of  $t$ . The other direction is trivial.

### 5.1.5 Patterns of Type $\circ|+$

With  $\alpha$  as a new symbol we define:

$$t' := \alpha^{|t|+1} t \alpha^{|t|+1}$$

$$p' := \alpha^+(1 | \cdots | s | \alpha)^{|t|} p (1 | \cdots | s | \alpha)^{|t|} \alpha^+$$

If  $t \in \mathcal{M}(p)$  then there is some substring of  $t$  that is matched by  $p$ . This part is also matched in the membership instance. The not matched prefix of  $t$  is matched by the sequence of alternatives. The remaining  $\alpha$ s in the text are matched by  $\alpha^+$ .

If  $t' \in \mathcal{L}(p')$ , then  $p$  has to match some part of  $t$  because it cannot match any  $\alpha$ . ■

## 5.2 Patterns of Type $|+|\circ$

In the last section we have seen the reduction from pattern matching to membership and thus concluded the proof for the lower bounds of these pattern types. As shown in the introduction and originally in [BI16] almost all patterns that are hard for pattern matching are also hard for membership. Except for the two patterns  $|\circ|$  and  $|\circ+$  for which we have shown faster algorithms in Chapter 3. They are easy (i.e. linear time solvable) for membership. But there is also one pattern that is easy for pattern matching while it does not have near linear time algorithms for membership. This so called *word break problem* considers patterns of type  $+|\circ$ . It can be seen as the task of splitting an input string into words occurring in a dictionary. It has the same complexity as  $\star|\circ$ . The problem can be solved in time  $\mathcal{O}(n(m \log m)^{1/3} + m)$  as shown in [BGL17]. They also show that this is optimal if all combinatorial  $k$ -clique algorithms need  $n^{k-o(1)}$  time. Hence, it is the only intermediate problem between quadratic and near linear time complexity for membership.

In this section we show the hardness for membership with pattern of type  $+|\circ$ . This is one possible extension of  $+|\circ$  to patterns of depth four. We only consider this pattern since  $\star+|\circ$  is identical to  $+|\circ$  from a complexity theoretic perspective and the hardness of  $\circ+|\circ$  comes from  $\circ+|$  for which we proved the hardness in Theorem 5.1.

**Theorem 5.3.** *Membership with patterns of type  $+|\circ$  and even constant sized alphabet cannot be solved in time  $\mathcal{O}(nm/\log^{20} n)$ , unless FPH as stated in Hypothesis 3 is false.*

**MAIN IDEA** As for the previous lower bounds we encode the formula and its evaluation by the text and the pattern. But since our pattern is represented by several dictionaries containing multiple words, we have to make sure that the text can be matched (if the formula is evaluated to true) and more important that the words are only used for their intended purpose. The main idea is that the encoding of each gate is enclosed by the root to gate path for this gate, we call it the trace in the following. This additional information

allows us to skip several parts of the text because all descendants of a gate  $g$  have  $g$  in their trace. During this *skipping phase* we “disable” the check of the formula whether it is satisfied or not. The encoding of the INPUT gates takes care that we do not care about the assignment of the variable if we are in this skipping phase. The encoding of the AND gate has to make sure that we carry forward whether we are in this skipping phase or the actual *matching phase*. For the OR gate this is somewhat different since we can ignore one of the sub-formulas if the other one is evaluated to true. We define the dictionary such that at least one sub-pattern matches the text if we are in the matching phase.

In the following we define several dictionaries  $D_g^M$  and  $D_g^S$  for each gate  $g$ . These dictionaries contain the words that are added to the final dictionary for this gate. The dictionary  $D_g^M$  describes the words that are used in the matching phase (corresponds to  $p_g$  from the previous chapter) while the words for the skipping phase are grouped together in  $D_g^S$  (corresponds to  $q_g$ ). We define  $D_g$  to be the dictionary containing all words and dictionaries  $D_{g'}^M$  and  $D_{g'}^S$  for each gate  $g'$  that appears in the formula  $F_g$ .

### 5.2.1 Encoding the Formula

As in the previous chapter we identify each gate with its ID, i.e. an integer in  $[2s]$ . For a gate  $g$  let  $\langle g \rangle$  be the binary encoding of the gate ID with  $\lfloor \log s \rfloor + 2 = \Theta(\log s)$  bits padded with zeros if necessary.

When considering a gate  $g$ , let  $h_0 h_1 \dots h_d$  be the path from the root  $r = h_0$  of  $F$  to the gate  $g = h_d$ . To simplify notation we identify each gate  $\hat{g}$  with  $2^{\langle \hat{g} \rangle} 2$ , i.e. its encoding enclosed by 2s that are used as separators.

**INPUT Gates** If  $F_g(a, b) = a_i$ , we define:

$$t_g := h_0 \dots h_d a_i h_d \dots h_0 \quad D_g^M := \{h_0 \dots h_d 1 h_d \dots h_0\}$$

If  $F_g(a, b) = b_i$ , we define

$$t_g := h_0 \dots h_d 1 h_d \dots h_0 \quad D_g^M := \{h_0 \dots h_d b_i h_d \dots h_0\}$$

We set  $D_g^S := \{h_i \dots h_d 0 h_d \dots h_i, h_i \dots h_d 1 h_d \dots h_i \mid i \in [d]\}$ .

**AND Gate** We define the texts and the corresponding dictionaries as follows:

$$\begin{aligned} t_g &:= h_0 \dots h_d h_d t_1 t_2 h_d h_d \dots h_0 \\ D_g^M &:= \{h_0 \dots h_d h_d, h_d h_d \dots h_0\} \\ D_g^S &:= \{h_i \dots h_d h_d h_0 \dots h_{i-1}, h_{i-1} \dots h_0 h_0 \dots h_{i-1}, h_{i-1} \dots h_0 h_d h_d \dots h_i \mid i \in [d]\} \end{aligned}$$

**OR Gate** We define the text and the additional dictionaries for  $g$  as follows:

$$\begin{aligned} t_g &:= h_0 \dots h_d h_d t_1 h_d h_d t_2 h_d h_d \dots h_0 \\ D_g^M &:= \{h_0 \dots h_d h_d, h_d h_d h_0 \dots h_d, h_d \dots h_0 h_d h_d \dots h_0\} \\ &\quad \cup \{h_0 \dots h_d h_d h_0 \dots h_d, h_d \dots h_0 h_d h_d, h_d h_d \dots h_0\} \\ D_g^S &:= \{h_i \dots h_d h_d h_0 \dots h_{i-1}, h_{i-1} \dots h_0 h_d h_d h_0 \dots h_{i-1}, h_{i-1} \dots h_0 h_d h_d \dots h_i, \mid i \in [d]\} \end{aligned}$$

