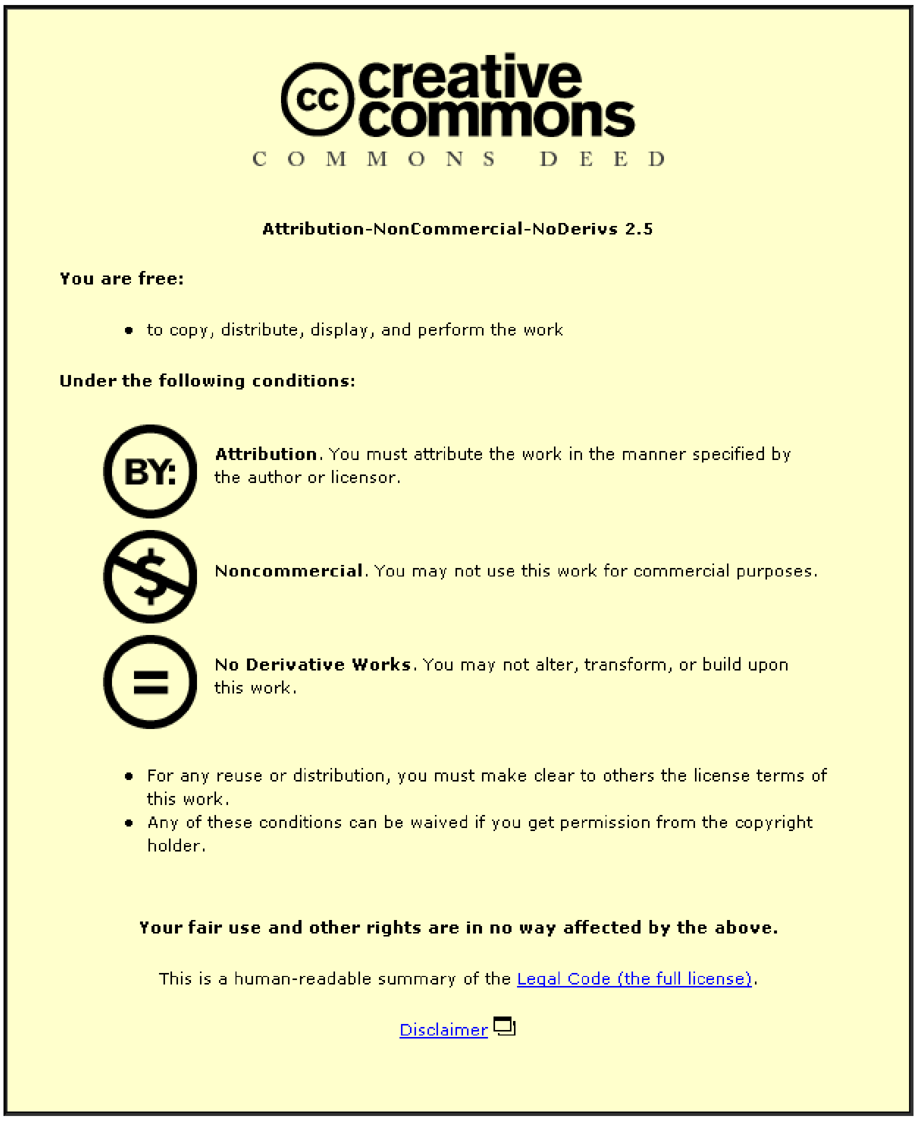


Loughborough University

Thesis Access Form

Copy No…………...…………………….**Location**…………………………………………………….……………...…

Author…………...……………………………………………….…………………………………………..……

Title………..

Status of access OPEN / RESTRICTED / CONFIDENTIAL

Moratorium Period:…………………………………years, ending…………../…………200………………………

Conditions of access approved by (CAPITALS):…………………………………………………………………

Supervisor (Signature)………………………………………...…………………………………...

School of…………………………………………………………………...…………………………………

Author's Declaration: *I agree the following conditions:*

Open access work shall be made available (in the University and externally) and reproduced as necessary at the discretion of the University Librarian or Dean of School. It may also be digitised by the British Library and made freely available on the Internet to registered users of the EThOS service subject to the EThOS supply agreements.

The statement itself shall apply to ***ALL*** *copies including electronic copies:*

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Restricted/confidential work: All access and any photocopying shall be strictly subject to written permission from the University Dean of School and any external sponsor, if any.

Author's signature……………………………………….Date…………………………………...…………...……...

users declaration: for signature during any Moratorium period (Not Open work): ***I undertake to uphold the above conditions:***			
Date	Name (CAPITALS)	Signature	Address

4.1e

On the Membership Problem for Pattern Languages and Related Topics

by

Markus L. Schmid

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

30 June 2012

Research Student Office, Academic Registry
Loughborough University, Leicestershire, LE11 3TU, UK
Switchboard: +44 (0)1509 263171 Fax: +44 (0)1509 223938

Certificate of Originality

This is to certify that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgements or in footnotes, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a higher degree.

....

Markus L. Schmid

30 June 2012

Abstract

In this thesis, we investigate the complexity of the membership problem for pattern languages. A *pattern* is a string over the alphabet $\Sigma \cup X$, where $X := \{x_1, x_2, x_3, \dots\}$ is a countable set of *variables* and Σ is a finite alphabet containing *terminals* (e. g., $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$). Every pattern, e. g., $\beta := x_1\, x_2\, \mathtt{ab}\, x_2\, \mathtt{b}\, x_1\, \mathtt{c}\, x_2$, describes a *pattern language*, i. e., the set of all words that can be obtained by uniformly substituting the variables in the pattern by arbitrary strings over Σ. Hence, $u :=$ `cacaaabaabcaccaa` is a word of the pattern language of β, since substituting `cac` for x_1 and `aa` for x_2 yields u. On the other hand, there is no way to obtain the word $u' :=$ `bbbababbacaaba` by substituting the occurrences of x_1 and x_2 in β by words over Σ.

The problem to decide for a given pattern α and a given word w whether or not w is in the pattern language of α is called the *membership problem for pattern languages*. Consequently, (β, u) is a positive instance and (β, u') is a negative instance of the membership problem for pattern languages. For the unrestricted case, i. e., for arbitrary patterns and words, the membership problem is NP-complete. In this thesis, we identify classes of patterns for which the membership problem can be solved efficiently.

Our first main result in this regard is that the *variable distance*, i. e., the maximum number of different variables that separate two consecutive occurrences of the same variable, substantially contributes to the complexity of the membership problem for pattern languages. More precisely, for every class of patterns with a bounded variable distance the membership problem can be solved efficiently. The second main result is that the same holds for every class of patterns with a bounded *scope coincidence degree*, where the scope coincidence degree is the maximum number of intervals that cover a common position in the pattern, where each interval is given by the leftmost and rightmost occurrence of a variable in the pattern.

The proof of our first main result is based on automata theory. More precisely, we introduce a new automata model that is used as an algorithmic framework in order to show that the membership problem for pattern languages can be solved in time that is exponential only in the variable distance of the corresponding pattern.

We then take a closer look at this automata model and subject it to a sound theoretical analysis. The second main result is obtained in a completely different way. We encode patterns and words as relational structures and we then reduce the membership problem for pattern languages to the homomorphism problem of relational structures, which allows us to exploit the concept of the treewidth. This approach turns out be successful, and we show that it has potential to identify further classes of patterns with a polynomial time membership problem.

Furthermore, we take a closer look at two aspects of pattern languages that are indirectly related to the membership problem. Firstly, we investigate the phenomenon that patterns can describe regular or context-free languages in an unexpected way, which implies that their membership problem can be solved efficiently. In this regard, we present several sufficient conditions and necessary conditions for the regularity and context-freeness of pattern languages. Secondly, we compare pattern languages with languages given by so-called *extended regular expressions with backreferences* (REGEX). The membership problem for REGEX languages is very important in practice and since REGEX are similar to pattern languages, it might be possible to improve algorithms for the membership problem for REGEX languages by investigating their relationship to patterns. In this regard, we investigate how patterns can be extended in order to describe large classes of REGEX languages.

Acknowledgements

I am indebted most to Daniel Reidenbach. As my supervisor, he has been supporting me throughout my research with great intensity and I profited from his experience and his skills in many ways. At the same time, he always respected my own individual ideas and gave me sufficient space to gain independence. Daniel's high demands he places on himself has been a great inspiration and encouragement for me.

Furthermore, I would like to thank those who have enabled and encouraged me to take up my PhD research and to write this thesis. In this regard, I am very grateful to Dominik Freydenberger for supervising my Bachelor Thesis and to Georg Schnitger, Detlef Wotschke and Nicole Schweikardt for their excellent and motivating lectures about various aspects of theoretical computer science.

Last but not least, I wish to express my gratitude to my family and friends for their continuous support in many different ways.

Contents

Chapter 1

Introduction

1.1 On Identifying a Pattern in a Word

According to the Oxford Dictionary [57], the term *pattern* describes "something serving as a model". In this regard, the mould that is used by a smith in the manufacturing process of a knife as well as the templates used by a tailor in the production of textiles both are patterns. Hence, a pattern can have the function of a blueprint that is used in order to produce copies of one and the same object with a preferably high grade of precision.

On the other hand, patterns are used in a much less precise way in order to describe sets of different objects that show similarities only in few details. For example, the terms *ballad*, *sonnet* or *limerick* describe patterns for poems, and the terms *opera*, *sonata* or *fugue* are used in order to describe different forms of musical pieces. The expressiveness of these kinds of patterns is usually large and they constitute essential and powerful means of communication. However, their advantages come at a high cost: while a layman, provided with the appropriate measuring instruments, is well able to identify those knifes that do not meet a certain standard or those textiles that contain manufacturing errors, it takes some expert knowledge to tell whether or not a musical piece is a fugue.

Consequently, for patterns the following question is crucial: given an arbitrary object, does this object satisfy a certain pattern? Furthermore, the more expressive a pattern is, the more complex it seems it is to answer this question. But how exactly can we quantify the expressive power of a pattern? To this end, we interpret a pattern as a formal descriptor of a set of objects, i. e., the set of all objects that satisfy the pattern. In this regard, a mould for knifes or a template for cloth are descriptors of rather boring sets of objects, i. e., the set of identical knifes of some kind and the set of identically shaped pieces of cloth, respectively. The term fugue, on the other hand, is a descriptor of an interesting and complex class of musical pieces.

The above developed definitions of patterns on the one hand and objects that satisfy patterns on the other hand are rather informal. In this thesis, we wish to study, in a formal sense, the complexity of computing the answer to instances of the question whether or not a given object satisfies a certain pattern. Hence, a mathematically sound formalisation is required.

To this end, we apply the most fundamental mathematical objects that are commonly used in order to encode and represent information and that are processable by computers: sequences of symbols. For the sake of concreteness, we define an alphabet of *terminal symbols* $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and we call every sequence of terminal symbols a *word*. For example, $w_1 := \mathtt{abacbab}$, $w_2 := \mathtt{aaaacaaab}$ and $w_3 := \mathtt{accbcccbb}$ all are words. If a word contains *variables* (possibly in addition to terminals) from the set $X := \{x_1, x_2, x_3, \ldots\}$, then we call it a *pattern*[1]. Thus, $\alpha := \mathtt{a}\,x_1\,\mathtt{c}\,x_1\,\mathtt{b}$ is an example of a pattern. Intuitively, the variables in a pattern are placeholders for other words. Hence, in accordance with our initial view of patterns as production tools, the patterns defined here are blueprints for words, where the positions labeled by a variable are placeholders for other components, which are taken from the set of words over Σ. More precisely, α describes all words that can be obtained by substituting both occurrences of x_1 by just some word. Naturally, we have to substitute both occurrences of variable x_1 by the same word, since otherwise it does not make sense to use different variables in the first place. We conclude that α describes the *pattern language* $L(\alpha) := \{\mathtt{a}\,u\,\mathtt{c}\,u\,\mathtt{b} \mid u \in \Sigma^+\}$, where Σ^+ denotes the set of all (non-empty) words over the alphabet Σ and we say that all words $w \in L(\alpha)$ *satisfy* the pattern α. The following illustration demonstrates that in fact the above defined words w_1, w_2 and w_3 all satisfy α:

$$\begin{aligned} w_1 &= \mathtt{a}\,\overset{x_1}{\overbrace{\mathtt{ba}}}\,\mathtt{c}\,\overset{x_1}{\overbrace{\mathtt{ba}}}\,\mathtt{b}, \\ w_2 &= \mathtt{a}\,\overset{x_1}{\overbrace{\mathtt{aaa}}}\,\mathtt{c}\,\overset{x_1}{\overbrace{\mathtt{aaa}}}\,\mathtt{b}, \\ w_3 &= \mathtt{a}\,\overset{x_1}{\overbrace{\mathtt{ccb}}}\,\mathtt{c}\,\overset{x_1}{\overbrace{\mathtt{ccb}}}\,\mathtt{b}\,. \end{aligned}$$

Deciding for a given word w on whether or not it is a member of $L(\alpha)$ is not very difficult. It can be done by checking whether or not all of the following conditions are satisfied (in the following, $|w|$ denotes the length of a word w):

[1]It goes without saying that the above definition is just one possible way to formalise a pattern in a word. There exist a large number of quite different mathematically sound formalisations of patterns in mathematical objects, each of which caters for specific aspects and is tailored to certain mathematical problems. Our concept of patterns is due to Dana Angluin and we shall explain the role of these patterns and their importance for theoretical computer science in more detail in Section 2.2.2.

- $|w|$ is odd.
- The first symbol of w is $\mathtt{a}$, the last symbol of w is $\mathtt{b}$ and the symbol in the middle is $\mathtt{c}$.
- The factor between the first and middle symbol and the factor between the middle and last symbol are equal.

However, this is an *ad hoc* procedure, which cannot be generalised to more involved patterns as, e. g., $\beta := x_1\,\mathtt{a}\,x_2\,x_3\,x_1\,x_3\,\mathtt{b}\,x_2\,x_1$. Intuitively, in order to check whether or not a word w satisfies β, we have to check whether or not there exist words $u_1, u_2, u_3 \in \Sigma^+$, such that w can be written as (or, more formally, can be *factorised* into) $w = u_1\,\mathtt{a}\,u_2\,u_3\,u_1\,u_3\,\mathtt{b}\,u_2\,u_1$. This, as it seems, is only possible by trying out a large number of different factorisations of w. In fact, if the complexity classes P and NP do not coincide, then it can be shown that for the class of all possible such patterns, this question cannot be answered in a way that is essentially better than testing all possible factorisations. More precisely, this problem, which we call the *membership problem for pattern languages*, is NP-complete.

In order to develop a gut feeling for the complexity of the membership problem, we now take a more general point of view. Let α be a pattern of form $y_1\,y_2 \cdots y_n$, $y_i \in X$, $1 \leq i \leq n$, i. e., it does not contain any terminal symbols which makes the following considerations easier (and, at the same time, the loss of generality caused is negligible). Solving the membership problem for α and a given word w is the task of finding a factorisation $w = u_1\,u_2 \cdots u_n$, $u_i \in \Sigma^+$, $1 \leq i \leq n$, such that, for every i, j, $1 \leq i < j \leq n$, if $y_i = y_j$, then $u_i = u_j$ follows. More intuitively speaking, we have to decompose w in n factors in such a way that all the factors corresponding to the same variable in α (which comprises exactly n occurrences of variables) are equal. Trying out all possible factorisations is a correct, but time-consuming way to solve this task.

On second thoughts, we observe that the number of factorisations that we have to investigate does not depend on the length of the pattern, but rather on the number of different variables. If α contains only 2 variables, all that needs to be done is to consider all possibilities to allocate a factor to each variable, since this already implies a full factorisation of w. For example, in order to check whether or not a word w satisfies $x_1\,x_2\,x_2\,x_1\,x_2\,x_1\,x_2$, we enumerate all tuples (u_1, u_2), where u_1 and u_2 are words with $|u_1| \leq |w|$ and $|u_2| \leq |w|$, and then check whether or not $w = u_1\,u_2\,u_2\,u_1\,u_2\,u_1\,u_2$. We can further boost this algorithm by enumerating only those tuples (u_1, u_2) that satisfy $3\,|u_1| + 4\,|u_2| = |w|$, since if this is not satisfied, then $|w| \neq |u_1\,u_2\,u_2\,u_1\,u_2\,u_1\,u_2|$. In the worst case, however, we have to enumerate a number of factorisations that is exponential in the number of different variables in the pattern.

We are now provided with a first understanding of the complexity of the membership problem and we have also learned that the number of different variables is the crucial parameter that contributes to its complexity. However, all these insights are fairly basic and somewhat unsatisfying, since we can easily come up with specific example patterns for which it is quite easy to solve the membership problem even though the number of variables is large. There even seem to be simple classes of patterns with an unbounded number of variables, for which the membership problem can be solved efficiently. For example, for every n, the pattern $\alpha_n := x_1\, x_2 \cdots x_n\, x_1\, x_2 \cdots x_n$ describes the language $\{u\, u \mid u \in \Sigma^+, |u| \geq n\}$ and, thus, it is very easy to solve the membership problem for these kinds of patterns. This suggests that parameters of patterns other than their number of different variables exist that also substantially contribute to the complexity of the membership problem. If we can identify such parameters, then it is likely that by restricting them, we obtain classes of patterns for which the membership problem can be solved efficiently. Consequently, the goal of this thesis can be paraphrased in the following way:

> We want to find large classes of patterns for which the membership problem can be solved in time that is polynomial in the length of the input word and the number of variables.

1.2 Organisation of this Thesis

The present thesis is structured in the following way. In Chapter 2, we present most technical concepts and definitions that are used throughout the thesis. However, some more specialised definitions are provided in the individual chapters where they are required. In addition to basic definitions of formal language theory, Chapter 2 also contains a detailed definition of patterns and pattern languages, which have already been outlined in Section 1.1. Furthermore, a discussion of the most prominent known results regarding pattern languages and their importance in theoretical computer science and discrete mathematics is provided. Since the membership problem for pattern languages is the main topic of this thesis, we spend some more time on it, thoroughly explaining its different aspects. Chapter 2 is concluded with the definition of two fundamental technical concepts that play an important role in theoretical computer science and, in the scope of this thesis, serve as central tools in order to obtain our main results: finite automata and relational structures in conjunction with the treewidth.

The main part of the thesis is formed by Chapters 3, 4 and 5. In Chapter 3, the first approach to the problem of identifying classes of pattern languages with a polynomial time membership problem is presented. Since this approach is based on finite automata, we start with a comparison of automata and pattern languages. After that, so-called nondeterministically bounded modulo counter automata (NBMCA) are defined. This special purpose automaton model serves as the central tool in order to prove the main result of Chapter 3, i. e., we identify an infinite hierarchy of classes of patterns, for which the membership problem can be solved efficiently.

Before we present our second main approach, the next chapter, Chapter 4, is an interlude that focuses on the model of NBMCA and provides a theoretical sound analysis of this class of automata. We investigate questions about their expressive power and decidability properties and we also take a closer look at stateless variants of NBMCA with and without restricted nondeterminism. The chapter is concluded by a study of a variant of multi-head automata that is only marginally related to pattern languages and mainly motivated by the special kind how nondeterminism is used by NBMCA.

In Chapter 5, we identify classes of pattern languages with a polynomial time membership problem in quite a different way as it is done in Chapter 3. More precisely, we encode patterns and words as relational structures and thereby reduce the membership problem for pattern languages to the homomorphism problem for relational structures. It turns out that this is a very convenient and powerful way to treat the membership problem.

In Chapters 6 and 7, we study two aspects of patterns that again are of indirect importance for the membership problem. Firstly, in Chapter 6, we investigate the phenomenon that patterns can describe regular or context-free languages in an unexpected way. This implies that there are classes of patterns with an efficient membership problem, simply because these patterns describe regular or context-free languages. Secondly, in Chapter 7 we investigate possibilities to combine pattern languages with regular expressions in order to describe subclasses of so-called extended regular expressions with backreferences (REGEX). These REGEX are a widely applied tool to define formal languages and they are a generalisation of pattern languages. It is likely, although not obvious, that insights about the membership problem for pattern languages can be transferred to REGEX.