## CORRECTNESS AND SIZE BOUND

**Lemma 5.4.** *For all assignments  $a, b$  and gates  $g$ : Let  $r = h_0, \dots, h_d = g$  be the trace of gate  $g$  for formula  $F$ .*

- $t_g(a) \in \mathcal{L}(h_0 \dots h_{i-1} (D_g(b))^+ h_{i-1} \dots h_0)$  for all  $i \in [d]$ .<sup>1</sup>
- $t_g(a) \notin \mathcal{L}(h_0 \dots h_{i-1} (D_g(b))^+ h_{j-1} \dots h_0)$  for all  $i \neq j \in [0, d]$ .<sup>2</sup>

*Proof.* One can easily show the correctness of the first claim by a structural induction on the output gate using the words for the skipping phase. Likewise we show the second case by a structural induction on the output gate.

**INPUT Gate** The statement is true by definition of the dictionary.

**AND Gate** By assumption the claim is true for  $g_1$  and  $g_2$ . But assume the claim is false for  $g$ . We can only match the “prefix”  $h_i \dots h_d h_d$  with the word  $h_i \dots h_d h_d h_0 \dots h_{i-1}$ . And analogously for the “suffix”. The joining part of  $t_1 t_2$  has to be matched by some  $h_{k-1} \dots h_0 h_0 \dots h_{k-1}$  for  $k \in [0, \dots, d]$  (possibly the empty string). Hence,  $t_1 \in \mathcal{L}(h_0 \dots h_{i-1} (D_g(b))^+ h_{k-1} \dots h_0)$  and  $t_2 \in \mathcal{L}(h_0 \dots h_{k-1} (D_g(b))^+ h_{j-1} \dots h_0)$ . But from  $i \neq j$  it follows that  $k \neq i$  or  $k \neq j$  and we have a contradiction.

**OR Gate** Again assume the claim is true for  $g_1$  and  $g_2$ . We have to match the “prefix”  $h_i \dots h_d h_d$  with the word  $h_i \dots h_d h_d h_0 \dots h_{i-1}$ . And analogously for the “suffix”. If the joining part of  $t_1 h_d h_d t_2$  was matched by  $h_{k-1} \dots h_0 h_d h_d h_0 \dots h_{k-1}$  for some  $k \in [d]$ , the same proof as for the AND gate applies. Otherwise,  $h_d \dots h_0 h_d h_d$  or  $h_d h_d h_0 \dots h_d$  was used. Let it w.l.o.g. be the first one. But since  $i \in [0, d]$  we have  $i \neq d + 1$  and hence the same argument as for the AND gate applies showing a contradiction for  $t_1$ .  $\square$

**Lemma 5.5** (Correctness of the Construction). *For all assignments  $a, b$  and gates  $g$ :  $F_g(a, b) = \text{true} \iff t_g(a) \in \mathcal{L}((D_g(b))^+)$ .*

*Proof.* We proof the claim by an induction on the output gate.

**INPUT Gate “ $\Rightarrow$ ”** Follows directly from the construction of the text and the dictionary.

**INPUT Gate “ $\Leftarrow$ ”** Since the words in  $D_g^S$  cannot match  $h_0$ , a word from  $D_g^M$  has to be chosen. Since the text can only be matched by the word in the dictionary if the gate evaluates to true, the result follows directly.

**AND Gate “ $\Rightarrow$ ”**  $F_g(a, b) = \text{true} = F_{g_1}(a, b) \wedge F_{g_2}(a, b) \Rightarrow F_{g_1}(a, b) = \text{true} \wedge F_{g_2}(a, b) = \text{true}$ . Hence, we can use  $D_1$  and  $D_2$  to match  $t_1$  and  $t_2$ , respectively. The remaining parts are matched by the words in  $D_g^M$ .

<sup>1</sup>  $A^+ := (a_1 | a_2 | \dots | a_{|A|})^+$  where  $A$  is a set of word  $a_i$ .

<sup>2</sup>  $h_0 h_{-1}$  and  $h_{-1} h_0$  denote the empty string in this context.



**AND Gate “ $\Leftarrow$ ”** If  $t_g \in \mathcal{L}(p_g)$ , then the  $h_d h_d$ s in the text have to be matched. Since we enumerated the gates, there are no other words than those in  $D_g^M$  and  $D_g^S$  that can match this part of the text. Furthermore, we know that the first and last occurrence of  $h_0$  is matched. Hence, the first and last trace has to be matched by words from  $D_g^M$ . It cannot be matched by word from other  $D_{g'}^M$  since they all have  $g' g'$  as part of the words. Since the last  $h_{g_1} h_{g_1}$  in  $t_1$  and the first  $h_{g_2} h_{g_2}$  in  $t_2$  have to be matched and this can only be done by words in  $D_1$  and  $D_2$ , we have that the whole  $t_1$  and  $t_2$  were matched by the corresponding dictionary. Otherwise, we had a contradiction to Lemma 5.4. This shows the correctness of the claim by the inductive hypothesis.

**OR Gate “ $\Rightarrow$ ”**  $F_g(a, b) = F_{g_1}(a, b) \vee F_{g_2}(a, b) = \text{true}$ . Assume w.l.o.g. that  $F_{g_1}(a, b) = \text{true}$ , the other case is symmetric. By assumption  $t_1 \in \mathcal{L}((D_1)^+)$ . We match the prefix of  $t_g$  in the obvious way. We match the  $h_d h_d$  in the middle and the following  $h_0 \dots h_d$  of  $t_2$  with the corresponding word from  $D_g^M$ . By the first claim of the previous lemma, we have  $t_2 \in \mathcal{L}(h_0 \dots h_d (D_g)^+ h_d \dots h_0)$ . Then we match the remaining suffix  $h_d \dots h_0 h_d h_d \dots h_0$  with the corresponding word from  $D_g^M$ .

**OR Gate “ $\Leftarrow$ ”**  $t \in \mathcal{L}((D_g)^+)$  implies that the first and last  $h_0$  have to be matched.

(a) Assume the first was matched by  $h_0 \dots h_d h_d$ . According to Lemma 5.4 the  $h_d h_d$  in the middle must be matched by  $h_d h_d h_0 \dots h_d$  and not by a word in  $D_g^S$ . But this implies  $F_{g_1}(a, b) = \text{true}$  by the induction hypothesis since the words from  $D_g \cup D_2 \supseteq D_g^M \cup D_g^S$  cannot be used to match substrings of  $t_1$  due to the occurrence of  $\hat{g} \hat{g}$  for gates  $\hat{g}$  in  $F_{g_1}$  in the text  $t_1$ .

(b) The first  $h_0$  of the text was matched by  $h_0 \dots h_d h_d h_0 \dots h_d$ . Then the middle  $h_d h_d$  must be matched by  $h_d \dots h_0 h_d h_d$  to not contradict the lemma. But the same lemma requires that the last  $h_0$  is matched by  $h_d h_d \dots h_0$  and hence the inductive hypothesis gives us a satisfying assignment since only words from  $D_2$  could be used.  $\square$

LENGTH OF THE TEXT AND THE PATTERN

**Lemma 5.6.** *We have the following size bounds:*

- $|t_r| \in \mathcal{O}(sd \log s) \subseteq \mathcal{O}(s^2 \log s)$
- $|D_r| \in \mathcal{O}(sd) \subseteq \mathcal{O}(s^2)$
- $\forall x \in D_r : |x| \in \mathcal{O}(d \log s) \subseteq \mathcal{O}(s \log s)$

*Proof.* The lemma follows directly from the encoding and the following observations:

- $|t_g| \leq |t_1| + |t_2| + \mathcal{O}(d \log s)$
- $|D_g^M| \in \mathcal{O}(1)$  and  $|D_g^S| \in \mathcal{O}(d)$
- $D_r = \bigcup_{\text{gates } g} D_g^M \cup D_g^S$   $\square$

## 5.2.2 Final Reduction

Let  $A = \{a^{(1)}, \dots, a^{(n)}\}$  be the first set and  $B = \{b^{(1)}, \dots, b^{(m)}\}$  be the second set of half-assignments.

**DEFINING THE OUTER OR** By the structure of the pattern it is very easy to select one specific assignment  $b^{(j)}$ . Furthermore, it should be clear that the text should be a concatenation of the  $t(a^{(i)})$ s separated by some special symbol. But it is not completely trivial how to match all the remaining symbols of the text. We solve this problem by blowing-up the text and the pattern such that we can distinguish between the following three states matching periods/states: (a) checking if two assignments satisfy the formula (b) ignoring the prefix (symbols before the actual match), and (c) ignoring the suffix (symbols after the actual match). Furthermore, a change between these states is only possible at the boundaries of the  $t(a^{(i)})$ s. By the design of the text we force the pattern to go through all three states.

**Definition 5.7** (Blow-Up of a Text). *Let  $t = t_1 \dots t_n$  be a text of length  $n$  and  $u$  be some arbitrary string. We define  $\overleftarrow{t}^u := ut_1ut_2 \dots ut_n$ . For a set of texts  $W = \{w_1, \dots, w_k\}$  we define  $\overleftarrow{W}^u := \{\overleftarrow{w}_1^u, \dots, \overleftarrow{w}_k^u\}$ .*

Using this we define the final text and pattern as follows:

$$\begin{aligned} t &:= \beta\gamma\delta \bigcirc_{i=1}^n \left( \overleftarrow{t(a^{(i)})}^{\alpha\beta\gamma} \alpha\beta\gamma\delta \right) \alpha\beta \\ p &:= p_1^+ \mid p_2^+ \mid \dots \mid p_m^+ \\ p_j &:= \beta\gamma x \alpha \mid \beta\gamma\delta \mid \overleftarrow{D_r(b^{(j)})}^{\alpha\beta\gamma} \mid \alpha\beta\gamma\delta\alpha\beta \mid \gamma x \alpha\beta \quad x \in \{0, 1, 2, \delta\} \end{aligned}$$

**CORRECTNESS AND SIZE BOUND** By proving the correctness and completeness of this final construction and its size bound we finish the proof of Theorem 5.3.

**Lemma 5.8.** *If there are  $a^{(k)}$  and  $b^{(l)}$  such that  $F(a^{(k)}, b^{(l)}) = \text{true}$ , then  $t \in \mathcal{L}(p)$ .*

*Proof.* We show that we can match  $t$  to  $p_1^+$ . We match the prefix of  $t$  and the first  $k-1$  groups of the text by repetitions of  $\beta\gamma x \alpha$  for all values  $x \in \{0, 1, 2, \delta\}$ . But the last three symbols of the  $k-1$ th group are matched by  $\beta\gamma\delta$ . This is possible since before and thus after each information carrying symbol (i.e. 0,1,2) there is an  $\alpha\beta\gamma$ . Then by Lemma 5.5 and the definition of the blow-up we get

$$\overleftarrow{t(a^{(k)})}^{\alpha\beta\gamma} \in \mathcal{L} \left( \left( \overleftarrow{D_r(b^{(l)})}^{\alpha\beta\gamma} \right)^+ \right)$$

The following  $\alpha\beta\gamma\delta\alpha\beta$  is matched by the corresponding pattern. The remaining symbols and groups are matched in a straight forward way by repetitions of  $\gamma x \alpha\beta$ .  $\square$

**Lemma 5.9.** *If  $t \in \mathcal{L}(p)$ , then there are  $a^{(k)}$  and  $b^{(l)}$  such that  $F(a^{(k)}, b^{(l)}) = \text{true}$ .*

*Proof.* We can already fix the  $l$  due to the structure of the pattern. There is no way to match the text just with words  $\beta\gamma x\alpha$  or  $\gamma x\alpha\beta$ . Thus, we must have made use of the words  $\beta\gamma\delta$  and  $\alpha\beta\gamma\delta\alpha\beta$ . But this application is only possible at the boundary of two text groups. Hence, let  $k$  be the first group that is not matched by  $\beta\gamma x\alpha$  or  $\gamma x\alpha\beta$ . Observe that we cannot directly switch to an application of the word  $\gamma x\alpha\beta$  since this is only possible if there is a  $\delta$ . Therefore, we get

$$\overleftarrow{t(a^{(k)})}^{\alpha\beta\gamma} \in \mathcal{L}\left(\left(\overleftarrow{D_r(b^{(l)})}^{\alpha\beta\gamma}\right)^+\right)$$

for some  $k \in [n]$ . But since the  $\alpha\beta\gamma$  always match to each other, we can ignore them and get

$$t(a^{(k)}) \in \mathcal{L}\left(\left(D_r(b^{(l)})\right)^+\right)$$

proving the claim by Lemma 5.5. □

**Corollary 5.10.** *The final text has length  $\mathcal{O}(nsd \log s) \subseteq \mathcal{O}(ns^2 \log s)$  and the pattern has size  $\mathcal{O}(msd^2 \log s) \subseteq \mathcal{O}(ms^3 \log s)$ .* ■

*Remark.* At a first glance it seems that this reduction can directly be transferred to the pattern matching case without using the blow-up since only a substring of  $t$  has to be matched. But actually this is not possible.