Finally, in Chapter 8, we summarise the results presented in this thesis and address some open questions and ideas for future research.

1.3 Original Contribution

All major results of the present thesis have been previously published by the author in conference proceedings or journals. The following list is intended to help to map the results presented in the subsequent chapters to the corresponding articles: Section 3.3 of Chapter 3 has been originally presented in [69] and Section 3.4 in [68] and its journal version [67]. The interlude, Chapter 4, contains work published in [70] (Section 4.1) and [71] (Section 4.2). In [72], most of the results of Chapter 5 are contained and Chapters 6 and 7 have been presented in [73] and [77], respectively.

Chapter 2

Preliminaries

In this chapter we introduce basic and general definitions. More specific technical concepts that are especially tailored to the results of this thesis shall be given in the individual chapters.

We assume the reader to be familiar with the standard mathematical concepts and notations and, furthermore, the elementary definitions in formal language and automata theory (cf. Salomaa [76], Hopcroft et al. [32]) and complexity theory (cf. Papadimitriou [58]).

We start this chapter by giving a brief overview of some standard definitions concerning words and languages and then we formally define the concept of pattern languages. After that, we discuss the known results regarding pattern languages and explain their importance in theoretical computer science and discrete mathematics, focusing on the membership problem. We conclude this chapter by defining some classes of automata as well as relational structures and the concept of the treewidth.

2.1 Words and Languages

Let $\mathbb{N}$ denote the set of all positive integers and let $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. The symbols $\subseteq$ and $\subset$ refer to subset and proper subset relation, respectively. For any set A, $\mathcal{P}(A)$ denotes the powerset of A.

A (*finite*) *alphabet* is a (finite) set of symbols. For an arbitrary alphabet A, a *string* or *word* (*over* A) is a finite sequence of symbols from A, and ε stands for the *empty word*. The notation A^+ denotes the set of all nonempty strings over A, and $A^* := A^+ \cup \{\varepsilon\}$. For the *concatenation* of two words u, v we write $u \cdot v$ or simply $u\,v$, and u^k denotes the k-fold concatenation of u, i. e., $u^k := u_1\, u_2 \cdots u_k$, where $u_i = u$, $1 \leq i \leq k$. We say that a word $v \in A^*$ is a *factor* of a word $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1\, v\, u_2$. If $u_1 = \varepsilon$ (or $u_2 = \varepsilon$), then v is a *prefix* of w (or a *suffix*, respectively). The notation $|K|$ stands for the size of a set K or the length of a string K. The term $\mathrm{alph}(w)$ denotes the set of all symbols occurring in w and, for each $a \in \mathrm{alph}(w)$, $|w|_a$ refers to the number of occurrences of a in w. A word w' is a *permutation* of a word w if and only if $\mathrm{alph}(w) = \mathrm{alph}(w')$ and, for every $a \in \mathrm{alph}(w)$, $|w|_a = |w'|_a$. If we wish to refer to the symbol at a certain position in a word $w = a_1\, a_2 \cdots a_n$, $a_i \in A$, $1 \leq i \leq n$, over some alphabet A, then we use $w[i] := a_i$, $1 \leq i \leq n$, and if the length of a string is unknown, then we denote its last symbol by $w[-] := w[|w|]$. Furthermore, for each j, j', $1 \leq j < j' \leq |w|$, let $w[j, j'] := a_j\, a_{j+1} \cdots a_{j'}$ and $w[j, -] := w[j, |w|]$. In case that $j > |w|$, we define $w[j, -] = \varepsilon$. A word w' is the *reversal* of a word w if and only if, for every i, $1 \leq i \leq |w|$, $w[i] = w'[|w| - i + 1]$. Furthermore, for every word w, its reversal is denoted by w^R.

For any alphabets A, B, a *morphism* is a function $h : A^* \to B^*$ that satisfies $h(vw) = h(v)h(w)$ for all $v, w \in A^*$; h is said to be *nonerasing* if and only if, for every $a \in A$, $h(a) \neq \varepsilon$.

For any alphabet A, a language (over A) is a set $L \subseteq A^*$ of words over A. For arbitrary languages L_1, L_2 we define $L_1^+ := \{u_1\, u_2 \cdots u_n \mid u_i \in L_1, 1 \leq i \leq n, n \in \mathbb{N}\}$, $L_1^* := L_1^+ \cup \{\varepsilon\}$ and $L_1 \cdot L_2 := \{u \cdot v \mid u \in L_1, v \in L_2\}$.

Let u and v be words over the alphabet A. The *shuffle operation*, denoted by $⧢$, is a binary operation on words, defined by

$$u ⧢ v := \{x_1\, y_1\, x_2\, y_2 \cdots x_n\, y_n \mid n \in \mathbb{N}, x_i, y_i \in (A \cup \{\varepsilon\}), 1 \leq i \leq n, \\ u = x_1\, x_2 \cdots x_n, v = y_1\, y_2 \cdots y_n\}\,.$$

We extend the definition of the shuffle operation to the case of more than two words in the following inductive way. Let u, v and w be words over the alphabet A. Then $(u ⧢ v) ⧢ w := \bigcup_{w' \in u ⧢ v} w ⧢ w'$. We note that since the shuffle operation is obviously associative, we can drop the brackets, i. e., for arbitrary words $w_1, w_2, \ldots, w_k \in A^*$, the term $w_1 ⧢ w_2 ⧢ \ldots ⧢ w_k$ is well defined. Furthermore, we call $\Gamma := w_1 ⧢ w_2 ⧢ \ldots ⧢ w_k$ the *shuffle* of $w_1, \ldots, w_k$ and each word $w \in \Gamma$ is a *shuffle word* of $w_1, \ldots, w_k$. For example, $\mathtt{bcaabac} \in \mathtt{abc} ⧢ \mathtt{ba} ⧢ \mathtt{ca}$.

The classes of *regular* languages and *context-free* languages are denoted by REG and CF, respectively. We use regular expressions as they are commonly defined (see, e. g., Yu [89]) and for any regular expression r, $L(r)$ denotes the language described by r.

In order to prove some of the technical claims in this thesis, the following two versions of the well-known pumping lemma for regular languages as stated in Yu [89] will be used.

Lemma 2.1. *Let $L \subseteq \Sigma^*$ be a regular language. Then there is a constant n,* *depending on L, such that for every $w \in L$ with $|w| \geq n$ there exist $x, y, z \in \Sigma^*$ such that $w = xyz$ and*

1. *$|xy| \leq n$,*
2. *$|y| \geq 1$,*
3. *$xy^kz \in L$ for every $k \in \mathbb{N}_0$.*

Lemma 2.2. *Let $L \subseteq \Sigma^*$ be a regular language. Then there is a constant n, depending on L, such that for all $u, v, w \in \Sigma^*$, if $|w| \geq n$, then there exist $x, y, z \in \Sigma^*$, $y \neq \varepsilon$, such that $w = xyz$ and, for every $k \in \mathbb{N}_0$, $uxy^kzv \in L$ if and only if $uwv \in L$.*

2.2 Patterns and Pattern Languages

We shall now formally define pattern languages. Let Σ be a (finite) alphabet of so-called *terminal symbols* and X an infinite set of *variables* with $\Sigma \cap X = \emptyset$. We normally assume $X := \{x_1, x_2, x_3, \ldots\}$. A *pattern* is a nonempty string over $\Sigma \cup X$, a *terminal-free pattern* is a nonempty string over X and a *word* is a string over Σ. For any pattern α, we refer to the set of variables in α as $\text{var}(\alpha)$ and, for any variable $x \in \text{var}(\alpha)$, $|\alpha|_x$ denotes the number of occurrences of x in α. A morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ is called a *substitution* if $h(a) = a$ for every $a \in \Sigma$.

Let $\alpha \in (\Sigma \cup X)^*$ be a pattern. The *erasing pattern (or* E*-pattern) language* of α is defined by

$$L_{\text{E},\Sigma}(\alpha) := \{h(\alpha) \mid h : (\Sigma \cup X)^* \to \Sigma^* \text{ is a substitution}\},$$

and the *non-erasing pattern (or* NE*-pattern) language* of α is defined by

$$L_{\text{NE},\Sigma}(\alpha) := \{h(\alpha) \mid h : (\Sigma \cup X)^* \to \Sigma^* \text{ is a nonerasing substitution}\}.$$

If the difference between the E and NE case is negligible, then we use the notation $L_{\text{Z},\Sigma}(\alpha)$, $\text{Z} \in \{\text{E}, \text{NE}\}$, in order to denote pattern languages.

2.2.1 Properties and Parameters of Patterns

A convenient way to prove results about pattern languages, e. g., about the complexity of their membership problem, is to show that properties of pattern languages can be reduced to structural properties of their corresponding patterns. Since a pattern is a special kind of word, its structure, in contrast to the structure of a pattern language, can be easily analysed. In this section, we present properties and parameters of patterns that are crucial for the results in this thesis. However, before we do so, we wish to define some terminology.

In the following, when we speak of a *property of a pattern*, then we refer to some predicate that is either satisfied or not satisfied by any pattern. A *parameter of a pattern*, on the other hand, is a function that maps a pattern to an integer. More formally, a property of a pattern is a mapping $(\Sigma \cup X)^* \to \{\mathtt{true}, \mathtt{false}\}$ and a parameter of a pattern is a mapping $(\Sigma \cup X)^* \to \mathbb{N}_0$. For example, every pattern either satisfies or does not satisfy the property of not containing any terminal symbol. On the other hand, the *number of variables* in a pattern α, i. e., the number $|\operatorname{var}(\alpha)|$, is probably the first parameter of patterns that comes to mind. Although this parameter is important for a variety of questions and it obviously contributes to the complexity of pattern languages, it is somewhat trivial in the sense that it ignores the order of the variables, which, as shall be shown, is often crucial.

Next, we present two important properties of patterns that have been introduced by Shinohara [80]. A pattern is *non-cross* if and only if between any two occurrences of the same variable x no other variable different from x occurs, e. g., the pattern $\mathtt{a}x_1\mathtt{ba}x_1x_2\mathtt{a}x_2x_2x_3x_3\mathtt{b}x_4$ is non-cross, whereas $x_1\mathtt{b}x_1x_2\mathtt{ba}x_3x_3x_4x_4\mathtt{bc}x_2$ is not. A pattern is *regular* if and only if every variable has only one occurrence in the pattern, e. g., $\mathtt{a}x_1\mathtt{ba}x_2\mathtt{c}x_3\mathtt{bca}x_4\mathtt{a}x_5\mathtt{bb}$ is a regular pattern.

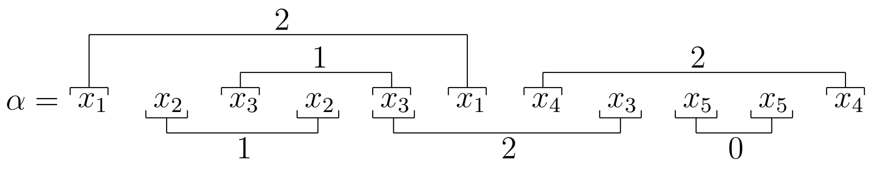
In this thesis, we are mainly interested in properties or parameters of patterns that yield classes of patterns the structure of which is restricted, but the number of variables is not. We can note that the class of non-cross patterns as well as the class of regular patterns constitute examples of such classes.

A colourful analogy of the non-cross property is that in a non-cross pattern every different variable occupies a territory and no variable is allowed to cross the territory that is occupied by another variable. While it is rather clear how a pattern looks like if we do not allow any crossing of variables in this sense, it is not straightforward to formally quantify a certain amount of crossing, i. e., to generalise the non-cross property to a parameter of patterns. For example, it is not clear whether this is achieved best by counting the number of territories that are crossed by at least one other variable or rather by taking the maximum number of different variables that are crossing the same territory.

In the following, we present two parameters that can be interpreted as generalisations of the non-cross property. The first parameter is the *variable distance*. Informally speaking, the variable distance is the maximum number of different variables separating any two consecutive occurrences of a variable:

Definition 2.3. The *variable distance* of a pattern α (or $\mathrm{vd}(\alpha)$ for short) is the smallest number $k \geq 0$ such that, for every $x \in \mathrm{var}(\alpha)$, every factorisation $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$ with $\beta, \gamma, \delta \in (\Sigma \cup X)^*$ and $|\gamma|_x = 0$ satisfies $|\,\mathrm{var}(\gamma)| \leq k$.

Obviously, $\mathrm{vd}(\alpha) \leq |\,\mathrm{var}(\alpha)| - 1$ for all patterns α. To illustrate the concept of the variable distance, we consider $\alpha := x_1\, x_2\, x_3\, x_2\, x_3\, x_1\, x_4\, x_3\, x_5\, x_5\, x_4$. In the following figure, for every two successive occurrences of any variable in α, the number of different variables occurring between these occurrences is shown:

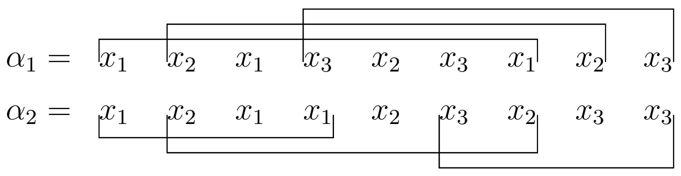

2
1
2
$\alpha = x_1\; x_2\; x_3\; x_2\; x_3\; x_1\; x_4\; x_3\; x_5\; x_5\; x_4$
1
2
0

Hence, it can be easily seen that $\mathrm{vd}(\alpha) = 2$.

The second parameter is the *scope coincidence degree*, which, intuitively, is the maximum number of intervals that cover a common position in the pattern, where each interval is given by the leftmost and rightmost occurrence of a variable in the pattern:

Definition 2.4. Let α be a pattern. For every $y \in \mathrm{var}(\alpha)$, the *scope of y in α* is defined by $\mathrm{sc}_\alpha(y) := \{i, i+1, \dots, j\}$, where i is the leftmost and j the rightmost position of y in α. The scopes of $y_1, y_2, \dots, y_k \in \mathrm{var}(\alpha)$ *coincide in α* if and only if $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. The *scope coincidence degree* of α ($\mathrm{scd}(\alpha)$) is the maximum number of variables in α such that their scopes coincide.

As an example, we consider the patterns $\alpha_1 := x_1\, x_2\, x_1\, x_3\, x_2\, x_3\, x_1\, x_2\, x_3$ and $\alpha_2 := x_1\, x_2\, x_1\, x_1\, x_2\, x_3\, x_2\, x_3\, x_3$. In the following figure the scopes of the variables in α_1 and α_2 are highlighted:


$\alpha_1 = x_1\; x_2\; x_1\; x_3\; x_2\; x_3\; x_1\; x_2\; x_3$
$\alpha_2 = x_1\; x_2\; x_1\; x_1\; x_2\; x_3\; x_2\; x_3\; x_3$

Hence, $\mathrm{scd}(\alpha_1) = 3$ and $\mathrm{scd}(\alpha_2) = 2$.

The variable distance as well as the scope coincidence degree can be computed in time that is polynomial in the length of the pattern. This aspect is discussed in a bit more detail by Proposition 3.13 on page 46 and by Proposition 3.37 on page 80, respectively. The following lemma relates the variable distance and the scope coincidence degree.

Lemma 2.5. *Let α be a pattern. Then* $\mathrm{scd}(\alpha) \leq \mathrm{vd}(\alpha) + 1$.

Proof. Let $\mathrm{scd}(\alpha) = k$, which, by definition, implies that, for k distinct variables $y_1, y_2, \dots, y_k \in \mathrm{var}(\alpha)$, $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. Furthermore, this implies that there exists a p, $1 \leq p \leq k$, such that α can be factorised into $\alpha = \beta \cdot y_p \cdot \gamma$ with $(\{y_1, y_2, \ldots, y_k\}/\{y_p\}) \subseteq (\text{var}(\beta) \cap \text{var}(\gamma))$. Now let q, $1 \leq q \leq k$, $q \neq p$, be such that β can be factorised into $\beta = \beta' \cdot y_q \cdot \beta''$ with $(\{y_1, y_2, \ldots, y_k\}/\{y_p, y_q\}) \subseteq \text{var}(\beta'')$ and $y_q \notin \beta''$. Since there is an occurrence of y_q in γ, γ can be factorised into $\gamma = \gamma' \cdot y_q \cdot \gamma''$ with $|\gamma'|_{y_q} = 0$. Hence, α contains the factor $y_q \cdot \beta'' \cdot y_p \cdot \gamma' \cdot y_q$, where $|\beta'' \cdot y_p \cdot \gamma'|_{y_q} = 0$ and $(\{y_1, y_2, \ldots, y_k\}/\{y_q\}) \subseteq \text{var}(\beta'' \cdot y_p \cdot \gamma')$, which implies $\text{vd}(\alpha) \geq k - 1 = \text{scd}(\alpha) - 1$. □

On the other hand, the variable distance cannot be bounded in terms of the scope coincidence degree since, for example, all patterns that are of the form $x_1 x_2 x_3 \ldots x_k x_{k+1} x_1$, $k \in \mathbb{N}$, have a variable distance of k, but a constant scope coincidence degree of 2.

The above defined parameters of the variable distance and the scope coincidence degree shall play a central role for our two main approaches to the membership problem for pattern languages presented in Chapters 3 and 5. It is shown in Section 3.3 that the membership problem with respect to any class of patterns with a bounded variable distance can be solved efficiently. In Section 3.4, the scope coincidence degree is applied for the first time, but to general words instead of patterns. More precisely, we present an algorithm for an optimisation problem on words, where the scope coincidence degree is the corresponding optimisation parameter. The scope coincidence degree for patterns, as this parameter is defined above, is crucial in context of Chapter 5, where it is shown that for the membership problem for pattern languages this parameter plays a similar role as the variable distance, i. e., bounding the scope coincidence degree of patterns allows us to solve their membership problem efficiently.

2.2.2 Known Results about Pattern Languages

The concept of NE-pattern languages was introduced by Angluin [6] in 1980 and soon afterwards complemented by Shinohara [79], who included the empty word as an admissible substitution word, leading to the definition of E-pattern languages. As revealed by numerous studies, the small difference between the definitions of NE- and E-pattern languages entails substantial differences between some of the properties of the resulting (classes of) formal languages (see, e. g., Mateescu and Salomaa [51] for a survey).

The original motivation of pattern languages (cf. Angluin [6]) is derived from *inductive inference*, i. e., the task of inferring a pattern from any given sequence of all words in its pattern language, for which numerous results can be found in the literature (see, e. g., Angluin [6], Shinohara [79], Lange and Wiehagen [48], Rossmanith and Zeugmann [75], Reidenbach [64, 66] and, for a survey, Ng and Shinohara [55]). On the other hand, due to their simple definition, pattern languages have connections to many areas of theoretical computer science and discrete mathematics, such as (un-)avoidable patterns (cf. Jiang et al. [41]), word equations (cf. Mateescu and Salomaa [50]), the ambiguity of morphisms (cf. Freydenberger et al. [22]), equality sets (cf. Harju and Karhumäki [28]) and extended regular expressions (cf. Câmpeanu et al. [11]).

It can be easily verified that the class of pattern languages is incomparable with the class of regular languages as well as with the class of context-free languages. For example, the well-known copy language, which, over an alphabet of size at least 2, is not context-free, can be described by the simple pattern $x_1 \, x_1$. On the other hand, pattern languages are always context-sensitive, since they can be accepted by linear bounded nondeterministic Turing machines (in fact, as shall be mentioned in Sections 3.1 and 4.2.3, pattern languages can be accepted in deterministic logarithmic space). In the following, we shall give an overview of the current state of research regarding the most prominent decision problems of patterns, namely their membership problem, inclusion problem and equivalence problem. Since the focus of this thesis is on the membership problem of pattern languages, we shall first give a brief overview of the inclusion problem and equivalence problem and then discuss the membership problem in more detail in Section 2.2.2.1.

For every $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$, the inclusion problem for Z-pattern languages is the problem to decide, for two given patterns α and β, whether or not $L_{\mathrm{Z},\Sigma}(\alpha) \subseteq L_{\mathrm{Z},\Sigma}(\beta)$. Similarly, the equivalence problem for Z-pattern languages is the problem to decide whether or not $L_{\mathrm{Z},\Sigma}(\alpha) = L_{\mathrm{Z},\Sigma}(\beta)$. In [6], Angluin has shown that the equivalence problem for NE-pattern languages (with respect to any terminal alphabet) is decidable. This is due to the fact that two patterns describe the same NE-pattern language if and only if they are equal up to a renaming of variables, i. e., one pattern can be obtained from the other by uniformly renaming the variables. On the other hand, the question of whether or not the inclusion problem for pattern languages is decidable had been open for a long time until it was answered in the negative by Jiang et al. [42] for both the E and NE case. More precisely, Jiang et al. [42] show that there is no effective procedure deciding the inclusion problem for the class of all pattern languages over *all* alphabets. It has later been shown that, for any fixed alphabet of size at least 2, the inclusion problem is also undecidable for the class of pattern languages defined over this fixed alphabet (see Freydenberger and Reidenbach [21]). Moreover, the inclusion problem remains undecidable for patterns with a bounded number of variables (see Bremer and Freydenberger [10]). Hence, for NE-pattern languages we have the remarkable situation that the inclusion problem is undecidable, whereas the equivalence problem is trivially decidable. This is a property that, according to Jiang et al. [42] and the references therein, is shared only by few classes of formal languages including languages accepted by finite deterministic multi-tape automata, simple languages and deterministic context-free languages (in fact, for deterministic context-free languages the decidability status of the equivalence problem is still open).

The equivalence problem for E-pattern languages is much more difficult. In fact, its decidability status is still open and it is subject to ongoing research (see, e. g., Jiang et al. [42], Ohlebusch and Ukkonen [56], Reidenbach [65], Freydenberger and Reidenbach [21]).

2.2.2.1 The Membership Problem

In this section, we discuss the membership problem for pattern languages in detail and outline the most important known results about its complexity. First of all, we present a formal definition:

Definition 2.6. Let $Z \in \{\mathrm{E}, \mathrm{NE}\}$, let Σ be an alphabet and let $C \subseteq (\Sigma \cup X)^*$ be a class of patterns. The *membership problem for* Z*-pattern languages with respect to C and* Σ is defined in the following way:

$$\text{Z-PATMem}_\Sigma(C) := \{(\alpha, w) \mid \alpha \in C, w \in \Sigma^*, w \in L_{\mathrm{Z},\Sigma}(\alpha)\}\,.$$

The *membership problem for* Z*-pattern languages with respect to* Σ is defined by Z-PATMem$_\Sigma$:= Z-PATMem$_\Sigma((\Sigma \cup X)^*)$.

If the dependency of the alphabet is negligible or understood from the context, then we ignore it, i. e., we write Z-PATMem(C) and Z-PATMem.

In the following, let Σ be an alphabet, let $C \subseteq (\Sigma \cup X)^*$ be a class of patterns and let $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. We say that Z-PATMem$_\Sigma(C)$ is *decidable* if and only if there exists a total computable function which, for every $\alpha \in (\Sigma \cup X)^*$ and for every $w \in \Sigma^*$, decides on whether or not $(\alpha, w) \in$ Z-PATMem$_\Sigma(C)$. It can be easily verified that the membership problem for pattern languages is decidable and we already outlined in Chapter 1 how this can be done. However, regarding our definition above, we can observe a particularity. If C is not decidable, then Z-PATMem$_\Sigma(C)$ is also not decidable, since $\alpha \in C$ if and only if $(\alpha, h(\alpha)) \in$ Z-PATMem$_\Sigma(C)$, where h is some substitution. In this regard, the dependency of the class C in our definition of the membership problem seems to be problematic. However, since we are interested in complexity issues we shall see that this definition is suitable for our purposes.

We say that the membership problem for Z-pattern languages with respect to some class of patterns C and Σ (or Z-PATMem$_\Sigma(C)$) can be *solved in time*

$O(f(n, m))$ for some function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, if and only if there exists an algorithm that, for any $\alpha \in C$ and $w \in \Sigma^*$, decides correctly on whether or not $(\alpha, w) \in$ Z-PATMem$_\Sigma(C)$ in time $O(f(|\alpha|, |w|))$.

It should be emphasised that the above notation particularly implies that the algorithm solving Z-PATMem(C) can assume the input pattern α to be from the class C, which implies that the complexity of deciding the class C is not taken into account. This definition is convenient for us, since it allows us to treat these two aspects, i. e., identifying classes of patterns C on the one hand and solving the membership problem with respect to such a class on the other hand, separately. Naturally, for all our concrete complexity results about the membership problem with respect to some class C, we shall also explicitly mention the complexity of deciding whether or not an arbitrary pattern is in the class C. However, for the classes of patterns considered in this thesis, this problem can always be solved in polynomial time, which implies that a polynomial time solvability of the membership problem with respect to such classes C is not affected by the complexity of deciding C.

As already mentioned in Chapter 1, the membership problem for pattern languages is NP-complete, which, for NE-pattern languages, has been shown by Angluin [5] and independently by Ehrenfeucht and Rozenberg [16][1] and, for E-pattern languages, by Jiang et al. [41].

Theorem 2.7 (Angluin [5], Ehrenfeucht and Rozenberg [16], Jiang et al. [41])**.** *Let* $Z \in \{E, NE\}$ *and* $\Sigma := \{\mathtt{a}, \mathtt{b}\}$. *The membership problem for* Z*-pattern languages with respect to* Σ *is NP-complete.*

We note that Theorem 2.7 implies that PATMem$_\Sigma$ is NP-complete for any alphabet Σ with $|\Sigma| \geq 2$. If, on the other hand, $|\Sigma| = 1$, then PATMem$_\Sigma$ can be solved in polynomial time. This is due to the fact that, for every pattern $\alpha \in (\Sigma \cup X)^*$, with $|\Sigma| = 1$, the lengths of the words of the unary language $L_{Z,\Sigma}(\alpha)$ are solely characterised by the number of occurrences of the variables in α, i. e., their order is insignificant. For example, the pattern $\alpha := \mathtt{a}\, x_1\, x_2\, \mathtt{a}\, x_2\, x_1\, x_2$ (as well as the pattern $\mathtt{a}\,\mathtt{a}\, x_1\, x_1\, x_2\, x_2\, x_2$ and, in general, every permutation of α) describes the NE-pattern language $L_{NE,\Sigma}(\alpha) = \{\mathtt{a}^n \mid \exists k_1, k_2 \in \mathbb{N} : n = 2k_1 + 3k_2 + 2\}$. It is straightforward to construct a finite automaton that accepts exactly $L_{NE,\Sigma}(\alpha)$ and this holds for all unary pattern languages. Consequently, unary pattern languages

[1]Ehrenfeucht and Rozenberg show that it is an NP-complete problem to decide, for two given words u and v, where u is a word over an infinite alphabet and v is a word over a binary alphabet, whether or not there exists a nonerasing morphism that maps u to v. Hence, in terms of pattern languages, Ehrenfeucht and Rozenberg show a result that is slightly stronger than the NP-completeness of NE-pattern languages, i. e., they show the NP-completeness of terminal-free NE-pattern languages.

are always regular languages and, thus, the membership problem can be solved in polynomial time.[2]

For alphabets of size at least 2, a brute-force algorithm can solve the membership problem in time that is exponential only in the number of variables in the pattern (for a detailed complexity analysis see Ibarra et al. [35]). Intuitively speaking, such a brute-force algorithm simply enumerates all morphisms and checks whether or not they map the input pattern to the input word. This directly implies that if we restrict the number of variables to a constant, then the membership problem can be solved in polynomial time. More precisely, if for a class C of patterns there exists a constant $k \in \mathbb{N}$, such that, for every $\alpha \in C$, $|\operatorname{var}(\alpha)| \leq k$, then Z-PATMem($C$) can be solved in polynomial time. This result is neither surprising nor particularly informative. Thus, for the membership problem, classes of patterns with a bounded number of variables are usually not very interesting[3]. This is in contrast to the learnability of pattern languages or their inclusion problem since, as explained above, in these areas important results are concerned with patterns with a bounded number of variables. In the remainder of this section, we are mainly concerned with classes of patterns with an *unbounded* number of variables for which it is known that the membership problem can be solved efficiently.

We first consider the full class of patterns, for which the membership problem can only be solved in polynomial time if the input words are restricted in some way. This follows directly from the NP-completeness of the problem and the assumption that P does not equal NP. A result of this kind is provided by Geilke and Zilles [26], who show that the membership problem can be solved in polynomial time for the whole class of patterns provided that the length of the input words is bounded by a constant. More formally, Geilke and Zilles show that, for every constant k, the class Z-PATMem$_{\Sigma,k} := \{(\alpha, w) \mid \alpha \in (\Sigma, \cup X)^*, w \in \Sigma^*, |w| \leq k, w \in L_{\text{Z},\Sigma}(\alpha)\}$ can be decided in polynomial time. Strictly speaking, this is not a result about the membership problem for pattern languages, since the restriction of the input words restricts the model of pattern languages, i. e., an algorithm deciding Z-PATMem$_{\Sigma,k}$

[2]The computational problem of solving an equation with non-negative integer coefficients as, e. g., $n = 2k_1 + 3k_2 + 2$, is often called Money-Changing-Problem and it is NP-complete, which seems to contradict the polynomial time solvability of the membership problem with respect to unary pattern languages. The reason for this is that the Money Changing Problem is *weakly* NP-complete, i. e., it is only NP-complete since its input merely consists of numbers in binary representation, which means that the input length for the Money Changing Problem is exponentially smaller than for the membership problem for pattern languages, where we have to regard the lengths of the input strings as input length of the problem.

[3]However, it is worth mentioning that a more refined complexity analysis of the membership problem with respect to patterns with a bounded number of variables, provided by Stephan et al. [83], shows that the membership problem for pattern languages is fixed parameter intractable if parameterised by the number of variables.

solves the membership problem for any pattern α, but only for a finite subclass of $L_{\mathrm{Z},\Sigma}(\alpha)$, which, in the strict sense of the definition, is not a pattern language. For these reasons, we shall concentrate on classes C of patterns that are such that the membership problem for any $\alpha \in C$ and *any* $w \in \Sigma^*$ can be solved in polynomial time, i. e., Z-PATMem$_\Sigma(C)$ is solvable in polynomial time. Hence, our main research question can be stated in the following way:

Question 2.8. How can patterns be restricted in order to obtain classes C such that Z-PATMem$_\Sigma(C)$ is solvable in polynomial time.

In reference to Question 2.8, it is very unlikely that a restriction of the terminals in the patterns is helpful, since the membership problem remains NP-complete even for terminal-free patterns (cf. Ehrenfeucht and Rozenberg [16], Schneider [78]). More precisely, for every Σ with $|\Sigma| \geq 2$ and $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$, Z-PATMem$_\Sigma(X^*)$ is NP-complete.

In the following, let C_{reg} and C_{nc} denote the classes of regular patterns and non-cross patterns respectively. In Shinohara [80], it is shown that for $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$ the membership problem for Z-pattern languages with respect to C_{reg} or C_{nc} can be solved in polynomial time. Furthermore, it can be easily seen that it can be decided in polynomial time whether or not a given pattern is regular or non-cross. For regular patterns the polynomial time solvability of the membership problem follows trivially from the fact that, for any regular pattern α, $L_{\mathrm{Z},\Sigma}(\alpha)$ is a regular language. This is due to the fact that the pattern language of a regular pattern α is the set of all words that contain the terminal segments of α in the same order as they occur in α. For example, in the E case, the pattern $x_1\,\mathtt{ab}\,x_2\,\mathtt{acba}\,x_3\,\mathtt{c}\,x_4\,\mathtt{ac}\,x_5$ describes the set of all words that contain non-overlapping occurrences of the factors `ab`, `acba`, `c` and `ac` in exactly this order. In the NE case, we further require that between these occurrences of the factors at least one symbol occurs. It is straightforward to show how a finite automaton can recognise such languages.

Theorem 2.9 (Shinohara [80])**.** *Let Σ be an alphabet and $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. Then* Z-PATMem$_\Sigma(C_{\mathrm{reg}})$ *is decidable in polynomial time.*

Regarding non-cross patterns the situation is slightly more complicated. It can be shown that for every non-cross pattern α, $L_{\mathrm{Z},\Sigma}(\alpha)$ can be accepted by a nondeterministic two-way 4-head automaton. For such automata, the acceptance problem, i. e., the problem to decide whether or not a given word is accepted by the automaton, is exponential only in the number of input heads.

Theorem 2.10 (Shinohara [80])**.** *Let Σ be an alphabet and $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. Then* Z-PATMem$_\Sigma(C_{\mathrm{nc}})$ *is decidable in polynomial time.*

To the knowledge of the author of the present thesis, the classes C_{reg} and C_{nc} are the only known non-trivial classes of patterns for which the membership problem can be solved in polynomial time in the strict sense that Z-PATMem$_\Sigma(C)$, $C \in \{C_{\mathrm{reg}}, C_{\mathrm{nc}}\}$, is decidable in polynomial time.

2.2.3 Related Concepts

In this section, we mention two areas of computer science for which the membership problem for pattern languages plays a central role, namely extended regular expressions with backreferences and parameterised pattern matching.

2.2.3.1 Extended Regular Expressions with Backreferences

Since their introduction by Kleene in 1956 [45], *regular expressions* have not only constantly challenged researchers in formal language theory, but they also attracted pioneers of applied computer science as, e. g., Thompson [86], who developed one of the first implementations of regular expressions, marking the beginning of a long and successful tradition of their practical application (see Friedl [23] for an overview). In order to suit practical requirements, regular expressions have undergone various modifications and extensions which lead to so-called *extended regular expressions with backreferences* (*REGEX* for short). The introduction of these new features of extended regular expressions has frequently not been guided by theoretically sound analyses and only recent studies have led to a deeper understanding of their properties (see, e. g., Câmpeanu et al. [11]).

The main difference between REGEX and *classical* regular expressions is the concept of backreferences. Intuitively speaking, a backreference points back to an earlier subexpression, meaning that it has to be matched to the same word as the earlier subexpression has been matched to. For example, $r := (_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1 \cdot \mathtt{c} \cdot \backslash 1$ is a REGEX, where $\backslash 1$ is a *backreference* to the *referenced subexpression* in between the parentheses $(_1$ and $)_1$. The language described by r, denoted by $\mathcal{L}(r)$, is the set of all words $w\mathtt{c}w$, $w \in \{\mathtt{a}, \mathtt{b}\}^*$, which is a non-regular language.

In this regard, backreferences are used in a very similar way as the variables in patterns are used, and it is straightforward to see that pattern languages are included in the class of languages that can then be described by REGEX. For example, let $\alpha := x_1\, \mathtt{a}\, x_2\, x_1\, x_1\, \mathtt{ba}\, x_2$ be a pattern and let Σ be some alphabet. The language $L_{\mathrm{E},\Sigma}(\alpha)$ can be described by the REGEX $(_1\ \Sigma^*\ )_1 \mathtt{a} (_2\ \Sigma^*\ )_2 \backslash 1 \backslash 1\, \mathtt{ba} \backslash 2$. This directly implies that all negative decidability results on pattern languages carry over to REGEX languages. Furthermore, the NP-completeness of the membership problem carries over to REGEX languages as well. This is particularly worth mentioning as today's text editors and programming languages (such as Perl, Python, Java, etc.) all provide so-called *REGEX engines* that compute the solution to the membership problem for any language given by a REGEX and an arbitrary string (cf. Friedl [23]). Hence, despite its theoretical intractability, algorithms that perform the match test for REGEX are a practical reality. While pattern languages merely describe a proper subset of REGEX languages, they cover what is computationally hard, i. e., the concept of backreferences. Hence, investigating the membership problem for pattern languages helps to improve algorithms solving the match test for extended regular expressions with backreferences.

In Chapter 7, we introduce extensions of pattern languages and investigate their usefulness in describing REGEX languages. In this way, we gain a better understanding of backreferences in REGEX in comparison to the weaker concept of variables in patterns.

2.2.3.2 Parameterised Pattern Matching

The membership problem for pattern languages can also be considered as a kind of pattern matching task, since we have to decide whether or not a given word satisfies a given pattern. In fact, this pattern matching aspect of pattern languages, independently from Angluin's work, has recently been rediscovered in the pattern matching community in terms of so-called *parameterised pattern matching*, where a text is not searched for all occurrences of a specific factor, but for all occurrences of factors that satisfy a given pattern with parameters (i. e., variables). In the original version of parameterised pattern matching introduced by Baker [7], variables in the pattern can only be substituted by single symbols and, furthermore, the substitution must be injective, i. e., different variables cannot be substituted by the same symbol. Amir et al. [3] generalise this problem by dropping the injectivity condition and Amir and Nor [4] add the possibility of substituting variables by words instead of single symbols and they also allow "don't care" symbols to be used in addition to variables. In 2009, Clifford et al. [14] considered parameterised pattern matching as introduced by Amir and Nor, but without "don't care" symbols, which leads to patterns as introduced by Angluin. In [4], motivations for the membership problem for pattern languages can be found from such diverse areas as software engineering, image searching, DNA analysis, poetry and music analysis, or author validation.

2.3 Two Fundamental Algorithmic Toolkits

The two main approaches to the membership problem for pattern languages presented in this thesis utilise two technical toolkits that are tailored to algorithmic purposes. The first such toolkit are finite automata and the second are relational structures in conjunction with the homomorphism problem for relational structures. In Chapters 3 and 5, it shall be explained in detail how these toolkits serve our purpose and how they are applied in order to achieve the main results. Hence, in this section, we only give the basic definitions regarding automata and relational structures.

2.3.1 Finite Automata

In this section, we summarise some of the basic definitions of automata theory. For those concepts not covered in this section, the reader is referred to Hopcroft et al. [32].

In the whole thesis, for an arbitrary class of automata models, e. g., the set DFA of deterministic finite automata, the expression "a DFA" refers to any automaton from DFA. The class of *deterministic* and *nondeterministic* (*one-way one-head*) *automata* is denoted by DFA and NFA, respectively.

Next, we define *multi-head automata* in a bit more detail (for a comprehensive survey on multi-head automata the reader is referred to Holzer et al. [31] and to the references therein). For every $k \in \mathbb{N}$ let 1DFA(k), 2DFA(k), 1NFA(k) and 2NFA(k) denote the class of *deterministic one-way*, *deterministic two-way*, *nondeterministic one-way* and *nondeterministic two-way automata* with k input heads, respectively. A 1DFA(k), 2DFA(k), 1NFA(k) or 2NFA(k) is given as a tuple $(k, Q, \Sigma, \delta, q_0, F)$ comprising the number of *input heads* $k \geq 1$, a set of *states* Q, the *input alphabet* Σ, the *transition function* δ, an *initial state* $q_0 \in Q$ and a set of *accepting states* $F \subseteq Q$. The transition function is a mapping $Q \times (\Sigma \cup \{¢, \$\})^k \to Q \times D^k$ for deterministic and $Q \times (\Sigma \cup \{¢, \$\})^k \to \mathcal{P}(Q \times D^k)$ for nondeterministic devices, where D, i. e., the set of input head movements, is $\{0, 1\}$ in case of one-way automata and $D = \{-1, 0, 1\}$ for the two-way versions.