Indeed the encoding of the formula can also be done for pattern matching. But our construction inherently depends on the fact that the whole text has to be matched and for pattern matching we are only interested whether some substring of the text is matched. Although it is possible to show Lemma 5.8 for the matching case, where we replace the  $\mathcal{L}$  by  $\mathcal{M}$ , the other direction would not work since we cannot “force” the pattern to match the whole string. We could show a lower bound for patterns of type  $\circ|+|\circ$ . But this hardness result is also implied by the lower bound for  $\circ|+$  given in Section 4.6.



## 6.1 The Results

**PATTERN MATCHING** In the last chapters we have taken a very fine-grained look at pattern matching and the membership problem. We have shown tighter conditional lower bounds based on FPH for the only quadratic time pattern of depth two,  $\circ\star$ . Likewise, we have proven that one cannot shave off more than a constant number of log-factors from the quadratic time approach for four pattern types of depth three ( $\circ|\circ$ ,  $\circ+\circ$ ,  $\circ+|$ , and  $\circ|+$ ). In [BT09] a general algorithm was shown solving the problem in time  $\mathcal{O}(nm \text{polylog} \log n / \log^{3/2} n)$ . Hence we can see our results as proof that this algorithm is optimal up to a constant number of log-factors. Furthermore, this implies that there is no need to search for specialized algorithms that perform better on these special pattern types. This is again due to the fact that the general algorithm would only be slower by a constant number of log-factors. In contrast, we have shown for the two remaining hard pattern types of depth three ( $| \circ |$  and  $| \circ +$ ) that this is possible by an application of the polynomial method. All of this does of course not apply to the patterns with near linear time algorithms.

Actually one could improve the constant for the number of log-factors even further by defining texts and patterns that are completely tailored for each specific pattern type. Nevertheless, the number of log-factors stays constant and thus these reductions are not considered here.

Based on our results we could state the optimized algorithm for pattern matching and membership: Check the type of the pattern. Decide whether the pattern is easy and then use the fast algorithm for the instance. Since this decision only depends on the outer operation of the pattern, this can be checked quickly. If there is no fast algorithm for this type, the general algorithm for the problem is used.

In Figure 6.1 we give an overview over the pattern types for pattern matching and their running times. While the results in red are new (tighter) results shown in this thesis, the results in blue are refinements based on the result we have shown. All types at the leaves have no super-poly-logarithmic improvement unless FPH is false. Of course except for the patterns that can be simplified by the observations in the introduction or are trivial. By this the complexity of all patterns of depth larger than three can be reduced to one of these patterns. Hence, we obtain a full dichotomy for pattern matching. Except for patterns of type  $| \circ |$  and  $| \circ +$  this classification is also tight up to log-factors.

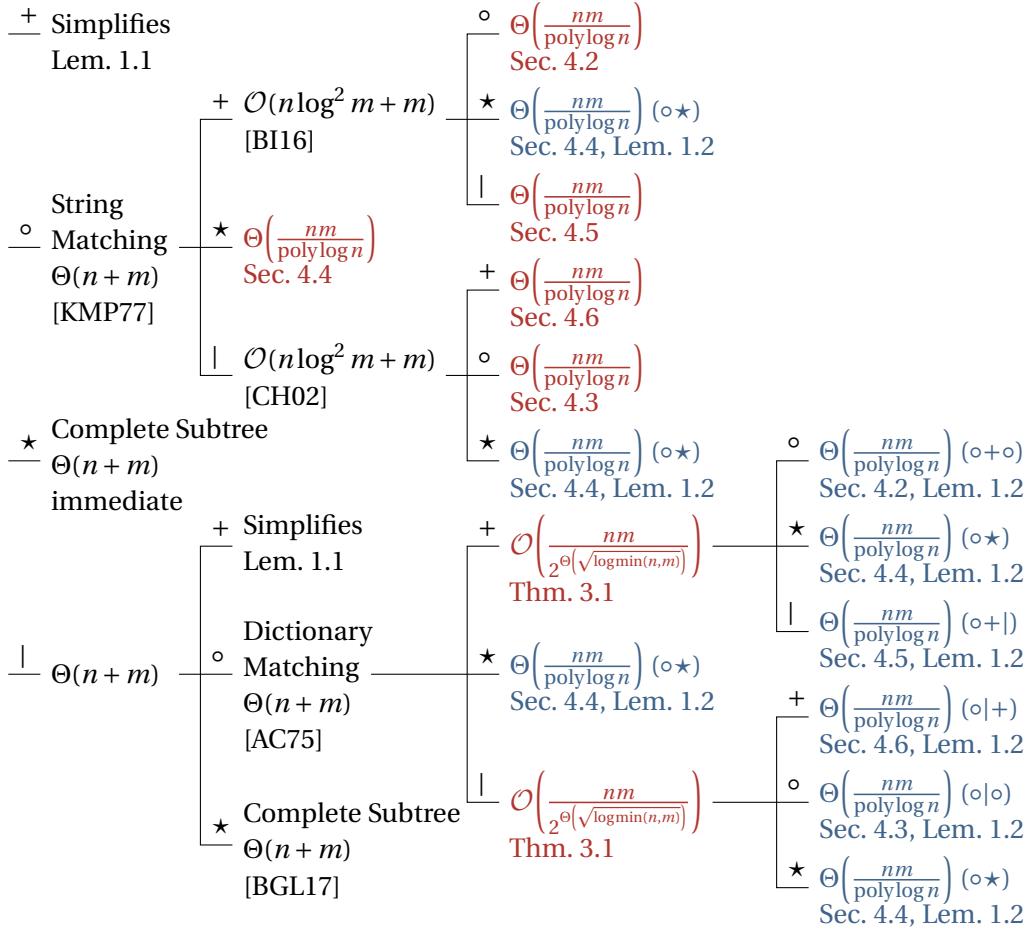


FIGURE 6.1: Running times for pattern matching assuming FPH.

**MEMBERSHIP** For membership we are in quite a similar situation. Based on the reductions for pattern matching we showed tighter conditional lower bounds for many patterns that have no sub-quadratic time algorithm. We also gave a reduction for one pattern of depth four which is a generalization of the word break problem. The results are illustrated in the Figures 6.3, 6.4, and 6.2. One can easily check that for almost all hard pattern types occurring in the figures the current algorithm as mentioned in [BT09] is optimal. The only exceptions are patterns of type  $+|\circ+$  and  $+|\circ|$ . We address this in the next section.

As for pattern matching, this rules out faster algorithms for these hard pattern types. Moreover, the general algorithm is also applicable for *non-homogenous* patterns of *unbounded* depth while all lower bounds are for *homogeneous* patterns of *bounded* depth. But since the upper and lower bounds match up to log-factors, the general algorithm is well suited also for these hard patterns and can be used instead.

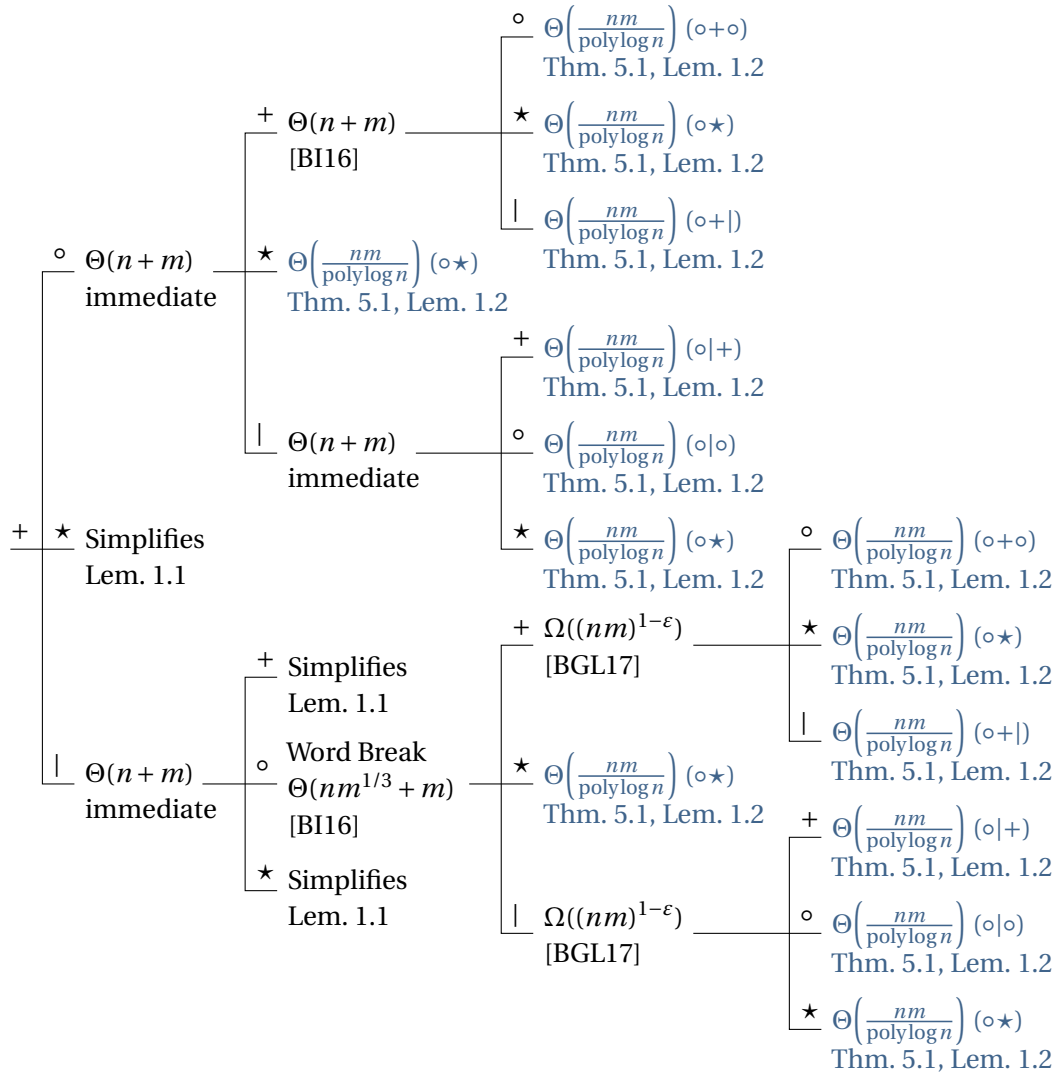


FIGURE 6.2: Running times for membership with types starting with + assuming FPH.

## 6.2 Open Problems and Further Questions

**TIGHT DICHOTOMY FOR MEMBERSHIP** As mentioned above it remains an open question whether one can show a tighter lower bound for membership with patterns of type  $+|\circ+$  and  $+|\circ|$ . A quite interesting observation is that both types combine two intermediate problems: The first three operators, i.e.  $+|\circ$ , represent the word break problem which has a  $\Theta(nm^{1/3} + m)$  time algorithm for membership. This is also the reason why we actually have to consider these patterns. The last three operators represent the pattern types for which we showed faster algorithm in Chapter 3. Even though these patterns are easy for membership, they are hard for pattern matching but still allow super-polylogarithmic improvements. Nevertheless, the resulting patterns of depth four have no