Let M be a 2DFA(k) or 2NFA(k). An *input* to M is any string of the form $¢w\$$, where $w \in \Sigma^*$ and the symbols $¢, \$$ (referred to as *left* and *right endmarker*, respectively) are not in Σ. Let $\delta(p, b_1, b_2, \ldots, b_k) \ni (q, m_1, m_2, \ldots, m_k)$. For each i, $1 \leq i \leq k$, we call the element b_i the *input symbol scanned by head* i and m_i the *instruction for head* i and, furthermore, we assume that $b_i = ¢$ implies $m_i \neq -1$ and $b_i = \$$ implies $m_i \neq 1$. A *configuration* of M on some input $¢w\$$ is a tuple containing a state and k positions in $¢w\$$. A configuration $c := (p, h_1, h_2, \ldots, h_k)$ can be changed into a configuration $c' := (q, h'_1, h'_2, \ldots, h'_k)$ (denoted by the relation $c \vdash_{M,w} c'$) if and only if there exists a transition $\delta(p, b_1, b_2, \ldots, b_k) \ni (q, m_1, m_2, \ldots, m_k)$ with $¢w\$[h_i] = b_i$ and $h'_i = h_i + m_i$, $1 \leq i \leq k$. To describe a *computation of* M *(on input* $¢w\$$*)* we use the reflexive and transitive closure of the relation $\vdash_{M,w}$, denoted by $\vdash^*_{M,w}$. The *initial configuration of M (on input ¢w\$)* is the configuration $(q_0, 0, 0, \ldots, 0)$. An *accepting configuration of M (on input ¢w\$)* is any configuration of form $(q_f, h_1, h_2, \ldots, h_k)$, $q_f \in F$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$. M accepts the word w if and only if $\widehat{c}_0 \vdash^*_{M,w} \widehat{c}_f$, where $\widehat{c}_0$ is the initial configuration, and $\widehat{c}_f$ is an accepting configuration.

The definitions of the previous paragraph apply to 1DFA(k) and 1NFA(k) in an analogous way, with the only difference that there is no left endmarker and the instructions for the heads are `0` or `1`.

We now briefly define *counter automata* in a more informal way; all the details are defined in an analogous way as for multi-head automata. For every $k \in \mathbb{N}$ let 1CDFA(k), 2CDFA(k), 1CNFA(k) and 2CNFA(k) denote the class of *deterministic one-way*, *deterministic two-way*, *nondeterministic one-way* and *nondeterministic two-way counter automata* with one input head and k counters. The counters can only store positive values. In each transition, these counters can be incremented, decremented or left unchanged and, furthermore, it can be checked whether or not a certain counter stores value 0. A transition of a counter automaton depends on the state, the currently scanned input symbol and the set of counters currently storing 0. For more details on counter automata see, e. g., Ibarra [36] or Holzer et al. [31].

For an arbitrary automaton M, $L(M)$ denotes the set of all words accepted by M. For an arbitrary class A of automata, let $L(A)$ denote the class of languages defined by automata in A, i. e., $L(A) := \{L(M) \mid M \in A\}$. 2NFA and 2CNFA denote the class of nondeterministic two-way multi-head automata and counter automata, respectively, with any number of input heads and any number of counters, respectively. This notation is analogously used for all other above defined classes of multi-head and counter automata.

We conclude this section with an observation that shall be useful for our applications of multi-head automata:

Observation 2.11. The input heads of a two-way multi-head automaton can be used in order to implement a counter that can store numbers between 0 and the current input length in the following way. We use two input heads, a *left* input head that scans the left endmarker and a *right* one, that scans a position i, i. e., the number stored by the counter. An increment or decrement is performed by moving the right input head a step to the right or to the left, respectively. The left input head is needed to retrieve the value of the counter without loosing it, which is done by moving the right input head to the left until it reaches the left endmarker and simultaneously moving the left endmarker to the right. From now on the roles of the left and right endmarker are changed.

2.3.2 Relational Structures

We now define relational structures, tree decompositions and the concept of the treewidth. For a comprehensive textbook reference about these standard definitions, the reader is referred to Chapters 4, 11 and 13 of Flum and Grohe [19].

A *(relational) vocabulary* τ is a finite set of relation symbols. Every relation symbol $R \in \tau$ has an *arity* $\mathrm{ar}(R) \geq 1$. A *τ-structure* $\mathcal{A}$ (or simply *structure*), comprises a finite set A called the *universe* and, for every $R \in \tau$, an *interpretation* $R^{\mathcal{A}} \subseteq A^{\mathrm{ar}(R)}$. For example, a graph $\mathcal{G} = (V, E)$ can be given as a relational structure $\mathcal{A}_{\mathcal{G}}$ over the relational vocabulary $\{E'\}$, $\mathrm{ar}(E') = 2$, with universe $V' := V$ and the binary relation E' is interpreted as $E'^{\mathcal{A}_{\mathcal{G}}} = E$.

Let $\mathcal{A}$ and $\mathcal{B}$ be structures of the same vocabulary τ with universes A and B, respectively. A *homomorphism* from $\mathcal{A}$ to $\mathcal{B}$ is a mapping $h : A \to B$ such that for all $R \in \tau$ and for all $a_1, a_2, \ldots, a_{\mathrm{ar}(R)} \in A$, $(a_1, a_2, \ldots, a_{\mathrm{ar}(R)}) \in R^{\mathcal{A}}$ implies $(h(a_1), h(a_2), \ldots, h(a_{\mathrm{ar}(R)})) \in R^{\mathcal{B}}$.

Next, we introduce the concepts of *tree decompositions* and *treewidth* of a graph (see Chapter 11 of Flum and Grohe [19]).

Definition 2.12. A *tree decomposition* of a graph $\mathcal{G} := (V, E)$ is a pair $(\mathcal{T}, \{B_t \mid t \in T\})$, where $\mathcal{T} := (T, F)$ is a tree and the B_t, $t \in T$, are subsets of V such that the following is satisfied:

1. For every $v \in V$, the set $\{t \in T \mid v \in B_t\}$ is nonempty and connected in $\mathcal{T}$.

2. For every edge $\{u, v\} \in E$ there is a $t \in T$ such that $\{u, v\} \subseteq B_t$.

The *width* of the tree decomposition $(\mathcal{T}, \{B_t \mid t \in T\})$ is the number $\max\{|B_t| \mid t \in T\} - 1$. The *treewidth of* $\mathcal{G}$ *(denoted by* $\mathrm{tw}(\mathcal{G})$) is the minimum of the widths of the tree decompositions of $\mathcal{G}$.

A tree decomposition, the underlying tree of which is a path, is also called a *path decomposition* and the *pathwidth* of a graph $\mathcal{G}$ (denoted by $\mathrm{pw}(\mathcal{G})$) is defined as the treewidth, just with respect to path decompositions. For the sake of convenience, we shall denote a path decomposition as a sequence $(B_1, B_2, \ldots, B_k)$ of sets of vertices without the component of the tree $\mathcal{T}$. Obviously, $\mathrm{tw}(\mathcal{G}) \leq \mathrm{pw}(\mathcal{G})$.

Tree decompositions for general τ-structures are defined in a similar way as for graphs, with the difference that the sets B_t contain now elements from the universe A of the structure instead of vertices. Furthermore, analogously as for tree decompositions of graphs, the sets $\{t \in T \mid a \in B_t\}$, $a \in A$, must be nonempty and connected in $\mathcal{T}$, but instead of requiring each edge to be represented in some B_t, we require that, for every relation symbol $R \in \tau$ and every tuple $(a_1, \ldots, a_{\mathrm{ar}(R)}) \in R^{\mathcal{A}}$ there is a $t \in T$ such that $a_1, \ldots, a_{\mathrm{ar}(R)} \in B_t$ (see Chapter 11 of Flum and

Grohe [19] for a detailed definition). Path decompositions, the treewidth and the pathwidth of relational structures are also defined in an analogous way as for graphs. Tree decompositions of relational structures can also be characterised in terms of classical graphs. To this end, we need the concept of the *Gaifman graph* of a τ-structure $\mathcal{A}$, which is the graph that has the universe A of $\mathcal{A}$ as vertices, and two vertices are connected if and only if they occur together in some relation (see Chapter 11 of Flum and Grohe [19]).

Proposition 2.13. *A relational structure has the same tree decompositions as its Gaifman graph.*

The previous proposition particularly implies that the treewidth of a structure equals the treewidth of its Gaifman graph. Thus, the Gaifman graph provides a convenient means to handle tree decompositions and the treewidth of structures. We say that a class of structures C has bounded treewidth if and only if there exists a $k \in \mathbb{N}$ such that, for every $\mathcal{A} \in C$, $\mathrm{tw}(\mathcal{A}) \leq k$.

The *homomorphism problem HOM* is the problem to decide, for given structures $\mathcal{A}$ and $\mathcal{B}$, whether there exists a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. For any set of structures C, by HOM(C) we denote the homomorphism problem that is restricted in such a way that the left hand input structure is from C. If C is a class of structures with bounded treewidth, then HOM(C) can be solved in polynomial time. This is a classical result that has been first achieved in terms of *constraint satisfaction problems* by Freuder [20] (see also Chapter 13 of Flum and Grohe [19]).

Theorem 2.14 (Freuder [20])**.** *Let C be a set of structures with bounded treewidth. Then HOM(C) is solvable in polynomial time.*

We shall briefly sketch how a tree decomposition of a structure $\mathcal{A}$ can be used in order to decide on whether or not there exists a homomorphism from $\mathcal{A}$ to another structure $\mathcal{A}'$. The naive way of deciding on the existence of a homomorphism is to simply enumerate all possible mappings from A to A', the universes of structures $\mathcal{A}$ and $\mathcal{A}'$, respectively, and check whether or not one of them satisfy the homomorphism condition. However, with a tree decomposition $(\mathcal{T} := (T, F), \{B_t \mid t \in T\})$ of $\mathcal{A}$, for every $t \in T$, we can first compute all mappings from B_t to A' that satisfy the homomorphism condition with respect to the elements in B_t. Then, by inductively merging these partial mappings according to the tree structure $\mathcal{T}$, we can construct a homomorphism from $\mathcal{A}$ to $\mathcal{A}'$ if one exists. The correctness of this last step is provided by the conditions stating that, for every $a \in A$, $\{t \in T \mid a \in B_t\}$ is nonempty and connected in $\mathcal{T}$ and, for every relation symbol $R \in \tau$ and every tuple $(a_1, \ldots, a_{\mathrm{ar}(R)}) \in R^{\mathcal{A}}$ there is a $t \in T$ such that $a_1, \ldots, a_{\mathrm{ar}(R)} \in B_t$. In this procedure, we do not need to enumerate complete mappings, but only mappings for a number of elements that is bounded by the width of the tree decomposition. Hence, the time complexity of this approach is exponential only in the treewidth.

Chapter 3

First Approach: Automata

In this chapter, we present our first approach to the membership problem for pattern languages, which is based on finite automata.

In Section 3.1, we compare multi-head automata with patterns and we present a few examples of how multi-head automata can recognise pattern languages. In Section 3.2, we introduce the so-called Nondeterministically Bounded Modulo Counter Automata (NBMCA), an automata model that is tailored to pattern languages. This model is quite general and for our actual application we need a slightly more specialised version of NBMCA, called Janus automata, which are also defined in Section 3.2. The more general NBMCA shall be subjected to a detailed analysis in Chapter 4. In Sections 3.3, we present our main result, which is achieved by applying Janus automata. We conclude this chapter by Section 3.4, in which we investigate a scheduling problem that is directly motivated by our application of Janus automata.

3.1 Multi-head Automata and Pattern Languages

In this section, we show how different variants of multi-head automata can recognise pattern languages. First, we show in a non-constructive way that nondeterministic two-way multi-head automata can recognise pattern language by observing that pattern languages can be recognised in nondeterministic logarithmic space. We then present a constructive way to transform a pattern into a nondeterministic two-way multi-head automaton that recognises the corresponding pattern language. Although the nondeterminism and the ability to move input heads in both directions is convenient for this construction, it is not necessary. More precisely, pattern languages can also be recognised by nondeterministic *one-way* multi-head automata and by *deterministic* two-way multi-head automata.

However, it seems impossible to recognise pattern languages with *deterministic one-way* multi-head automata, but we cannot formally prove this claim.

We now recall that nondeterministic two-way multi-head automata characterise the class NL of languages that can be recognised in nondeterministic logarithmic space (see, e. g., Sudborough [84]). For an arbitrary pattern α, we can define a nondeterministic logarithmic space bounded Turing machine T_α that accepts $L_{\mathrm{Z},\Sigma}(\alpha)$, $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. We illustrate this with an example. Let $\alpha := x_1\, x_2\, x_2\, x_1$ and let w be an input for T_α. We first guess two numbers k_1, k_2 between 0 and $|w|$ (or, in case of $\mathrm{Z} = \mathrm{NE}$, between 1 and $|w|$) and we store these numbers on the working tape, which requires only logarithmic space. Then we check whether or not these numbers induce a factorisation of w, i. e., $w = u_1\, u_2\, u_3\, u_4$ with $|u_1| = |u_4| = k_1$ and $|u_2| = |u_3| = k_2$. If this is satisfied, then we check whether or not this factorisation is a valid one, i. e., we check whether or not $u_1 = u_4$ and $u_2 = u_3$, which implies $w \in L_{\mathrm{Z},\Sigma}(\alpha)$. This can be done by using the input head of T_α and the numbers k_1, k_2 stored on the working tape. It is straightforward to generalise this procedure to arbitrary patterns. Thus, pattern languages can be recognised in nondeterministic logarithmic space, which particularly implies that pattern languages can be recognised by nondeterministic two-way multi-head automata.

However, we are interested in a constructive way to transform a pattern into a nondeterministic two-way multi-head automata that accepts the corresponding pattern language, which is provided by the following proposition:

Proposition 3.1. *Let $\alpha \in (\Sigma \cup X)^*$ be a pattern and let $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. There exists an $M \in \mathrm{2NFA}(2\,|\operatorname{var}(\alpha)| + 1)$ with $L(M) = L_{\mathrm{Z},\Sigma}(\alpha)$.*

Proof. Let $\alpha := y_1\, y_2 \cdots y_n$, $y_i \in (\Sigma \cup X)$, $1 \leq i \leq n$, and let $m := |\operatorname{var}(\alpha)|$. We now sketch how M checks whether or not some input w belongs to $L_{\mathrm{Z},\Sigma}(\alpha)$. The automaton M uses $2m$ of its input heads in order to implement m counters (denoted by *counter* 1, *counter* 2, ..., *counter* m) as described in Observation 2.11, (see page 21). Next, for every i, $1 \leq i \leq m$, counter i is incremented to a nondeterministically chosen value k_i (with $k_i \geq 1$, if $\mathrm{Z} = \mathrm{NE}$). Now M checks whether or not $w = u_1\, u_2 \cdots u_n$, where $|u_i| = 1$ if $y_i \in \Sigma$ and $|u_i| = k_j$ if $y_i = x_j$, which can be done by using the counters and the remaining head. If this holds, then the remaining head is used in order to check whether or not, for every $1 \leq i < j \leq n$, $y_i = y_j$ implies $u_i = u_j$. Again, this can be done with the aid of the counters. □

The proof of Proposition 3.1 heavily depends on both the nondeterminism of the automaton as well as its ability to move input heads in both directions. The question arises whether or not we can also recognise pattern languages by one-way multi-head automata or deterministic multi-head automata. We first show how pattern languages can be recognised by one-way nondeterministic multi-head automata. In order to give an idea for the proof of the following proposition, we recall that in the proof of Proposition 3.1, we nondeterministically guess numbers and then check whether these numbers induce a factorisation of the input word. For the construction in the one-way case, we again guess numbers, but we do not check whether these numbers induce a factorisation and simply proceed in a similar way as in the two-way case. Only at the end of the procedure, we then verify if the guessed numbers actually induce a factorisation of the input word.

Proposition 3.2. *Let* $\alpha \in (\Sigma \cup X)^*$ *be a pattern and let* $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. *There exists an* $M \in \mathrm{1NFA}(2\,|\alpha|)$ *with* $L(M) = L_{\mathrm{Z},\Sigma}(\alpha)$.

Proof. We prove the statement of the proposition only for the case $\mathrm{Z} = \mathrm{NE}$. The case $\mathrm{Z} = \mathrm{E}$ can be dealt with analogously.

Let $\alpha := y_1\, y_2 \cdots y_n$, $y_i \in (\Sigma \cup X)$, $1 \leq i \leq n$. We label n input heads by $m_1, m_2, \ldots, m_n$ and the remaining n input heads by $s_1, s_2, \ldots, s_n$. In the following procedure, all the heads m_i, $1 \leq i \leq n$, are used as mere markers that do not read input symbols and all the heads s_i, $1 \leq i \leq n$, are used in order to scan the input. Furthermore, for any input head m_i or s_i, by $p(m_i)$ or $p(s_i)$ we denote its current position in the input word. We now sketch how M checks whether or not some input w belongs to $L_{\mathrm{NE},\Sigma}(\alpha)$. First, we execute the following three steps.

1. All the input heads are nondeterministically moved to some positions of the input word, such that $1 = p(m_1) < p(m_2) < \ldots < p(m_n) \leq |w|$ and, for every i, $1 \leq i \leq n$, $p(m_i) = p(s_i)$.

2. For every i, $1 \leq i \leq n$, if $y_i \in \Sigma$, then we check whether or not head s_i scans symbol y_i and, if this is satisfied, we move head s_i one step to the right.

3. For every $x \in \mathrm{var}(\alpha)$ we do the following. For all the heads s_i, $1 \leq i \leq n$, with $y_i = x$, we check whether or not they are scanning exactly the same symbol and then all these heads are simultaneously moved one step to the right. This procedure is repeated for a nondeterministically chosen number of steps.

Next, we check whether or not $p(s_n) = |w| + 1$, i. e., s_n scans the right endmarker, and, for every i, $1 \leq i \leq n-1$, whether or not $p(s_i) = p(m_{i+1})$. This can be done by moving heads s_i and m_{i+1} simultaneously to the right and check whether or not they reach the right endmarker at the same time. If this is satisfied, then the input is accepted. If this is not satisfied or in the above procedure an input head that scans the right endmarker is moved to the right, then the input is rejected.

The correctness of the above procedure is established by the following considerations. Let $k_1, k_2, \ldots, k_n$ be the number of steps the input heads $s_1, s_2, \ldots, s_n$ are moved to the right in steps 2 and 3. If w is accepted by M, then this means that the numbers $s_1, s_2, \ldots, s_n$ are the splitting points of a factorisation of w, where the numbers $k_1, k_2, \ldots, k_n$ are the lengths of the factors. More precisely, $w := u_1 u_2 \cdots u_n$, where, for every i, $1 \leq i \leq n$, $|u_i| = k_i$ and for every i, $1 \leq i \leq n-1$, $|u_1 u_2 \cdots u_i| = m_{i+1} - 1$. From steps 2 and 3, it then follows immediately that, for every i, $1 \leq i \leq n$, if $y_i \in \Sigma$, then $u_i = y_i$ and, for every i, j, $1 \leq i < j \leq n$, if $y_i, y_j \in X$ and $y_i = y_j$, then $u_i = u_j$. Hence, $w \in L_{\mathrm{NE},\Sigma}(\alpha)$. On the other hand, if $w \in L_{\mathrm{NE},\Sigma}(\alpha)$, then there exists such a factorisation of α as described above and then the numbers $k_1, k_2, \ldots, k_n$ and the initial positions of the heads $m_1, m_2, \ldots, m_n$ can be guessed in such a way that w is accepted.

□

We note that for the whole procedure above the input head m_1 is not needed and, thus, $L_{\mathrm{NE},\Sigma}(\alpha)$ can be accepted by a nondeterministic one-way automaton with only $2\,|\alpha| - 1$ heads. We choose to use $2\,|\alpha|$ input heads out of convenience, since it simplifies the construction.

In the proofs of both Proposition 3.1 and 3.2, the nondeterminism of the automaton is used in order to guess a factorisation of the input word, which is then checked in a deterministic way. Hence, if deterministic multi-head automata are capable of recognising pattern languages, a different technique must be applied. In the next proposition, we state that deterministic two-way multi-head automata can recognise pattern languages, but we shall give a formal proof of this claim later in this thesis:

Proposition 3.3. *Let $\alpha \in (\Sigma \cup X)^*$ be a pattern and let $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. There exists an $M \in \mathrm{2DFA}(2\,|\mathrm{var}(\alpha)| + 1)$ with $L(M) = L_{\mathrm{Z},\Sigma}(\alpha)$.*

Intuitively speaking, a deterministic two-way multi-head automaton can recognise pattern language by deterministically trying out all possible factorisations of the input word and check every single one in a similar way as done in the proofs of Propositions 3.1 and 3.2. In Chapter 4, we shall introduce and investigate a special kind of multi-head automaton with restricted nondeterminism that is a very convenient tool in order to prove Proposition 3.3. Hence, we shall defer a formal proof of Proposition 3.3, which can be found in Section 4.2.3 on page 135.