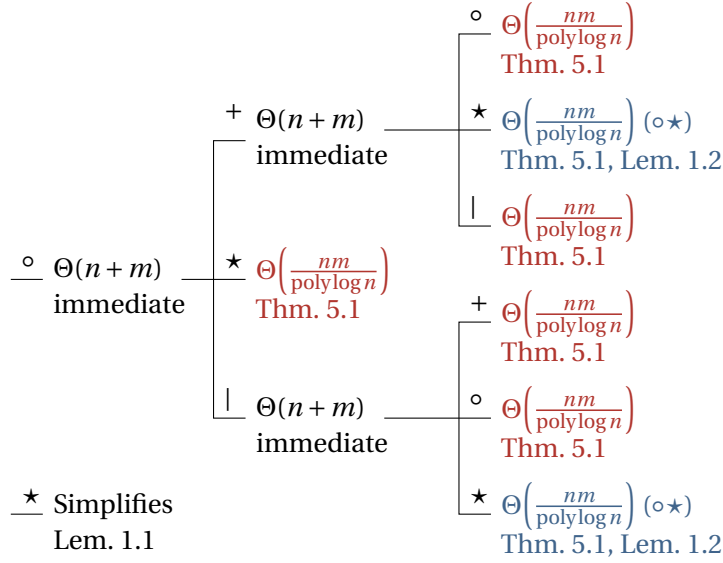


FIGURE 6.3: Complexity for membership with types starting with  $\circ$  and  $\star$  assuming FPH.

strongly sub-quadratic time algorithms as shown in [BGL17], unless SETH is false. But since no sub-type rules out super-poly-logarithmic improvements we can still hope for faster algorithms. One possible way could be a modification of the word break algorithm using the polynomial method for  $|\circ|$  and  $|\circ+$  with small sub-patterns while the large problems are processed sequentially before putting everything together.

Conversely it seems challenging to show a lower bound based on FORMULA-PAIR for these pattern types. Recall that for such a reduction we have to encode one set of half-assignments by the pattern. For the reductions in Chapter 4 we did this by concatenating all assignments (with corresponding gadget) such that the text and the pattern were aligned in an appropriate way if there was a satisfying assignment. For the pattern type  $|\circ+$  of depth four it was even easier to do this because we could use the outer alternative as outer OR. But for the two remaining pattern types the outer operation is a Kleene Plus and thus we have to use a different approach. It should feel intuitive that the alternative following the Plus is used to encode the outer OR when doing the reduction. But then we have to ensure that words corresponding to different assignments are not used simultaneously. All of these observations lead to the conjecture that membership with patterns of type  $|\circ+$  and  $|\circ|$  can be solved in time  $\Theta(nm/2^{\Theta(\sqrt{\log \min(n,m)})})$ .

**CHANGING THE SIZE OF THE ALPHABET** All our reductions for the lower bounds used a constant sized alphabet. This implies that the problems are also hard for non-constant sized alphabets. But none of the alphabets we defined was binary. Thus, it remains open to check whether the same bounds can be achieved by only using the symbols 0 and 1 or whether larger alphabets are needed.

While in this situation we wanted to reduce the size of the alphabet  $\mathfrak{s}$ , we also can consider a larger alphabet for the algorithms we gave. The final algorithm depends on the fact that the size of the alphabet is small because we split the pattern into large





## 6. CONCLUSION

---

**ROBUSTNESS OF HOMOGENEOUS PATTERNS** We know from [B116] that  $\circ+$ -pattern matching can be solved in near linear time. But the similar  $\circ\star$ -pattern matching requires quadratic time and even super-poly-logarithmic factors cannot be shaved off. This is somewhat surprising since pattern of type  $\circ+$  are a special case of patterns of type  $\circ\star$ . Thus, it is an interesting question how robust our definition of homogenous pattern is with regard to changes of the operations. That is, how many Kleene Plus can we change to a Kleene Star such that pattern matching is still solvable in near linear or at least strongly sub-quadratic time. Likewise one could think of similar changes for the other pattern types. But this has to be addressed in some further work.

## A

---

## Homogeneous Patterns of Unbounded Depth

In Chapter 4 we have seen several reductions from FORMULA-PAIR to pattern matching with *homogeneous* patterns of *bounded* depth. As already mentioned, the first lower bound for pattern matching ruling our log-factor improvements in [AB18] used *non-homogeneous* patterns of *unbounded*. In this appendix we show some small modifications of the result in Appendix E of the full version of [AB18] that transform the pattern into a *homogeneous* pattern of *unbounded* depth. Hence, we use the same notation as in this paper in the following.

The encoding of the INPUT and AND gate stays the same since their encoding does not increase the depth of the pattern nor change its type. Instead we change the encoding of the OR gate and the outer OR.

**OR GATES** We first transform  $(0|1)^*$  into the homogeneous expression  $(0^*1^*)^*$ . One can easily prove that both patterns describe the same language. Based on this we change the definition of the texts and the patterns such that they are of type  $\circ\star\circ\star\dots$ .

$$\begin{aligned} t_g &:= \$t_1\#(h')t_2\$ \\ p_g &:= \$(p_1\#(h')(0^*1^*\$^*)^*)((0^*1^*\$^*)^*\#(h')p_2)^*\$ \end{aligned}$$

For the correctness the same arguments as in the original proof hold. Furthermore, we notice that by introducing the fresh symbol  $\$$  we force the pattern to match some text. Otherwise we could choose the empty text as the matched substring. But by the  $\$$  at least one of the sub-patterns has to match. We are sure that exactly one sub-pattern is matched because the  $\#(h')$  appears only once in the text and the two  $\$$  in the beginning and the end have to be matched. Since the text contains the symbol  $\$$ , we have to be able to match it and add it therefore to the alternative we transformed into a homogeneous pattern as mentioned in the beginning.

**OUTER OR** For the outer OR, we use a similar approach as for the OR.

$$\begin{aligned} t &:= \#(H)\$t(a^{(1)})\#(H)\$\dots\#(H)\$t(a^{(n)})\#(H) \\ p &:= \#(H)(\$p(b^{(1)})\$)^*(\$p(b^{(2)})\$)^*\dots(\$p(b^{(n)})\$)^*\#(H) \end{aligned}$$

We first notice, that the trivial string  $\varepsilon$  cannot be matched because the  $\#(H)$  at the beginning and the end of the pattern have to match some  $\#(H)$  in the text. Thus, at least

one of the  $p(b^{(i)})$  has be match text since the  $\#(H)$  are not consecutive. Due to the new symbols, the  $p(b^{(i)})$  can only match some  $t(a^{(j)})$ . Furthermore, at most one of the  $p(b^{(i)})$  can be matched since there are just two  $\$$  between two  $\#(H)$ . The same argument shows that each repetition is taken at most once. Otherwise the second repetition would behave as a new  $p(b^{(i)})$ .