3.2 Nondeterministically Bounded Modulo Counter Automata

We now present a special kind of counter automaton that is tailored to recognising pattern languages. This model constitutes the central tool that is applied in order to obtain the main result of this chapter. It is similar to the counter automata that are briefly explained in Section 2.3.1, i. e., it comprises a constant number of counters which form its main computational resource, but there are substantial differences in how these counters work.

A *Nondeterministically Bounded Modulo Counter Automaton*, NBMCA(k) for short, is a two-way one-head automaton with k counters. More precisely, it is a tuple $M := (k, Q, \Sigma, \delta, q_0, F)$, where $k \in \mathbb{N}$ is the number of *counters*, Q is a finite nonempty set of *states*, Σ is a finite nonempty alphabet of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states* and δ is a mapping $Q \times \Sigma \times \{\mathtt{t}_0, \mathtt{t}_1\}^k \to Q \times \{-1, 0, 1\} \times \{0, 1, \mathtt{r}\}^k$. The mapping δ is called the *transition function*. An *input* to M is any word of the form ¢w\$, where $w \in \Sigma^*$ and the symbols ¢, \$ (referred to as *left* and *right endmarker*, respectively) are not in Σ. Let $(p, b, s_1, \dots, s_k) \to_\delta (q, r, d_1, \dots, d_k)$. We call the element b the *scanned input symbol* and r the *input head movement*. For each $j \in \{1, 2, \dots, k\}$, the element $s_j \in \{\mathtt{t}_0, \mathtt{t}_1\}$ is the *counter message of counter* j, and d_j is called the *counter instruction for counter* j. The transition function δ of an NBMCA(k) determines whether the input head are moved to the left ($r = -1$), to the right ($r = 1$) or left unchanged ($r = 0$), and whether the counters are incremented ($d_j = 1$), left unchanged ($d_j = 0$) or reset ($d_j = \mathtt{r}$). In case of a reset, the counter value is set to 0 and a new counter bound is nondeterministically guessed. Hence, every counter is bounded, but these bounds are determined in a nondeterministic way. In order to define the language accepted by an NBMCA, we need to define the concept of an NBMCA computation.

Let M be an NBMCA and $w := b_1 \cdot b_2 \cdots b_n$, $b_i \in \Sigma$, $1 \leq i \leq n$. A *configuration of M (on input w)* is an element of the set

$$\widehat{C}_M := \{[q, h, (c_1, C_1), \dots, (c_k, C_k)] \mid q \in Q, 0 \leq h \leq n+1, \\ 0 \leq c_i \leq C_i \leq n, 1 \leq i \leq k\} \ .$$

The pair (c_i, C_i), $1 \leq i \leq k$, describes the current configuration of the i^{th} counter, where c_i is the *counter value* and C_i the *counter bound*. The element h is called the *input head position*.

An *atomic move* of M is denoted by the relation $\vdash_{M,w}$ over the set of configurations. Let $(p, b, s_1, \dots, s_k) \to_\delta (q, r, d_1, \dots, d_k)$. Then, for all c_i, C_i, $1 \leq i \leq k$, where $c_i < C_i$ if $s_i = \mathtt{t}_0$ and $c_i = C_i$ if $s_i = \mathtt{t}_1$, and for every h with $0 \leq h \leq n+1$, we define $[p, h, (c_1, C_1), \ldots, (c_k, C_k)] \vdash_{M,w} [q, h', (c'_1, C'_1), \ldots, (c'_k, C'_k)]$. Here, the elements h' and c'_j, C'_j, $1 \leq j \leq k$, are defined in the following way:

$$h' := \begin{cases} h + r & \text{if } 0 \leq h + r \leq n + 1 \ , \\ h & \text{else } . \end{cases}$$

For each $j \in \{1, \ldots, k\}$, if $d_j = \mathtt{r}$, then $c'_j := 0$ and, for some $m \in \{0, 1, \ldots, n\}$, $C'_j := m$. If, on the other hand, $d_j \neq \mathtt{r}$, then $C'_j := C_j$ and

$$c'_j := c_j + d_j \mod (C_j + 1) \, .$$

To describe a *sequence of (atomic) moves of M (on input w)* we use the reflexive and transitive closure of the relation $\vdash_{M,w}$, denoted by $\vdash^*_{M,w}$. M accepts the word w if and only if $\widehat{c}_0 \vdash^*_{M,w} \widehat{c}_f$, where $\widehat{c}_0 := [q_0, 0, (0, C_1), \ldots, (0, C_k)]$ for some $C_i \in \{0, 1, \ldots, |w|\}$, $1 \leq i \leq k$, is an *initial configuration*, and $\widehat{c}_f := [q_f, h, (c_1, C_1), \ldots (c_k, C_k)]$ for some $q_f \in F$, $0 \leq h \leq n + 1$ and $0 \leq c_i \leq C_i \leq n$, $1 \leq j \leq k$, is a *final configuration.*

In every computation of an NBMCA, the counter bounds are nondeterministically initialised, and the only nondeterministic step an NBMCA is able to perform during the computation consists in guessing a new counter bound for some counter. Apart from that, every transition is defined completely deterministically by δ.

Next, as an example, we define an NBMCA with only one counter that recognises the language $L_{\text{rev}} := \{ww^R \mid w \in \Sigma^*\}$.

Proposition 3.4. $L_{\text{rev}} \in \mathcal{L}(\text{NBMCA}(1))$.

Proof. We sketch how an NBMCA(1) can be defined that accepts L_{rev}. In a first step, by moving the input head from the left endmarker to the right endmarker, it is checked whether or not the message of the counter changes from $\mathtt{t}_0$ to $\mathtt{t}_1$ exactly when the input head reaches the right endmarker, i. e., whether or not the counter bound equals the length of the input. Furthermore, at the same time it is checked whether or not the input w has even length. This can be easily done with the finite state control. In case that $|w|$ is odd or the counter bound is not $|w|$, the input is rejected by entering a non-accepting trap state. Now, the counter can be used to execute the following three steps in a loop.

1. Move the input head one step to the right.
2. Move the input head for $|w| + 1$ steps by initially moving it to the right and reversing its direction if the right endmarker is reached.

3. Move the input head for $|w| + 1$ steps by initially moving it to the left and reversing its direction if the left endmarker is reached.

This loop is executed until the right endmarker is reached in step 1. It can be easily verified that this exactly happens in the $(|w| + 1)^{\text{th}}$ iteration of the loop. Furthermore, for every i, $1 \leq i \leq |w|$, in the i^{th} iteration of the loop, the position reached after step 1 is i and the position reached after step 2 is $|w| - i + 1$. So in order to check on whether or not $w = uu^R$, $u \in \Sigma^*$, it is sufficient to store the symbol at position i after step 1 in the finite state control and compare it to the symbol at position $|w| - i + 1$ after step 2 in each iteration of the loop. If eventually the right endmarker is reached after step 1, the automaton accepts its input and if, on the other hand, the symbol stored in the finite state control does not equal the symbol scanned after step 2, the input is rejected. □

In the above example, we use the nondeterminism of the NBMCA(1) in order to guess a specific position of the input word. Hence, one nondeterministic operation of the automaton is sufficient in order to recognise L_{rev}. This contrasts with how a pushdown automaton recognises L_{rev}, since this is usually done by moving the input head step by step to the right and guessing in every single step whether or not the middle position of the input is reached. Consequently, the number of nondeterministic steps of such a pushdown automaton is not bounded by a constant.

3.2.1 Janus Automata

In order to prove the main results of this chapter, we use a slightly different version of NBMCA, i. e., NBMCA with two instead of only one input head, which we call *Janus automata* (the choice of this name shall be explained later on). We require the first input head to be always positioned to the left of the second input head, so there are a well-defined left and right head. For the sake of completeness, we now present a formal definition of Janus automata and mention that this definition is analogous to the one of NBMCA with the small differences explained above.

A *Janus automaton with k counters* (denoted by JFA(k)) is a device $M := (k, Q, \Sigma, \delta, q_0, F)$, where $k \geq 0$ is the number of *counters*, Q is a finite nonempty set of *states*, Σ is a finite nonempty alphabet of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states* and δ is a mapping $Q \times \Sigma^2 \times \{\mathtt{t}_=, \mathtt{t}_<\}^k \to Q \times \{-1, 0, 1\}^2 \times \{0, 1, \mathtt{r}\}^k$. The mapping δ is called the *transition function*.

An *input* to M is any string of the form $¢w\$$, where $w \in \Sigma^*$ and the symbols $¢, \$$ (referred to as *left* and *right endmarker*, respectively) are not in Σ. Let $\delta(p, a_1, a_2, s_1, \ldots, s_k) = (q, r_1, r_2, d_1, \ldots, d_k)$. For each $i \in \{1, 2\}$, we call the element a_i the *input symbol scanned by head i* and r_i the *instruction for head i*. For each $j \in \{1, 2, \dots, k\}$, the element $s_j \in \{\mathtt{t}_=, \mathtt{t}_<\}$ is the *counter message of counter* j, and d_j is called the *counter instruction for counter* j.

Let $M := (k, Q, \Sigma, \delta, q_0, F)$ be a JFA(k) and $w := b_1 \cdot b_2 \cdot \ldots \cdot b_n$, $b_i \in \Sigma$, $1 \leq i \leq n$. A *configuration of* M *(on input* $¢w\$$*)* is an element of the set

$$\widehat{C}_M := \{(q, h_1, h_2, (c_1, C_1), \dots, (c_k, C_k)) \mid q \in Q, 0 \leq h_1 \leq h_2 \leq n+1, \\ 0 \leq c_i \leq C_i \leq n, 1 \leq i \leq k\} \ .$$

The pair (c_i, C_i), $1 \leq i \leq k$, describes the current configuration of the ith counter, where c_i is the *counter value* and C_i the *counter bound*. The element h_i, $i \in \{1, 2\}$, is called the *head position of head* i.

An *atomic move of* M *(on input* $¢w\$$*)* is denoted by the relation $\vdash_{M,w}$ over the set of configurations. Let $\delta(p, a_1, a_2, s_1, \dots, s_k) = (q, r_1, r_2, d_1, \dots, d_k)$. Then, for all c_i, C_i, $1 \leq i \leq k$, where $c_i < C_i$ if $s_i = \mathtt{t}_<$ and $c_i = C_i$ if $s_i = \mathtt{t}_=$, and for all h_1, h_2, $0 \leq h_1 \leq h_2 \leq n+1$, with $b_{h_i} = a_i$, $i \in \{1, 2\}$, we define $(p, h_1, h_2, (c_1, C_1), \dots, (c_k, C_k)) \vdash_{M,w} (q, h'_1, h'_2, (c'_1, C'_1), \dots, (c'_k, C'_k))$. Here, the elements h'_i, $i \in \{1, 2\}$, and c'_j, C'_j, $1 \leq j \leq k$, are defined as follows:

$$h'_i := \begin{cases} h_i + r_i & \text{if } 0 \leq h_1 + r_1 \leq h_2 + r_2 \leq n+1 \ , \\ h_i & \text{else } . \end{cases}$$

For each $j \in \{1, \dots, k\}$, if $d_j = \mathtt{r}$, then $c'_j := 0$ and, for some $m \in \{0, 1, \dots, n\}$, $C'_j := m$. If, on the other hand, $d_j \neq \mathtt{r}$, then $C'_j := C_j$ and

$$c'_j := c_j + d_j \mod (C_j + 1)\,.$$

To describe a *sequence of (atomic) moves of* M *(on input* w*)* we use the reflexive and transitive closure of the relation $\vdash_{M,w}$, denoted by $\vdash^*_{M,w}$. M accepts the word w if and only if $\widehat{c}_0 \vdash^*_{M,w} \widehat{c}_f$, where $\widehat{c}_0 := (q_0, 0, 0, (0, 0), \dots, (0, 0))$ is the *initial configuration*, and $\widehat{c}_f := (q_f, h_1, h_2, (c_1, C_1), \dots (c_k, C_k))$ is a *final configuration*, for some $q_f \in F$, $0 \leq h_1 \leq h_2 \leq n+1$ and $0 \leq c_i \leq C_i \leq n$, $1 \leq j \leq k$. For any Janus automaton M, let $L(M)$ denote the set of words accepted by M.

In our applications of this automata model, we use the counters in a particular but natural way. Let us assume that n is the counter bound of a certain counter with counter value 0. We can define the transition function in such a way that an input head is successively moved to the right and, in every step, the counter is incremented. As soon as the counter reaches its counter bound (i.e., its counter message changes from $\mathtt{t}_<$ to $\mathtt{t}_=$) we stop that procedure and can be sure that the input head has been moved exactly n steps. In this way an automaton can scan whole factors of the input, induced by counter bounds. Furthermore, as we have two input heads, we can use the counter with bound n to move them simultaneously to the right, checking symbol by symbol whether two factors of equal length are the same. It is also worth mentioning that we can use counters in the same way to move input heads from right to left instead of from left to right.

This way of using counters shall be made clear by sketching how a Janus automaton M could be defined that recognises the language

$$L := \{u\,\mathtt{a}\,v\,\mathtt{b}\,v\,u \mid u, v \in \{\mathtt{a}, \mathtt{b}\}^*\}\,.$$

The Janus automaton M uses two counters and applies the following strategy to check whether an input word w is in L. First, we reset both counters and therefore guess two new counter bounds C_1 and C_2. Then we check if $w = u\,\mathtt{a}\,v\,\mathtt{b}\,v\,u$ with $|u| = C_1$ and $|v| = C_2$. This is done by using the first counter to move the right head from position 1 (the symbol next to the left endmarker) to the right until it reaches position C_1+1. Then it is checked whether a occurs at this position. After that, by using the second counter, the right head is moved further to the right to position $C_1 + C_2 + 2$, where M checks for the occurrence of the symbol b. Next, again by using the second counter, the right head is moved another $C_2 + 1$ steps to the right in order to place it exactly where we expect the second occurrence of factor u to begin. Now, both input heads are moved simultaneously to the right for C_1 steps, checking in each step whether they scan the same symbol and whether after these C_1 steps the right head scans exactly the right endmarker. If this is successful, we know that w is of form $u\,\mathtt{a}\,v\,\mathtt{b}\,v'\,u$, with $|u| = C_1$ and $|v| = |v'| = C_2$. Hence, it only remains to check whether or not $v = v'$. This can be done by positioning both heads at the first positions of the factors v and v', i. e., moving the left head one step to the right and the right head $C_1 + C_2$ steps back to the left. In order to perform this, as well as the final matching of the factors v and v', M can apply its counters in the same way as before. If this whole procedure is successful, M enters an accepting state, and reject its input otherwise.

It is obvious that $w \in L$ if and only if there is a possibility to guess counter bounds such that M accepts w; thus, $L(M) = L$.

In the above procedure, the automaton can be interpreted as looking back (by means of the left head) and forward (by means of the right head) at the same time in order to compare an earlier factor with a later one. This explains why the automaton is called Janus automaton.

3.3 Large Classes of Patterns with a Polynomial Time Membership Problem

In this section, we present a quite general way of how a pattern can be transformed into a Janus automaton that recognises the corresponding pattern language. To this end, we define an intermediate model, which, intuitively speaking, is a sequence of instructions that tells a JFA how to move its input heads in order to recognise the pattern language. We further identify a parameter of these instruction sequences which determines the number of counters that a JFA requires in order to carry them out. Finally, it is shown that this parameter is bounded by the variable distance (see Section 2.2.1) of the corresponding pattern, which yields our main result, namely that the membership problem for pattern languages is solvable in polynomial time if the variable distance of the patterns is bounded.

In this section, the dependency on the terminal alphabet Σ is negligible and, furthermore, we shall only consider the E case and terminal-free patterns. All our results can be easily extended to the NE case, and also their generalisation to patterns with terminal symbols is straightforward. Hence, for the sake of convenience, for any terminal free pattern α, we denote $L_{\mathrm{E},\Sigma}(\alpha)$ by $L(\alpha)$ in the following.

3.3.1 Janus Automata for Pattern Languages

In this section, we demonstrate how Janus automata can be used for recognising pattern languages. More precisely, for an arbitrary terminal-free pattern α, we construct a JFA(k) M satisfying $L(M) = L(\alpha)$. Before we move on to a formal analysis of this task, we discuss the problem of deciding whether $w \in L(\alpha)$ for given α and w, i. e., the membership problem, in an informal way. We point out that the following basic ideas are sketched in Section 1.1 and are applied in the proof of Proposition 3.1 (see Section 3.1).

Let $\alpha = y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern with $m := |\operatorname{var}(\alpha)|$, and let $w \in \Sigma^*$ be a word. The word w is an element of $L(\alpha)$ if and only if there exists a factorisation $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$ such that $u_j = u_{j'}$ for all j, j', $1 \leq j < j' \leq |\alpha|$, with $y_j = y_{j'}$. We call such a factorisation $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$ a *characteristic factorisation for* $w \in L(\alpha)$ (or simply *characteristic factorisation* if w and α are obvious from the context). Thus, a way to solve the membership problem is to initially guess m numbers $l_1, l_2, \ldots, l_m$, then, if possible, to factorise $w = u_1 \cdot \ldots \cdot u_n$ such that $|u_j| = l_i$ for all j with $y_j = x_i$ and, finally, to check whether this is a characteristic factorisation for $w \in L(\alpha)$. A JFA(m) can perform this task by initially guessing m counter bounds, which can be interpreted as the lengths of the factors. The two input heads can be used to check if this factorisation has the above described properties. However, the number of counters that are then required directly depends on the number of variables, and the question arises if this is always necessary.

In the next definitions, we shall establish the concepts that formalise and generalise the way of checking whether or not a factorisation is a characteristic one.

Definition 3.5. Let $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern, and, for each $x_i \in \mathrm{var}(\alpha)$, let $n_i := |\alpha|_{x_i}$. The set *varpos*$_i(\alpha)$ is the set of all positions j satisfying $y_j = x_i$. The sequence $((l_1, r_1), (l_2, r_2), \ldots, (l_{n_i-1}, r_{n_i-1}))$ with $(l_j, r_j) \in \mathrm{varpos}_i(\alpha)^2$ and $l_j < r_j$, $1 \leq j \leq n_i - 1$, is a *matching order for* x_i *in* α if and only if the graph $(\mathrm{varpos}_i(\alpha), \{\{l_1, r_1\}, \{l_2, r_2\}, \ldots, \{l_{n_i-1}, r_{n_i-1}\}\})$ is a tree.

We consider an example in order to illustrate Definition 3.5. If, for some pattern α and some $x_i \in \mathrm{var}(\alpha)$, $\mathrm{varpos}_i(\alpha) := \{1, 3, 5, 9, 14\}$, then the sequences $((5, 1), (14, 3), (1, 3), (9, 3))$, $((1, 3), (3, 5), (5, 9), (9, 14))$ and $((5, 1),$ $(5, 3),$ $(5, 9),$ $(5, 14))$ are some of the possible matching orders for x_i in α, whereas the sequences $((1, 3), (9, 1), (3, 9), (5, 14))$ and $((1, 3), (3, 5), (5, 9), (9, 1))$ do not satisfy the conditions to be matching orders for x_i in α.

In order to obtain a matching order for a whole pattern α we simply combine matching orders for all $x \in \mathrm{var}(\alpha)$:

Definition 3.6. Let α be a terminal-free pattern with $m := |\,\mathrm{var}(\alpha)|$ and, for all i with $1 \leq i \leq m$, $n_i := |\alpha|_{x_i}$ and let $(m_{i,1}, m_{i,2}, \ldots, m_{i,n_i-1})$ be a matching order for x_i in α. The tuple $(m_1, m_2, \ldots, m_k)$ is a *complete matching order for* α if and only if $k = \sum_{i=1}^{m}(n_i - 1)$ and, for all i, j_i, $1 \leq i \leq m$, $1 \leq j_i \leq n_i - 1$, there is a j', $1 \leq j' \leq k$, with $m_{j'} = m_{i,j_i}$. The elements $m_j \in \mathrm{varpos}_i(\alpha)^2$ of a matching order $(m_1, m_2, \ldots, m_k)$ are called *matching positions.*

We introduce an example pattern

$$\beta := x_1 \cdot x_2 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_2 \cdot x_3\,,$$

which we shall use throughout the whole section in order to illustrate the main definitions. Regarding Definition 3.6, we observe that all possible sequences of the matching positions in $\{(1, 3), (2, 4), (4, 6), (5, 7)\}$ are some of the possible complete matching orders for β. As pointed out by the following lemma, the concept of a complete matching order can be used to check whether a factorisation is a characteristic one.

Lemma 3.7. *Let $\alpha = y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern and let $((l_1, r_1),$ $(l_2, r_2), \ldots, (l_k, r_k))$ be a complete matching order for α. Let w be an arbitrary word in some factorisation $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$. If $u_{l_j} = u_{r_j}$ for every j, $1 \leq j \leq k$, then $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$ is a characteristic factorisation.*

Proof. Let $x_i \in \mathrm{var}(\alpha)$ be arbitrarily chosen and let the sequence $((l'_1, r'_1), (l'_2, r'_2), \ldots, (l'_{k'}, r'_{k'}))$ be an arbitrary matching order for x_i in α. Assume that $u_{l'_j} = u_{r'_j}$ for all j, $1 \leq j \leq k'$. As $(\mathrm{varpos}_i(\alpha), \{(l'_1, r'_1), (l'_2, r'_2), \ldots, (l'_{k'}, r'_{k'})\})$ is a connected graph and as the equality of words is clearly a transitive relation, we can conclude that $u_j = u_{j'}$ for all j, j', $1 \leq j < j' \leq |\alpha|$, with $y_j = y_{j'} = x_i$. Applying this argumentation to all variables in α implies the statement of Lemma 3.7. □

With respect to the complete matching order $((4, 6), (1, 3), (2, 4), (5, 7))$ for the example pattern β, we apply Lemma 3.7 in the following way. If w can be factorised into $w = u_1 \cdot u_2 \cdot \ldots \cdot u_7$ such that $u_4 = u_6$, $u_1 = u_3$, $u_2 = u_4$ and $u_5 = u_7$, then $w \in L(\beta)$.

Let (l_1, r_1) and (l_2, r_2) be two consecutive matching positions of a complete matching order. It is possible to perform the comparison of factors u_{l_1} and u_{r_1} by positioning the left head on the first symbol of u_{l_1}, the right head on the first symbol of u_{r_1} and then moving them simultaneously over these factors from left to right, checking symbol by symbol if these factors are identical (cf. the example Janus automaton in Section 3.2.1). After that, the left head, located at the first symbol of factor u_{l_1+1}, has to be moved to the first symbol of factor u_{l_2}. If $l_1 < l_2$, then it is sufficient to move it over all the factors $u_{l_1+1}, u_{l_1+2}, \ldots, u_{l_2-1}$. If, on the other hand, $l_2 < l_1$, then the left head has to be moved to the left, and, thus, over the factors u_{l_1} and u_{l_2} as well. Furthermore, as we want to apply these ideas to Janus automata, the heads must be moved in a way that the left head is always located to the left of the right head. The following definition shall formalise these ideas.

Definition 3.8. In the following definition, let λ and ρ be constant markers. For all $j, j' \in \mathbb{N}$ with $j < j'$, we define a mapping g by $g(j, j') := (j+1, j+2, \ldots, j'-1)$ and $g(j', j) := (j', j'-1, \ldots, j)$.

Let $((l_1, r_1), (l_2, r_2), \ldots, (l_k, r_k))$ be a complete matching order for a terminal-free pattern α and let $l_0 := r_0 := 0$. For every matching position (l_i, r_i), $1 \leq i \leq k$, we define a sequence D_i^λ and a sequence D_i^ρ by

$$D_i^\lambda := ((p_1, \lambda), (p_2, \lambda), \ldots, (p_{k_1}, \lambda)) \text{ and}$$
$$D_i^\rho := ((p'_1, \rho), (p'_2, \rho), \ldots, (p'_{k_2}, \rho)),$$

where $(p_1, p_2, \ldots, p_{k_1}) := g(l_{i-1}, l_i)$, $(p'_1, p'_2, \ldots, p'_{k_2}) := g(r_{i-1}, r_i)$.

Now let $D'_i := ((s_1, \mu_1), (s_2, \mu_2), \ldots, (s_{k_1+k_2}, \mu_{k_1+k_2}))$ be a tuple satisfying the following two conditions. Firstly, it contains exactly the elements of D_i^λ and D_i^ρ such that the relative orders of the elements in D_i^λ and D_i^ρ are preserved. Secondly, for every j, $1 \leq j \leq k_1 + k_2$, $s_{j_l} \leq s_{j_r}$ needs to be satisfied, with $j_l = \max(\{j' \mid$

$1 \leq j' \leq j, \mu_{j'} = \lambda\} \cup \{j'_l\})$ and $j_r = \max(\{j' \mid 1 \leq j' \leq j, \mu_{j'} = \rho\} \cup \{j'_r\})$, where $(s_{j'_l}, \mu_{j'_l})$ and $(s_{j'_r}, \mu_{j'_r})$ are the leftmost elements of D'_i with $\mu_{j'_l} = \lambda$ and $\mu_{j'_r} = \rho$, respectively.

Now we append the two elements (r_i, ρ), (l_i, λ) in exactly this order to the end of D'_i and obtain D_i. Finally, the tuple $(D_1, D_2, \dots, D_k)$ is called a *Janus operating mode for α (derived from the complete matching order $((l_1, r_1), \dots, (l_k, r_k))$).*

We once again consider the example $\beta = x_1 \cdot x_2 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_2 \cdot x_3$. According to Definition 3.8 we consider the tuples D^λ_i and D^ρ_i with respect to the complete matching order $((4,6),(1,3),(2,4),(5,7))$ for β. We omit the markers λ and ρ for a better presentation. The tuples D^λ_i and D^ρ_i, $1 \leq i \leq 4$, are given by

$$\begin{aligned} D^\lambda_1 &= (1,2,3)\,, & D^\rho_1 &= (1,2,\dots,5)\,,\\ D^\lambda_2 &= (4,3,2,1)\,, & D^\rho_2 &= (6,5,4,3)\,,\\ D^\lambda_3 &= ()\,, & D^\rho_3 &= ()\,,\\ D^\lambda_4 &= (3,4)\,, & D^\rho_4 &= (5,6)\,. \end{aligned}$$

Therefore, $\Delta_\beta := (D_1, D_2, D_3, D_4)$ is a possible Janus operating mode for β derived from $((4,6),(1,3),(2,4),(5,7))$, where

$$\begin{aligned} D_1 &= ((1,\rho),(1,\lambda),(2,\rho),(2,\lambda),(3,\rho),(3,\lambda),(4,\rho),(5,\rho),(6,\rho),(4,\lambda)),\\ D_2 &= ((4,\lambda),(3,\lambda),\dots,(1,\lambda),(6,\rho),(5,\rho),\dots,(3,\rho),(3,\rho),(1,\lambda)),\\ D_3 &= ((4,\rho),(2,\lambda)),\\ D_4 &= ((3,\lambda),(5,\rho),(4,\lambda),(6,\rho),(7,\rho),(5,\lambda)). \end{aligned}$$

Intuitively, we interpreted a complete matching order as a list of instructions specifying how to check whether a factorisation is a characteristic one. Similarly, a Janus operating mode derived from a complete matching order can be seen as an extension of this complete matching order that also contains information of how two input heads have to be moved from one matching position to the next one. Hence, there is an immediate connection between Janus operating modes and Janus automata for terminal-free pattern languages, and we shall see that it is possible to transform a Janus operating mode for any pattern directly into a Janus automaton recognising the corresponding pattern language. As we are particularly interested in the number of counters a Janus automaton needs, we introduce an instrument to determine the quality of Janus operating modes with respect to the number of counters that are required to actually construct a Janus automaton.

Definition 3.9. Let $\Delta_\alpha := (D_1, D_2, \dots, D_k)$ be a Janus operating mode for a terminal-free pattern $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$. The *head movement indicator* of Δ_α is the tuple $\overline{\Delta_\alpha} = ((d'_1, \mu'_1), (d'_2, \mu'_2), \ldots, (d'_{k'}, \mu'_{k'}))$ with $k' = \sum_{i=1}^{k} |D_i|$ that is obtained by concatenating all tuples D_j, $1 \leq j \leq k$, in the order given by the Janus operating mode. For every i, $1 \leq i \leq k'$, let

$$s_i := |\{x \mid \exists\, j, j' \text{ with } 1 \leq j < i < j' \leq k', y_{d'_j} = y_{d'_{j'}} = x \neq y_{d'_i}\}|\,.$$

Then the *counter number of* Δ_α (or $\mathrm{cn}(\Delta_\alpha)$ for short) is $\max\{s_i \mid 1 \leq i \leq k'\}$.

We now briefly explain the previous definition in an informal manner. Apart from the markers λ and ρ, the head movement indicator $\overline{\Delta_\alpha}$, where Δ_α is a Janus operating mode for some α, can be regarded as a sequence $(d'_1, d'_2, \ldots, d'_{k'})$, where the d'_i, $1 \leq i \leq k'$, are positions in α. Hence, we can associate a pattern $D_\alpha := y_{d'_1} \cdot y_{d'_2} \cdot \ldots \cdot y_{d'_{k'}}$ with $\overline{\Delta_\alpha}$. In order to determine the counter number of Δ_α, we consider each position i, $1 \leq i \leq k'$, in D_α and count the number of variables different from $y_{d'_i}$ that are parenthesising position i in D_α. The counter number is then the maximum over all these numbers.

With regard to our example β, it can be easily verified that $\mathrm{cn}(\Delta_\beta) = 2$. We shall now see that, for every Janus operating mode Δ_α for a pattern α, we can construct a Janus automaton recognising $L(\alpha)$ with exactly $\mathrm{cn}(\Delta_\alpha) + 1$ counters:

Theorem 3.10. *Let α be a terminal-free pattern and let Δ_α be an arbitrary Janus operating mode for α. There exists a* $\mathrm{JFA}(\mathrm{cn}(\Delta_\alpha)+1)$ *M satisfying $L(M) = L(\alpha)$.*

Before we can prove this result, we need the following technical lemma:

Lemma 3.11. *Let α be a terminal-free pattern with $|\mathrm{var}(\alpha)| \geq 2$, and let $\Gamma := \{z_1, z_2, \ldots, z_m\} \subseteq \mathrm{var}(\alpha)$. The following statements are equivalent:*

a. For all $z, z' \in \Gamma$, $z \neq z'$, the pattern α can be factorised into $\alpha = \beta \cdot z \cdot \gamma \cdot z' \cdot \gamma' \cdot z \cdot \delta$ or $\alpha = \beta \cdot z' \cdot \gamma \cdot z \cdot \gamma' \cdot z' \cdot \delta$.

b. There exists a $z \in \Gamma$ such that α can be factorised into $\alpha = \beta \cdot z \cdot \gamma$ with $(\Gamma/\{z\}) \subseteq (\mathrm{var}(\beta) \cap \mathrm{var}(\gamma))$.

Proof. We prove by contraposition that a implies b. Hence, we assume that there exists no $z \in \Gamma$ such that α can be factorised into $\alpha = \beta \cdot z \cdot \gamma$ with $(\Gamma/\{z\}) \subseteq (\mathrm{var}(\beta) \cap \mathrm{var}(\gamma))$. Next, we define $l_1, l_2, \ldots, l_m$ to be the leftmost occurrences and $r_1, r_2, \ldots, r_m$ to be the rightmost occurrences of the variables $z_1, z_2, \ldots, z_m$ in α. Furthermore, we assume $l_1 < l_2 < \ldots < l_m$. By assumption, it is not possible that, for every i, $1 \leq i \leq m-1$, $r_i > l_m$ as this implies that α can be factorised into $\alpha = \beta \cdot z_m \cdot \gamma$, $|\beta| = l_m - 1$ with $(\Gamma/\{z_m\}) \subseteq (\mathrm{var}(\beta) \cap \mathrm{var}(\gamma))$. So we can assume that there exists an i, $1 \leq i \leq m-1$, with $r_i < l_m$. This implies that, for z_i, z_m, α can neither be factorised into $\alpha = \beta \cdot z_i \cdot \gamma \cdot z_m \cdot \gamma' \cdot z_i \cdot \delta$ nor into $\alpha = \beta \cdot z_m \cdot \gamma \cdot z_i \cdot \gamma' \cdot z_m \cdot \delta$. This proves that a implies b.

The converse statement, b implies a, can be easily comprehended. We assume that $z \in \Gamma$ satisfies the conditions of b, i. e., α can be factorised into $\alpha = \beta \cdot z \cdot \gamma$ with $(\Gamma/\{z\}) \subseteq (\text{var}(\beta) \cap \text{var}(\gamma))$. Now we arbitrarily choose $z', z'' \in \Gamma$, $z' \neq z''$, and we shall show that $\alpha = \beta' \cdot z' \cdot \gamma' \cdot z'' \cdot \gamma'' \cdot z' \cdot \delta'$ or $\alpha = \beta' \cdot z'' \cdot \gamma' \cdot z' \cdot \gamma'' \cdot z'' \cdot \delta'$. If either $z' = z$ or $z'' = z$, this is obviously true. In all other cases, the fact that there are occurrences of both z' and z'' to either side of the occurrence of z directly implies the existence of one of the aforementioned factorisations. □

Now we are able to present the proof of Theorem 3.10:

Proof. Let $\pi := \text{cn}(\Delta_\alpha) + 1$. In order to prove Theorem 3.10, we illustrate a general way of transforming a Janus operating mode $\Delta_\alpha := (D_1, D_2, \dots, D_k)$ of an arbitrary terminal-free pattern $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ into a Janus automaton M with $\text{cn}(\Delta_\alpha) + 1$ counters satisfying $L(M) = L(\alpha)$. We shall first give a definition of the automaton and then prove its correctness, i. e., $L(M) = L(\alpha)$.

We assume that the Janus operating mode is derived from the complete matching order $(m_1, m_2, \dots, m_k)$. Let us recall the main definitions that are used in this proof, namely the complete matching order and the Janus operating mode. We know that each element m_i, $1 \leq i \leq k$, of the complete matching order is a matching position, i. e., $m_i = (l_i, r_i)$, $l_i < r_i$ and $y_{l_i} = y_{r_i}$. The complete matching order is included in the Janus operating mode, since, for each i, $1 \leq i \leq k$, the tuple D_i corresponds to the matching position m_i in the following way: If $m_i = (l_i, r_i)$, then the last two elements of D_i are (r_i, ρ) and (l_i, λ). All the other pairs in a D_i are of form (j, μ) where $1 \leq j \leq |\alpha|$ and $\mu \in \{\lambda, \rho\}$.

Before we move on to the formal definitions of the states and transitions of the automaton, let us illustrate its behaviour in an informal way. As described at the beginning of Section 3.3.1, the membership problem can be solved by checking the existence of a characteristic factorisation $u_1 \cdot u_2 \cdot \ldots \cdot u_n$ of the input w. Furthermore, by Lemma 3.7, the complete matching order can be used as a list of instructions to perform this task. The factorisation is defined by the counter bounds, i. e., for every variable $x \in \text{var}(\alpha)$, the automaton uses a certain counter, the counter bound of which defines the length of all the factors u_i with $y_i = x$. However, if $\pi < |\text{var}(\alpha)|$ is satisfied, then the automaton does not have the number of counters required for such a representation. Therefore, it might be necessary to reuse counters. To define which counter is used for which variables, we use a mapping $\text{co} : \text{var}(\alpha) \to \{1, 2, \dots, \pi\}$. Note that, in case of $\pi < |\text{var}(\alpha)|$, this mapping is not injective. We defer a complete definition of the mapping co and, for now, just assume that there exists such a mapping.

Next, we show how a tuple D_p for an arbitrary p, $1 \leq p \leq k$, can be transformed into a part of the automaton. Therefore, we define

$$D_p := ((j_1, \mu_1), (j_2, \mu_2), \dots, (j_{k'}, \mu_{k'}), (j_r, \rho), (j_l, \lambda))$$

with $\mu_i \in \{\lambda, \rho\}$, $1 \leq i \leq k'$. Recall that D_p corresponds to the matching position $m_p := (j_l, j_r)$. Let us interpret the tuple D_p as follows: The pairs $(j_1, \mu_1), (j_2, \mu_2), \dots, (j_{k'}, \mu_{k'})$ define how the heads have to be moved in order to reach factors u_{j_l} and u_{j_r}, which then have to be matched. Let (j_i, μ_i), $1 \leq i \leq k'$, be an arbitrary pair of D_p. If $\mu_i = \lambda$ (or $\mu_i = \rho$), then the meaning of this pair is that the left head (or the right head, respectively) has to be moved a number of steps defined by the counter bound of counter $\mathrm{co}(y_{j_i})$. The direction the head has to be moved to depends on the matching position corresponding to the previous element D_{p-1}. In order to define these ideas formally, we refer to this previous matching position by $m_{p-1} := (j'_l, r'_l)$.

If $j'_l < j_l$, then we have to move the left head to the right passing the factors $u_{j'_l+1}, u_{j'_l+2}, \dots, u_{j_l-1}$; thus, we introduce the following states:

$$\{\text{l-forth}_{p,q} \mid j'_l + 1 \leq q \leq j_l - 1\} \ .$$

In every state $\text{l-forth}_{p,q}$, $j'_l + 1 \leq q \leq j_l - 1$, we move the left head as many steps to the right as determined by the currently stored counter bound for counter $\mathrm{co}(y_q)$. Hence, for every q, $j'_l + 1 \leq q \leq j_l - 1$, for all $a, a' \in \Sigma$ and for every $s_i \in \{\mathtt{t}_=, \mathtt{t}_<\}$, $i \in \{1, \dots, \pi\}/\{\mathrm{co}(y_q)\}$, we define

$$\delta(\text{l-forth}_{p,q}, a, a', s_1, s_2, \dots, s_\pi) := (\text{l-forth}_{p,q}, 1, 0, d_1, d_2, \dots, d_\pi) \ ,$$

where $s_{\mathrm{co}(y_q)} := \mathtt{t}_<$, $d_{\mathrm{co}(y_q)} := 1$, and, for every $i \in \{1, \dots, \pi\}/\{\mathrm{co}(y_q)\}$, $d_i := 0$.

Analogously, if $j_l < j'_l$, then we have to move the left head to the left over the factors $u_{j'_l}, u_{j'_l-1}, \dots, u_{j_l+1}, u_{j_l}$; to this end we use the following set of states:

$$\{\text{l-back}_{p,q} \mid j_l \leq q \leq j'_l\} \ .$$

As before, for every q, $j_l \leq q \leq j'_l$, for all $a, a' \in \Sigma$ and for every $s_i \in \{\mathtt{t}_=, \mathtt{t}_<\}$, $i \in \{1, \dots, \pi\}/\{\mathrm{co}(y_q)\}$, we define

$$\delta(\text{l-back}_{p,q}, a, a', s_1, s_2, \dots, s_\pi) := (\text{l-back}_{p,q}, -1, 0, d_1, d_2, \dots, d_\pi) \ ,$$

where $s_{\mathrm{co}(y_q)} := \mathtt{t}_<$, $d_{\mathrm{co}(y_q)} := 1$, and, for every $i \in \{1, \dots, \pi\}/\{\mathrm{co}(y_q)\}$, $d_i := 0$.

Note that, in the above defined transitions, the only difference between the cases $j'_l < j_l$ and $j_l < j'_l$, apart from the different states, is the head instruction for the left head. The states for the right head, i. e., r-forth$_{p,q}$ and r-back$_{p,q}$, and their transitions are defined analogously.