FINAL REMARKS One can easily check that the pattern actually has type  $\circ \star \circ \star \dots$ . This is because consecutive concatenations can be collapsed into one concatenation until the next type is a  $\star$ . Thus the last unproven claim of Theorem 4.1 follows. ■



---

## Bibliography

- [AB18] Amir Abboud and Karl Bringmann. “Tighter Connections Between Formula-SAT and Shaving Logs”. In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. LIPIcs. Full version: <http://arxiv.org/abs/1804.08978>. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 8:1–8:18. ISBN: 978-3-95977-076-7. DOI: 10.4230/LIPIcs.ICALP.2018.8.
- [Abb+16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. “Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made”. In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. Ed. by Daniel Wichs and Yishay Mansour. ACM, 2016, pp. 375–388. ISBN: 978-1-4503-4132-5. DOI: 10.1145/2897518.2897653.
- [AWY15] Amir Abboud, Richard Ryan Williams, and Huacheng Yu. “More Applications of the Polynomial Method to Algorithm Design”. In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*. Ed. by Piotr Indyk. SIAM, 2015, pp. 218–230. ISBN: 978-1-61197-374-7. DOI: 10.1137/1.9781611973730.17.
- [AW14] Amir Abboud and Virginia Vassilevska Williams. “Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems”. In: *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 2014, pp. 434–443. ISBN: 978-1-4799-6517-5. DOI: 10.1109/FOCS.2014.53.
- [AC75] Alfred V. Aho and Margaret J. Corasick. “Efficient String Matching: An Aid to Bibliographic Search”. In: *Commun. ACM* 18.6 (1975), pp. 333–340. DOI: 10.1145/360825.360855.
- [Alo+92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. “Simple Construction of Almost  $k$ -wise Independent Random Variables”. In: *Random Struct. Algorithms* 3.3 (1992), pp. 289–304. DOI: 10.1002/rsa.3240030308.
- [AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4.

- [BI16] Arturs Backurs and Piotr Indyk. “Which Regular Expression Patterns Are Hard to Match?” In: *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*. Ed. by Irit Dinur. Full version: <http://arxiv.org/abs/1511.07070>. IEEE Computer Society, 2016, pp. 457–466. ISBN: 978-1-5090-3933-3. DOI: 10.1109/FOCS.2016.56.
- [BT94] Richard Beigel and Jun Tarui. “On ACC”. In: *Computational Complexity 4* (1994), pp. 350–366. DOI: 10.1007/BF01263423.
- [BT09] Philip Bille and Mikkel Thorup. “Faster Regular Expression Matching”. In: *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*. Ed. by Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas. Vol. 5555. Lecture Notes in Computer Science. Springer, 2009, pp. 171–182. ISBN: 978-3-642-02926-4. DOI: 10.1007/978-3-642-02927-1\_16.
- [BB94] Maria Luisa Bonet and Samuel R. Buss. “Size-Depth Tradeoffs for Boolean Formulae”. In: *Inf. Process. Lett.* 49.3 (1994), pp. 151–155. DOI: 10.1016/0020-0190(94)90093-0.
- [Bri14] Karl Bringmann. “Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails”. In: *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 2014, pp. 661–670. ISBN: 978-1-4799-6517-5. DOI: 10.1109/FOCS.2014.76.
- [BGL17] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. “A Dichotomy for Regular Expression Membership Testing”. In: *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*. Ed. by Chris Umans. Full version: <http://arxiv.org/abs/1611.00918>. IEEE Computer Society, 2017, pp. 307–318. ISBN: 978-1-5386-3464-6. DOI: 10.1109/FOCS.2017.36.
- [BK15] Karl Bringmann and Marvin Künnemann. “Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping”. In: *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. Ed. by Venkatesan Guruswami. IEEE Computer Society, 2015, pp. 79–97. ISBN: 978-1-4673-8191-8. DOI: 10.1109/FOCS.2015.15.
- [BK] Karl Bringmann and Marvin Künnemann. *Fine-Grained Complexity Theory – Lecture Notes – Summer 2019*. Last checked: January 20th, 2020. URL: <https://www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/summer19/fine-complexity/>.

- [CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. “The Complexity of Satisfiability of Small Depth Circuits”. In: *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*. Ed. by Jianer Chen and Fedor V. Fomin. Vol. 5917. Lecture Notes in Computer Science. Springer, 2009, pp. 75–85. ISBN: 978-3-642-11268-3. DOI: 10.1007/978-3-642-11269-0\_6.
- [CW16] Timothy M. Chan and Ryan Williams. “Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky”. In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. Ed. by Robert Krauthgamer. SIAM, 2016, pp. 1246–1255. ISBN: 978-1-61197-433-1. DOI: 10.1137/1.9781611974331.ch87.
- [CH02] Richard Cole and Ramesh Hariharan. “Verifying candidate matches in sparse and wildcard matching”. In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. Ed. by John H. Reif. ACM, 2002, pp. 592–601. ISBN: 1-58113-495-9. DOI: 10.1145/509907.509992.
- [Cop82] Don Coppersmith. “Rapid Multiplication of Rectangular Matrices”. In: *SIAM J. Comput.* 11.3 (1982), pp. 467–471. DOI: 10.1137/0211037.
- [Cyg+12] Marek Cygan, Holger Dell, Daniel Lokshantov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. “On Problems as Hard as CNF-SAT”. In: *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*. IEEE Computer Society, 2012, pp. 74–84. ISBN: 978-1-4673-1663-7. DOI: 10.1109/CCC.2012.36.
- [Gal14] François Le Gall. “Powers of tensors and fast matrix multiplication”. In: *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*. Ed. by Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó. ACM, 2014, pp. 296–303. ISBN: 978-1-4503-2501-1. DOI: 10.1145/2608628.2608664.
- [IP01] Russell Impagliazzo and Ramamohan Paturi. “On the Complexity of k-SAT”. In: *J. Comput. Syst. Sci.* 62.2 (2001), pp. 367–375. DOI: 10.1006/jcss.2000.1727.
- [JMR07] Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. “Monitoring Regular Expressions on Out-of-Order Streams”. In: *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*. Ed. by Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis. IEEE Computer Society, 2007, pp. 1315–1319. ISBN: 1-4244-0802-4. DOI: 10.1109/ICDE.2007.369001.
- [Kin+12] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. “Proton: multitouch gestures as regular expressions”. In: *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*.