Up to now, we have introduced states that can move the input heads back or forth over whole factors of the input word. This is done by moving an input head and simultaneously incrementing a counter until it reaches the counter bound, i. e., the counter message changes to $\mathtt{t}_=$. It remains to define what happens if an input head is completely moved over a factor and the counter message changes to $\mathtt{t}_=$. Intuitively, in this case the automaton should change to another state and then move a head in dependency of another counter. Thus, e. g., if in state l-forth$_{p,i}$ the counter message of counter $\mathrm{co}(y_i)$ is $\mathtt{t}_=$, then the automaton should change into state l-forth$_{p,i+1}$. In order to simplify the formal definition we assume $j'_l < j_l$ and $j'_r < j_r$, as all other cases can be handled similarly. For every q, $1 \leq q \leq k' - 1$, for all $a, a' \in \Sigma$ and for every $s_i \in \{\mathtt{t}_=, \mathtt{t}_<\}$, $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, we define

$$\begin{aligned}&\delta(\text{l-forth}_{p,q}, a, a', s_1, s_2, \ldots, s_\pi) := (\text{l-forth}_{p,q+1}, 0, 0, d_1, d_2, \ldots, d_\pi)\ ,\\ &\text{if } \mu_p = \lambda \text{ and } \mu_{p+1} = \lambda\ ,\end{aligned}$$

$$\begin{aligned}&\delta(\text{l-forth}_{p,q}, a, a', s_1, s_2, \ldots, s_\pi) := (\text{r-forth}_{p,q+1}, 0, 0, d_1, d_2, \ldots, d_\pi)\ ,\\ &\text{if } \mu_p = \lambda \text{ and } \mu_{p+1} = \rho\ ,\end{aligned}$$

$$\begin{aligned}&\delta(\text{r-forth}_{p,q}, a, a', s_1, s_2, \ldots, s_\pi) := (\text{l-forth}_{p,q+1}, 0, 0, d_1, d_2, \ldots, d_\pi)\ ,\\ &\text{if } \mu_p = \rho \text{ and } \mu_{p+1} = \lambda\ ,\end{aligned}$$

$$\begin{aligned}&\delta(\text{r-forth}_{p,q}, a, a', s_1, s_2, \ldots, s_\pi) := (\text{r-forth}_{p,q+1}, 0, 0, d_1, d_2, \ldots, d_\pi)\ ,\\ &\text{if } \mu_p = \rho \text{ and } \mu_{p+1} = \rho\ ,\end{aligned}$$

where $s_{\mathrm{co}(y_q)} := \mathtt{t}_=$, $d_{\mathrm{co}(y_q)} = 1$, and, for every $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, $d_i := 0$.

Now, for every i, $1 \leq i \leq k'-1$, the transition changing the automaton from the state corresponding to the pair (j_i, μ_i) into the state corresponding to (j_{i+1}, μ_{i+1}) has been defined. Note, that in these transitions we increment the counter $\mathrm{co}(y_q)$ once more without moving the input head to set its value back to 0 again, such that it is ready for the next time it is used. However, it remains to define what happens if the counter $\mathrm{co}(y_{j_{k'}})$ reaches its counter bound in the state that corresponds to the final pair $(j_{k'}, \mu_{k'})$. In this case, the automaton enters a new state match$_p$, in which the factors u_{j_l} and u_{j_r} are matched. In the following definition, let $q := j_{k'}$.

For all $a, a' \in \Sigma$ and for every $s_i \in \{\mathtt{t}_=, \mathtt{t}_<\}$, $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, we define

$$\delta(\text{l-forth}_{p,q}, a, a', s_1, s_2, \ldots, s_\pi) := (\text{match}_p, 0, 0, d_1, d_2, \ldots, d_\pi) \ ,$$
$$\text{if } \mu_{j_{k'}} = \lambda \ ,$$

$$\delta(\text{r-forth}_{p,q}, a, a', s_1, s_2, \ldots, s_\pi) := (\text{match}_p, 0, 0, d_1, d_2, \ldots, d_\pi) \ ,$$
$$\text{if } \mu_{j_{k'}} = \rho \ ,$$

where $s_{\mathrm{co}(y_q)} := \mathtt{t}_=$, $d_{\mathrm{co}(y_q)} := 1$, and, for every $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, $d_i := 0$.

In the state match_p the factors u_{j_l} and u_{j_r} are matched by simultaneously moving both heads to the right. In the following definition, let $q := j_l$. For every $a \in \Sigma$ and for every $s_i \in \{\mathtt{t}_=, \mathtt{t}_<\}$, $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, we define

$$\delta(\text{match}_p, a, a, s_1, s_2, \ldots, s_\pi) := (\text{match}_p, 1, 1, d_1, d_2, \ldots, d_\pi) \ ,$$

where $s_{\mathrm{co}(y_q)} := \mathtt{t}_<$, $d_{\mathrm{co}(y_q)} := 1$, and, for every $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, $d_i := 0$.

Note, that these transitions are only applicable if both input heads scan the same symbol. If the symbol scanned by the left head differs from the one scanned by the right head, then no transition is defined and thus the automaton stops in a non-accepting state.

Finally, the very last transition to define in order to transform D_p into a part of the automaton is the case when counter $\mathrm{co}(y_{j_l})$ has reached its counter bound in state match_p. For the sake of convenience, we assume that the first pair of D_{p+1} is (j', λ) and, furthermore, that $m_{p+1} := (j''_l, j''_r)$ with $j_l < j''_l$. For all $a, a' \in \Sigma$ and for every $s_i \in \{\mathtt{t}_=, \mathtt{t}_<\}$, $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, we define

$$\delta(\text{match}_p, a, a', s_1, s_2, \ldots, s_\pi) := (\text{l-forth}_{p+1,j'}, 0, 0, d_1, d_2, \ldots, d_\pi) \ ,$$

where $s_{\mathrm{co}(y_q)} := \mathtt{t}_=$, $d_{\mathrm{co}(y_q)} := 1$, and, for every $i \in \{1, \ldots, \pi\}/\{\mathrm{co}(y_q)\}$, $d_i := 0$.

As mentioned above, this is merely the transition in the case that the first pair of D_{p+1} is (j', λ) and $j_l < j''_l$ is satisfied. However, all the other cases can be handled analogously. In the case that the first pair of D_{p+1} is (j', ρ) instead of (j', λ) we have to enter state $\text{r-forth}_{p+1,j'}$ instead of $\text{l-forth}_{p+1,j'}$. If $j_l > j''_l$ holds instead of $j_l < j''_l$ we have to enter a back-state (e. g., $\text{l-back}_{p+1,j'}$) instead. These transitions can also be interpreted as the passage between the part of the automaton corresponding to D_p and the part corresponding to the next tuple D_{p+1} of the Janus operating mode.

We have to explain a few special cases concerning the definitions above. Regarding the tuples D_1 and D_k we have to slightly change the definitions. Initially, both heads are located at the very left position of the input, i. e., the left endmarker "¢", therefore only l-forth$_{1,q}$ and r-forth$_{1,q}$ states are needed to transform D_1 into a part of the automaton. When the automaton is in state match$_k$ and the counter has reached its counter bound, then the state q_f is entered, which is the only final state of M. We recall, that $\alpha = y_1 \cdot y_2 \cdot \ldots \cdot y_n$. Whenever the automaton, for a p, $1 \leq p \leq k$, is in a state in $\{$l-forth$_{p,n}$, l-back$_{p,n}$, r-forth$_{p,n}$, r-back$_{p,n}\}$ or in a state match$_p$, where $m_p = (j, n)$, for some j, $j < n$, is a matching position, then this means that a head is moved over the rightmost factor u_n. When the automaton is in such a state for the first time and the counter bound of counter co(y_n) is reached, then the automaton blocks if the head does not scan the right endmarker "\$", as this implies $|u_1 \cdot u_2 \cdot \ldots \cdot u_n| < |w|$. In case that $|u_1 \cdot u_2 \cdot \ldots \cdot u_n| > |w|$ the automaton blocks at some point when it tries to move a head to the right that scans \$ since this transition is not defined. A formal definition of these special cases is omitted.

Obviously, each of the above defined transitions depend on a certain counter determined by the mapping co, so let us now return to the problem of defining this mapping. As already mentioned, this mapping co is in general not injective, hence it is possible that co(x) = co(z) for some $x \neq z$. This means, intuitively speaking, that there seems to be an undesirable connection between the lengths of factors u_j with $y_j = x$ and factors $u_{j'}$ with $y_{j'} = z$. However, this connection does not have any effect if it is possible to, initially, exclusively use the counter bound of counter co(x) = co(z) for factors corresponding to x and then exclusively for factors corresponding to variable z and never for factors corresponding to x again. In this case the automaton may reset this counter after it has been used for factors corresponding to x in order to obtain a new length for factors corresponding to z. This means that a counter is reused. We now formalise this idea.

Let $\overline{\Delta_\alpha} := ((d'_1, \mu'_1), (d'_2, \mu'_2), \ldots, (d'_{k''}, \mu'_{k''}))$ be the head movement indicator of the Janus operating mode. We consider the pattern $D_\alpha := y_{d'_1} \cdot y_{d'_2} \cdot \ldots \cdot y_{d'_{k''}}$. If, for some $x, z \in \text{var}(\alpha)$, $x \neq z$, D_α can be factorised into $D_\alpha = \beta \cdot x \cdot \gamma \cdot z \cdot \gamma' \cdot x \cdot \delta$, then the automaton cannot use the same counter for variables x and z; thus, co has to satisfy co$(x) \neq$ co(z).

Claim (1). There exists a total mapping co : var$(\alpha) \to \{1, 2, \ldots, \pi\}$ such that, for all $x, z \in \text{var}(\alpha)$, $x \neq z$, if $D_\alpha = \beta \cdot x \cdot \gamma \cdot z \cdot \gamma' \cdot x \cdot \delta$ or $D_\alpha = \beta \cdot z \cdot \gamma \cdot x \cdot \gamma' \cdot z \cdot \delta$, then co$(x) \neq$ co(z).

Proof. (*Claim* (1)) If there is no set of variables $\Gamma \subseteq \text{var}(\alpha)$ with $|\Gamma| > \pi$ such that for all $x, z \in \Gamma$, $x \neq z$, $D_\alpha = \beta \cdot x \cdot \gamma \cdot z \cdot \gamma' \cdot x \cdot \delta$ or $D_\alpha = \beta \cdot z \cdot \gamma \cdot x \cdot \gamma' \cdot z \cdot \delta$, then there obviously exists such a mapping co. So we assume to the contrary, that there exists a set of variables Γ, $|\Gamma| = \pi + 1$, with the above given properties. Now we can apply

Lemma 3.11 to the pattern D_α and conclude that there exist a $z' \in \Gamma$ such that D_α can be factorised into $D_\alpha = \beta \cdot z' \cdot \gamma$ with $(\Gamma / \{z'\}) \subseteq (\mathrm{var}(\beta) \cap \mathrm{var}(\gamma))$. This directly implies $\mathrm{cn}(\Delta_\alpha) \geq \pi = \mathrm{cn}(\Delta_\alpha) + 1$, which is a contradiction. $\square$ (*Claim* (1))

This shows that such a mapping co exists and, furthermore, we can note that it is straightforward to effectively construct it.

As already mentioned above, it may be necessary for the automaton to reset counters. More formally, if, for some j, $1 \leq j \leq \pi$, and for some $x, z \in \mathrm{var}(\alpha)$, $x \neq z$, $\mathrm{co}(x) = \mathrm{co}(z) = j$, then this counter j must be reset. We now explain how this is done. By definition of the states and transitions so far, we may interpret states as being related to factors u_q, i. e., for every p, $1 \leq p \leq k$, and every q, $1 \leq q \leq n$, the states in $\{\text{l-forth}_{p,q}, \text{l-back}_{p,q}, \text{r-forth}_{p,q}, \text{r-back}_{p,q}\}$ correspond to factor u_q and state match_p corresponds to both factors u_l and u_r, where $m_p = (l, r)$. For every $x \in \mathrm{var}(\alpha)$, the automaton resets counter $\mathrm{co}(x)$, using the special counter instruction $\mathtt{r}$, immediately after leaving the last state corresponding to a factor u_q with $y_q = x$. In order to define this transition formally, we assume that, for example, $\text{l-forth}_{p,q}$ with $y_q = x$ is that state and $\text{l-forth}_{p,q+1}$ is the subsequent state. For all $a, a' \in \Sigma$ and for every $s_i \in \{\mathtt{t}_=, \mathtt{t}_<\}$, $i \in \{1, \ldots, \pi\} / \{\mathrm{co}(x)\}$, we define

$$\delta(\text{l-forth}_{p,q}, a, a', s_1, s_2, \ldots, s_\pi) = (\text{l-forth}_{p,q+1}, 0, 0, d_1, d_2, \ldots, d_\pi) \ ,$$

where $s_{\mathrm{co}(x)} := \mathtt{t}_=$, $d_{\mathrm{co}(x)} := \mathtt{r}$, and, for every $i \in \{1, \ldots, \pi\} / \{\mathrm{co}(x)\}$, $d_i := 0$.

We recall, that by definition of a Janus automaton, all counter bounds are initially 0, so the automaton must initially reset all π counters. To define this transition formally, let $\text{l-forth}_{1,1}$ be the state corresponding to the first element of D_1. The first transition is defined by

$$\delta(q_0, ¢, ¢, \mathtt{t}_=, \mathtt{t}_=, \ldots, \mathtt{t}_=) = (\text{l-forth}_{1,1}, 0, 0, \mathtt{r}, \mathtt{r}, \ldots, \mathtt{r}) \ ,$$

where q_0 is the initial state of M. This concludes the definition of the automaton and we shall now prove its correctness, i. e., $L(M) = L(\alpha)$.

Let $w \in \Sigma^*$ be an arbitrary input word. From the above given definition, it is obvious that the automaton treats w as a sequence of factors $u_1 \cdot u_2 \cdot \ldots \cdot u_n$. The lengths of these factors u_i, $1 \leq i \leq n$, are determined by the counter bounds guessed during the computation. If $|u_1 \cdot u_2 \cdot \ldots \cdot u_n| \neq |w|$, then the automaton does not accept the input anyway, so we may only consider those cases where suitable counter bounds are guessed that imply $|u_1 \cdot u_2 \cdot \ldots \cdot u_n| = |w|$. Recall the complete matching order $(m_1, m_2, \ldots, m_k)$ with $m_p = (l_p, r_p)$, $1 \leq p \leq k$. By definition, in the states match_p, $1 \leq p \leq k$, the automaton matches factor u_{l_p} and u_{r_p}. If M reaches the accepting state q_f, then, for every p, $1 \leq p \leq k$, $u_{l_p} = u_{r_p}$

and, by applying Lemma 3.7, we conclude that $u_1 \cdot u_2 \cdot \ldots \cdot u_n$ is a characteristic factorisation. Hence, $w \in L(\alpha)$.

On the other hand, let $w' \in L(\alpha)$ be arbitrarily chosen. This implies that we can factorise w' into $w' = u_1 \cdot u_2 \cdot \ldots \cdot u_n$ such that for all j, j', $1 \leq j < j' \leq n$, $y_j = y_{j'}$ implies $u_j = u_{j'}$, i. e., $u_1 \cdot u_2 \cdot \ldots \cdot u_n$ is a characteristic factorisation. By definition, it is possible that the automaton guesses counter bounds such that the input word w' is treated in this factorisation $w' = u_1 \cdot u_2 \cdot \ldots \cdot u_n$, so M accepts w' and thus $w' \in L(M)$. Consequently, $L(M) = L(\alpha)$, which concludes the proof of correctness, and hence the proof of Theorem 3.10. □

We conclude this section by discussing the previous results in a bit more detail. The main technical tool defined in this section is the Janus operating mode. So far, we interpreted Janus operating modes as instructions specifying how two input heads can be used to move over a word given in a certain factorisation in order to check on whether this factorisation is a characteristic one. So, in other words, a Janus operating mode can be seen as representing an algorithm, solving the membership problem for the pattern language given by a certain pattern. Theorem 3.10 formally proves this statement.

A major benefit of this approach is, that from now on we can focus on Janus operating modes rather than on the more involved model of a Janus automaton. More precisely, the previous result shows that the task of finding an optimal Janus automaton for a terminal-free pattern language is equivalent to finding an optimal Janus operating mode for this pattern. Before we investigate this task in the subsequent section, we revise our perspective regarding Janus operating modes. There is no need to consider input words anymore and, thus, in the following we shall investigate properties of patterns and Janus operating modes exclusively. Therefore, we establish a slightly different point of view at Janus operating modes, i. e., we interpret them as describing input head movements over a pattern instead of over a word given in a factorisation:

Remark 3.12. Let $\Delta_\alpha := (D_1, D_2, \ldots, D_k)$ be an arbitrary Janus operating mode for some pattern $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ and let Δ_α be derived from the complete matching order $(m_1, m_2, \ldots, m_k)$. Furthermore, let $\overline{\Delta_\alpha} := ((d'_1, \mu'_1), (d'_2, \mu'_2), \ldots, (d'_{k'}, \mu'_{k'}))$ be the head movement indicator of the canonical Janus operating mode. We can interpret $\overline{\Delta_\alpha}$ as a sequence of input head movements over the pattern α, i. e., after i movements or steps of $\overline{\Delta_\alpha}$, where $1 \leq i \leq k'$, the left input head is located at variable $y_{d'_i}$ if $\mu'_i = \lambda$ or, in case that $\mu'_i = \rho$, the right input head is located at $y_{d'_i}$. So for every i, $1 \leq i \leq k'$, the sequence $\overline{\Delta_\alpha}$ determines the positions of both input heads after the first i movements of $\overline{\Delta_\alpha}$. More precisely, for every i, $1 \leq i \leq k'$, after i steps of $\overline{\Delta_\alpha}$, the positions l_i and r_i of the left head and the right head in α

are given by

$$l_i = \max\{d'_j \mid 1 \leq j \leq i, \mu'_j = \lambda\} \text{ and}$$
$$r_i = \max\{d'_j \mid 1 \leq j \leq i, \mu'_j = \rho\}\,.$$

We note that $\{d'_j \mid 1 \leq j \leq i, \mu'_j = \lambda\} = \emptyset$ is possible, which means that $\mu_j = \rho$, $1 \leq j \leq i$, or, in other words, that so far only the right head has been moved. In this case, we shall say that the left head has not yet entered α and therefore is located at position 0. The situation $\{d'_j \mid 1 \leq j \leq i, \mu'_j = \rho\} = \emptyset$ is interpreted analogously. As already mentioned above, for every i, $1 \leq i \leq k'$, we have either $l_i = d'_i$ or $r_i = d'_i$ (depending on μ_i). Furthermore, for every i, $1 \leq i \leq k'$, it is not possible that both heads are located at position 0.

This special perspective towards Janus operating modes, described in the previous remark, shall play a central role in the proofs for the following results.

3.3.2 Patterns with Restricted Variable Distance

We recall that the variable distance is a parameter of patterns that is defined in Section 2.2.1 (Definition 2.3). In this section, by applying Janus automata, we show that for patterns with a restricted variable distance the membership problem can be solved in polynomial time.

We first note that the problem of computing the variable distance $\mathrm{vd}(\alpha)$ for an arbitrary pattern α is not a difficult one:

Proposition 3.13. *For every terminal-free pattern α, the number $\mathrm{vd}(\alpha)$ can be computed in time $\mathrm{O}(|\alpha| \times |\mathrm{var}(\alpha)|)$.*

Proof. Let α be a terminal-free pattern and let $\mathrm{var}(\alpha) = \{x_1, x_2, \ldots, x_m\}$. It is possible to compute the variable distance of α in the following way. First, we initialise a variable $k := 0$. Now, we move over the pattern from left to right. Whenever a variable x_i is encountered, we carry out the following steps. If this is the first time that x_i is encountered, then we initialise a boolean array A_{x_i} of size $|\mathrm{var}(\alpha)|$ with value 0 in every cell. If, on the other hand, this is not the first time x_i is encountered, then we set $k := \max\{k, |A_{x_i}|\}$, where $|A_{x_i}|$ denotes the number of ones in A_{x_i}. In addition to that and regardless of whether this occurrence of variable x_i is the first one or not, we set cell i of array A_{x_j} to 1, for all j, $1 \leq j \leq m$, $i \neq j$, where such an array exists. It can be easily verified that when we reach the right end of α, then k stores the variable distance of α.