- Ed. by Joseph A. Konstan, Ed H. Chi, and Kristina Höök. ACM, 2012, pp. 2885–2894. ISBN: 978-1-4503-1015-4. DOI: 10.1145/2207676.2208694.
- [Kle56] Stephen Cole Kleene. “Representation of Events in Nerve Nets and Finite Automata”. In: *Automata Studies. (AM-34)*. Ed. by C. E. v. Shannon and J. McCarthy. Princeton University Press, 1956, pp. 3–42. DOI: 10.1515/9781400882618-002.
- [KMP77] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. “Fast Pattern Matching in Strings”. In: *SIAM J. Comput.* 6.2 (1977), pp. 323–350. DOI: 10.1137/0206024.
- [Lan92] David Landsman. “RNP-1, an RNA-binding motif is conserved in the DNA-binding cold shock domain”. In: *Nucleic Acids Research* 20.11 (June 1992), pp. 2861–2864. ISSN: 0305-1048. DOI: 10.1093/nar/20.11.2861. eprint: <http://oup.prod.sis.lan/nar/article-pdf/20/11/2861/7059965/20-11-2861.pdf>.
- [LM01] Quanzhong Li and Bongki Moon. “Indexing and Querying XML Data for Regular Path Expressions”. In: *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*. Ed. by Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass. Morgan Kaufmann, 2001, pp. 361–370. ISBN: 1-55860-804-4.
- [Mur01] Makoto Murata. “Extended Path Expressions for XML”. In: *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*. Ed. by Peter Buneman. ACM, 2001. ISBN: 1-58113-361-8. DOI: 10.1145/375551.375569.
- [Mye92] Eugene W. Myers. “A Four Russians Algorithm for Regular Expression Pattern Matching”. In: *J. ACM* 39.2 (1992), pp. 430–448. DOI: 10.1145/128749.128755.
- [NN93] Joseph Naor and Moni Naor. “Small-Bias Probability Spaces: Efficient Constructions and Applications”. In: *SIAM J. Comput.* 22.4 (1993), pp. 838–856. DOI: 10.1137/0222053.
- [NR03] Gonzalo Navarro and Mathieu Raffinot. “Fast and Simple Character Classes and Bounded Gaps Pattern Matching, with Applications to Protein Searching”. In: *Journal of Computational Biology* 10.6 (2003). PMID: 14980017, pp. 903–923. DOI: 10.1089/106652703322756140. eprint: <https://doi.org/10.1089/106652703322756140>.
- [Raz02] Ran Raz. “On the complexity of matrix product”. In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. Ed. by John H. Reif. ACM, 2002, pp. 144–151. ISBN: 1-58113-495-9. DOI: 10.1145/509907.509932.



- [Raz87] A. A. Razborov. “Lower bounds on the size of bounded depth circuits over a complete basis with logical addition”. In: *Mathematical notes of the Academy of Sciences of the USSR* 41.4 (Apr. 1987), pp. 333–338. ISSN: 1573-8876. DOI: 10.1007/BF01137685.
- [Smo87] Roman Smolensky. “Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity”. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by Alfred V. Aho. ACM, 1987, pp. 77–82. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28404.
- [Tho68] Ken Thompson. “Regular Expression Search Algorithm”. In: *Commun. ACM* 11.6 (1968), pp. 419–422. DOI: 10.1145/363347.363387.
- [Wil14a] Richard Ryan Williams. “The Polynomial Method in Circuit Complexity Applied to Algorithm Design (Invited Talk)”. In: *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*. Ed. by Venkatesh Raman and S. P. Suresh. Vol. 29. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014, pp. 47–60. ISBN: 978-3-939897-77-4. DOI: 10.4230/LIPIcs.FSTTCS.2014.47.
- [Wil14b] Ryan Williams. “Faster all-pairs shortest paths via circuit complexity”. In: *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. Ed. by David B. Shmoys. ACM, 2014, pp. 664–673. ISBN: 978-1-4503-2710-7. DOI: 10.1145/2591796.2591811.
- [Yu+06] Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz. “Fast and memory-efficient regular expression matching for deep packet inspection”. In: *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems, ANCS 2006, San Jose, California, USA, December 3-5, 2006*. Ed. by Laxmi N. Bhuyan, Michel Dubois, and Will Eather-ton. ACM, 2006, pp. 93–102. ISBN: 1-59593-580-0. DOI: 10.1145/1185347.1185360.



## Abbreviations and Notations

FPH	FORMULA-PAIR HYPOTHESIS
FSH	FORMULA-SAT HYPOTHESIS
SETH	STRONG EXPONENTIAL TIME HYPOTHESIS
$\mathbb{B}$	The boolean values true and false, identified with 0 and 1 if necessary.
$d(F)$	Depth of the formula $F$ , cf. Definition 2.20.
$\varepsilon$	Empty string consisting of zero symbols.
$F_g$	The sub-formula of $F$ rooted at gate $g$ .
$\mathbb{F}_2$	Finite field with 2 elements.
$\mathcal{L}(q)$	Language accepted by the pattern $q$ , cf. Definition 2.1
$\mathcal{M}(q)$	The language of words with a substring matched by $q$ , cf. Definition 2.3.
$\mathfrak{RE}_\Sigma$	The class of regular expressions over alphabet $\Sigma$ , cf. Definition 2.1.
$\mathfrak{RE}_\Sigma(T)$	Class of homogeneous pattern of type $T$ over $\Sigma$ , cf. Definition 2.6.
$\mathfrak{s}$	Size of the alphabet $\Sigma$
$s$	Size of a formula $F$
$\Sigma$	Alphabet containing $\mathfrak{s}$ symbols.
$\llbracket A \rrbracket$	Returns true/1 if the expression $A$ is correct and false/0 otherwise.
$\langle i \rangle$	Binary encoding of integer $i$ where the number of bits is clear from the context.
$\langle i \rangle_j$	The $j$ th bit of the binary encoding of $i$ with $j = 0$ as the least significant bit.
$[n]$	The set of the integers $\{1, 2, \dots, n\}$ .
$[n, m]$	The set of the integers $\{n, n + 1, \dots, m - 1, m\}$ .
$t^*$	The starred version of a text as defined in Definition 4.8.
$\bar{t}$	The barred version of a text as defined in Definition 4.14.
$\overleftarrow{t}^\beta$	The blow-up of a text with text string $\beta$ as in Definition 5.7.