For the above described procedure, in every step, we need to manipulate $|\mathrm{var}(\alpha)|$ arrays and we have to compute $|A_{x_i}|$ for some i, $1 \leq i \leq m$, which can be done in time $O(|\operatorname{var}(\alpha)|)$. Thus, the total runtime of this procedure is $O(|\alpha| \times |\operatorname{var}(\alpha)|)$. $\square$

The following vital result shows that for every possible Janus operating mode for some pattern α, its counter number is at least equal to the variable distance of α. Hence, the variable distance is a lower bound for the counter number of Janus operating modes.

Theorem 3.14. *Let Δ_α be an arbitrary Janus operating mode for a terminal-free pattern α. Then* $\operatorname{cn}(\Delta_\alpha) \geq \operatorname{vd}(\alpha)$.

Proof. Let $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern and let $(m_1, m_2, \ldots, m_k)$ be the complete matching order for α from which $\Delta_\alpha := (D_1, D_2, \ldots, D_k)$ is derived. Furthermore, let $\overline{\Delta_\alpha} := ((d'_1, \mu'_1), (d'_2, \mu'_2), \ldots, (d'_{k'}, \mu'_{k'}))$ be the head movement indicator of the Janus operating mode. This sequence $\overline{\Delta_\alpha}$ contains numbers d'_i, $1 \leq i \leq k'$, that are positions of α, i. e., $1 \leq d'_i \leq |\alpha|$, $1 \leq i \leq k'$. Hence, we can associate a pattern D_α with $\overline{\Delta_\alpha}$ and α in the following way: $D_\alpha := y_{d'_1} \cdot y_{d'_2} \cdot \ldots \cdot y_{d'_{k'}}$. By definition of the variable distance, we know that there exists an $x \in \operatorname{var}(\alpha)$ such that $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$ with $|\gamma|_x = 0$ and $|\operatorname{var}(\gamma)| = \operatorname{vd}(\alpha)$. We assume $\operatorname{vd}(\alpha) \geq 1$ (i. e., $\operatorname{var}(\gamma) \neq \emptyset$), as in the case $\operatorname{vd}(\alpha) = 0$, $\operatorname{cn}(\Delta_\alpha) \geq \operatorname{vd}(\alpha)$ trivially holds.

In the following, let $\Gamma := \operatorname{var}(\gamma) \cup \{x\}$. We shall prove the statement of the theorem by showing that there exists a variable $z \in \Gamma$ such that $D_\alpha = \overline{\beta} \cdot z \cdot \overline{\gamma}$ with $|(\operatorname{var}(\overline{\beta}) \cap \operatorname{var}(\overline{\gamma}))/\{z\}| \geq \operatorname{vd}(\alpha)$, which implies $\operatorname{cn}(\Delta_\alpha) \geq \operatorname{vd}(\alpha)$. To this end, we first prove the following claim:

Claim (1). For all $z, z' \in \Gamma$, $z \neq z'$, we can factorise D_α into $D_\alpha = \widetilde{\beta} \cdot z \cdot \widetilde{\gamma_1} \cdot z' \cdot \widetilde{\gamma_2} \cdot z \cdot \widetilde{\delta}$ or $D_\alpha = \widetilde{\beta} \cdot z' \cdot \widetilde{\gamma_1} \cdot z \cdot \widetilde{\gamma_2} \cdot z' \cdot \widetilde{\delta}$.

Proof. (*Claim* (1)) For arbitrary $z, z' \in \Gamma$, $z \neq z'$, there are two possible cases regarding the positions of the occurrences of z and z' in α. The first case describes the situation that there exists an occurrence of z' (or z) in α such that z (or z', respectively) occurs to the left and to the right of this occurrence. If this is not possible, the occurrences of z and z' are separated, i. e., the rightmost occurrence of z (or z') is to the left of the leftmost occurrence of z' (or z, respectively). More formally, it is possible to factorise α into

$$\alpha = \widehat{\beta} \cdot z \cdot \widehat{\gamma_1} \cdot z' \cdot \widehat{\gamma_2} \cdot z \cdot \widehat{\delta} \tag{3.1}$$

or into

$$\alpha = \beta \cdot x \cdot \widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2} \cdot z' \cdot \widehat{\gamma_3} \cdot x \cdot \delta \tag{3.2}$$

with $|\beta \cdot x \cdot \widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2}|_{z'} = 0$ and $|\widehat{\gamma_2} \cdot z' \cdot \widehat{\gamma_3} \cdot x \cdot \delta|_z = 0$. The two factorisations obtained by changing the roles of z and z' can be handled analogously and are, thus, omitted. We note that in the second factorisation, $\widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2} \cdot z' \cdot \widehat{\gamma_3}$ equals the factor γ from the above introduced factorisation $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$. This is due to the fact that we assume $z, z' \in \Gamma$.

We first observe that $z = x$ or $z' = x$ implies that the first factorisation is possible. If we cannot factorise α according to factorisation (3.1), then we can conclude that the rightmost occurrence of z is to the left of the leftmost occurrence of z' and, furthermore, as both $z, z' \in \Gamma$ and $z \neq x \neq z'$, these occurrences are both in the factor γ. Hence, factorisation (3.2) applies. We now show that in both cases the variables z, z' satisfy the property described in Claim (1). However, throughout the following argumentations, we need to bear in mind that Claim (1) describes a property of D_α and the two considered factorisations are factorisations of α.

We start with the case that α can be factorised into $\alpha = \widehat{\beta} \cdot z \cdot \widehat{\gamma_1} \cdot z' \cdot \widehat{\gamma_2} \cdot z \cdot \widehat{\delta}$. Let $p := |\widehat{\beta} \cdot z \cdot \widehat{\gamma_1} \cdot z' \cdot \widehat{\gamma_2}| + 1$, thus $y_p = z$. In the complete matching order $(m_1, \ldots, m_k)$ there has to be an m_q, $1 \leq q \leq k$, with $m_q := (j_l, j_r)$ and either $j_l = p$ or $j_r = p$. We assume that $j_l = p$; the case $j_r = p$ can be handled analogously. This implies, by the definition of Janus operating modes, that the last element of D_q is (p, λ).

In the following, we interpret the Janus operating mode as a sequence of input head movements over α, as explained in Remark 3.12. Both heads start at the very left position of the input, so in order to move the left head to position p in the pattern, it has to pass the whole part to the left of position p, i. e. $y_1 \cdot y_2 \cdot \ldots y_{p-1}$, from left to right (possibly changing directions several times). In this initial part of the pattern, the variables z and z' occur in exactly this order. We conclude that the left head has to pass an occurrence of z, then pass an occurrence of z' and finally reaches position p, where variable z occurs. Regarding D_α this means that a factorisation $D_\alpha = \widetilde{\beta} \cdot z \cdot \widetilde{\gamma_1} \cdot z' \cdot \widetilde{\gamma_2} \cdot z \cdot \widetilde{\delta}$ is possible.

Next, we consider the case that it is not possible to factorise $\alpha = \widehat{\beta} \cdot z \cdot \widehat{\gamma_1} \cdot z' \cdot \widehat{\gamma_2} \cdot z \cdot \widehat{\delta}$. As explained above, this implies that $\alpha = \beta \cdot x \cdot \widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2} \cdot z' \cdot \widehat{\gamma_3} \cdot x \cdot \delta$ with $|\beta \cdot x \cdot \widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2}|_{z'} = 0$ and $|\widehat{\gamma_2} \cdot z' \cdot \widehat{\gamma_3} \cdot x \cdot \delta|_z = 0$. Let $r_z := |\beta \cdot x \cdot \widehat{\gamma_1}| + 1$ and $l_{z'} := |\beta \cdot x \cdot \widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2}| + 1$ be the positions of the variables z and z' pointed out in the factorisation above. Obviously, r_z is the rightmost occurrence of z and $l_{z'}$ is the leftmost occurrence of z'. These positions r_z and $l_{z'}$ have to be covered by some matching positions in the complete matching order $(m_1, \ldots, m_k)$, i. e., there exist matching positions $m_i := (l_z, r_z)$ and $m_{i'} := (l_{z'}, r_{z'})$. We can assume that r_z is the right element and $l_{z'}$ the left element of a matching position, as these positions describe the rightmost and the leftmost occurrences of the variable z and z', respectively. Moreover, $(m_1, \ldots, m_k)$ has to contain a complete matching order for variable x in α. Since there is no occurrence of x in the factor γ, this implies the existence of a matching position $m_{i''} := (l_x, r_x)$ with $l_x \leq |\beta| + 1$ and $|\beta \cdot x \cdot \widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2} \cdot z' \cdot \widehat{\gamma_3}| + 1 \leq r_x$. We simply assume that $l_x = |\beta| + 1$ and $r_x = |\beta \cdot x \cdot \widehat{\gamma_1} \cdot z \cdot \widehat{\gamma_2} \cdot z' \cdot \widehat{\gamma_3}| + 1$, as this is no loss of generality regarding the following argumentation. Hence, we deal with the following situation (recall that l_x, r_x, r_z and $l_{z'}$ are positions of α):

$$\alpha = \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline \quad\beta\quad & x & \quad\widehat{\gamma_1}\quad & z & \quad\widehat{\gamma_2}\quad & z' & \quad\widehat{\gamma_3}\quad & x & \quad\delta\quad \\ \hline \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} \\ \multicolumn{1}{c}{} & \multicolumn{1}{c}{l_x} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{r_z} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{l_{z'}} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{r_x} & \multicolumn{1}{c}{} \end{array}$$

Now, in the same way as before, we interpret the Janus operating mode as a sequence of input head movements. We proceed by considering two cases concerning the order of the matching positions $m_{i'} = (l_{z'}, r_{z'})$ and $m_{i''} = (l_x, r_x)$ in the complete matching order, i. e., either $i' < i''$ or $i'' < i'$. In the latter case, $i'' < i'$, the right input head is moved from the leftmost variable in α to position r_x, hence, it passes z and z' in this order. Furthermore, the left input head is moved to position l_x. After that, since $i'' < i'$, the left input head has to be moved from position l_x to position $l_{z'}$, thus, passing position r_z where variable z occurs. Hence, we conclude $D_\alpha = \widetilde{\beta} \cdot z \cdot \widetilde{\gamma_1} \cdot z' \cdot \widetilde{\gamma_2} \cdot z \cdot \widetilde{\delta}$. Next, we assume $i' < i''$, so the left input head is moved from the leftmost variable in α to position $l_{z'}$, so again, an input head passes z and z' in this order. After that, the left input head is moved from position $l_{z'}$ to position l_x, thus, it passes variable z on position r_z. Again, we can conclude $D_\alpha = \widetilde{\beta} \cdot z \cdot \widetilde{\gamma_1} \cdot z' \cdot \widetilde{\gamma_2} \cdot z \cdot \widetilde{\delta}$. □ (*Claim* (1))

Hence, for all $z, z' \in \Gamma$, $z \neq z'$, D_α can be factorised into $D_\alpha = \widetilde{\beta} \cdot z \cdot \widetilde{\gamma_1} \cdot z' \cdot \widetilde{\gamma_2} \cdot z \cdot \widetilde{\delta}$ or $D_\alpha = \widetilde{\beta} \cdot z' \cdot \widetilde{\gamma_1} \cdot z \cdot \widetilde{\gamma_2} \cdot z' \cdot \widetilde{\delta}$, and therefore we can apply Lemma 3.11 and conclude that there exists a $z \in \Gamma$ such that D_α can be factorised into $D_\alpha = \overline{\beta} \cdot z \cdot \overline{\gamma}$ with $(\Gamma / \{z\}) \subseteq (\mathrm{var}(\overline{\beta}) \cap \mathrm{var}(\overline{\gamma}))$. This directly implies that $\mathrm{cn}(\Delta_\alpha) \geq |\Gamma| - 1 = \mathrm{vd}(\alpha)$. □

In the previous section, the task of finding an optimal Janus automaton for a pattern is shown to be equivalent to finding an optimal Janus operating mode for this pattern. Now, by the above result, a Janus operating mode Δ_α for some pattern α is optimal if $\mathrm{cn}(\Delta_\alpha) = \mathrm{vd}(\alpha)$ is satisfied. Hence, our next goal is to find a Janus operating mode with that property. To this end, we shall first define a special complete matching order from which the optimal Janus operating mode is then derived.

Definition 3.15. Let $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern with $p := |\mathrm{var}(\alpha)|$. For every $x_i \in \mathrm{var}(\alpha)$, let $\mathrm{varpos}_i(\alpha) := \{j_{i,1}, j_{i,2}, \ldots, j_{i,n_i}\}$ with $n_i := |\alpha|_{x_i}$, $j_{i,l} < j_{i,l+1}$, $1 \leq l \leq n_i - 1$. Let $(m_1, m_2, \ldots, m_k)$, $k = \sum_{i=1}^{p} n_i - 1$, be the enumeration of the set $\{(j_{i,l}, j_{i,l+1}) \mid 1 \leq i \leq p, 1 \leq l \leq n_i - 1\}$ such that, for every i', $1 \leq i' < k$, the left element of the pair $m_{i'}$ is smaller than the left element of $m_{i'+1}$. We call $(m_1, m_2, \ldots, m_k)$ the *canonical matching order for* α.

Proposition 3.16. *Let* α *be a terminal-free pattern. The canonical matching order for* α *is a complete matching order.*

Proof. For every $x_i \in \mathrm{var}(\alpha)$, let $\mathrm{varpos}_i(\alpha) := \{j_{i,1}, j_{i,2}, \ldots, j_{i,n_i}\}$ with $n_i := |\alpha|_{x_i}$, $j_{i,l} < j_{i,l+1}$, $1 \leq l \leq n_i - 1$. The tuple

$$((j_{i,1}, j_{i,2}), (j_{i,2}, j_{i,3}), \ldots, (j_{i,n_i-2}, j_{i,n_i-1}), (j_{i,n_i-1}, j_{i,n_i}))$$

is clearly a matching order for x_i in α. As the canonical matching order contains all these matching orders for each variable $x_i \in \mathrm{var}(\alpha)$, it is a complete matching order for α. □

Intuitively, the canonical matching order can be constructed by simply moving through the pattern from left to right and for each encountered occurrence of a variable x, this occurrence and the next occurrence of x (if there is any) constitutes a matching position. For instance, the canonical matching order for the example pattern β introduced in Section 3.3.1 is $((1,3),(2,4),(4,6),(5,7))$.

We proceed with the definition of a Janus operating mode that is derived from the canonical matching order. Before we do so, we informally explain how this is done. To this end, we employ the interpretation of Janus operating modes as instructions for input head movements. In each step of moving the input heads from one matching position to another, we want to move first the left head completely and then the right head. This is not a problem as long as the part the left head has to be moved over and the part the right head has to be moved over are not overlapping. However, if they are overlapping, then the left head would overtake the right head which conflicts with the definition of Janus operating modes. So in this special case, we first move the left head until it reaches the right head and then we move both heads simultaneously. As soon as the left head reaches the left element of the next matching position, we can keep on moving the right head until it reaches the right element of the next matching position.

Definition 3.17. Let $(m_1, m_2, \ldots, m_k)$ be the canonical matching order for a terminal-free pattern α. For any $m_{i-1} := (j'_1, j'_2)$ and $m_i := (j_1, j_2)$, $2 \leq i \leq k$, let $(p_1, p_2, \ldots, p_{k_1}) := g(j'_1, j_1)$ and $(p'_1, p'_2, \ldots, p'_{k_2}) := g(j'_2, j_2)$, where g is the function introduced in Definition 3.8. If $j_1 \leq j'_2$, then we define

$$D_i := ((p_1, \lambda), (p_2, \lambda), \ldots, (p_{k_1}, \lambda), (p'_1, \rho), (p'_2, \rho), \ldots, (p'_{k_2}, \rho), (j_2, \rho), (j_1, \lambda)) \ .$$

If, on the other hand, $j'_2 < j_1$, we define D_i in three parts

$$\begin{aligned} D_i := (&(p_1, \lambda), (p_2, \lambda), \dots, (j'_2, \lambda), \\ &(j'_2 + 1, \rho), (j'_2 + 1, \lambda), (j'_2 + 2, \rho), (j'_2 + 2, \lambda), \dots, (j_1 - 1, \rho), (j_1 - 1, \lambda), \\ &(j_1, \rho), (j_1 + 1, \rho), \dots, (j_2 - 1, \rho), (j_2, \rho), (j_1, \lambda)) \ . \end{aligned}$$

Finally, $D_1 := ((1, \rho), (2, \rho), \dots, (j - 1, \rho), (j, \rho), (1, \lambda))$, where $m_1 = (1, j)$. The tuple $(D_1, D_2, \dots, D_k)$ is called the *canonical Janus operating mode.*

If we derive a Janus operating mode from the canonical matching order $((1, 3)$, $(2, 4)$, $(4, 6)$, $(5, 7))$ for β as described in Definition 3.17 we obtain the canonical Janus operating mode $(((1, \rho)$, $(2, \rho)$, $(3, \rho)$, $(1, \lambda))$, $((4, \rho)$, $(2, \lambda))$, $((3, \lambda)$, $(5, \rho)$, $(6, \rho)$, $(4, \lambda))$, $((7, \rho)$, $(5, \lambda)))$. This canonical Janus operating mode has a counter number of 1, so its counter number is smaller than the counter number of the example Janus operating mode Δ_β given in Section 3.3.1 and, furthermore, equals the variable distance of β. Referring to Theorem 3.14, we conclude that the canonical Janus operating mode for β is optimal. The next lemma shows that this holds for every pattern.

Lemma 3.18. *Let α be a terminal-free pattern and let Δ_α be the canonical Janus operating mode for α. Then* $\mathrm{cn}(\Delta_\alpha) = \mathrm{vd}(\alpha)$.

Proof. Let $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ and let $\overline{\Delta_\alpha} := ((d'_1, \mu'_1), (d'_2, \mu'_2), \dots, (d'_{k'}, \mu'_{k'}))$ be the head movement indicator of the canonical Janus operating mode. This sequence $\overline{\Delta_\alpha}$ contains numbers d'_i, $1 \leq i \leq k'$, that are positions of α, i. e. $1 \leq d'_i \leq |\alpha|$, $1 \leq i \leq k'$. Hence, we can associate a sequence of variables $(y_{d'_1}, y_{d'_2}, \dots, y_{d'_{k'}})$ with $\overline{\Delta_\alpha}$.

In order to prove Lemma 3.18, we assume to the contrary that $\mathrm{cn}(\Delta_\alpha) > \mathrm{vd}(\alpha)$. This implies that there is a p, $1 \leq p \leq k'$, and a set Γ of at least $\pi := \mathrm{vd}(\alpha) + 1$ different variables $z_1, z_2, \dots, z_\pi$ such that $y_{d'_p} \notin \Gamma$ and, for every $z \in \Gamma$, there exist j, j', $1 \leq j < p < j' \leq k'$, with $y_{d'_j} = y_{d'_{j'}} = z$.

We can interpret $\overline{\Delta_\alpha}$ as a sequence of input head movements over the pattern α as explained in Remark 3.12. We are particularly interested in the position of the *left* head in α at step p of $\overline{\Delta_\alpha}$. Thus, we define $\widehat{p}$ such that $d'_{\widehat{p}} = \max\{d'_j \mid 1 \leq j \leq p, \mu'_j = \lambda\}$. However, we note that $\{d'_j \mid 1 \leq j \leq p, \mu'_j = \lambda\} = \emptyset$ is possible and in this case $d'_{\widehat{p}}$ would be undefined. So for now, we assume that $\{d'_j \mid 1 \leq j \leq p, \mu'_j = \lambda\} \neq \emptyset$ and consider the other case at the end of this proof. Moreover, we need to define the rightmost position in α that has been visited by any input head when we reach step p in $\overline{\Delta_\alpha}$. By definition of the canonical matching order, this has to be the right input head, as it is always positioned to the right of the left input head. Thus, we define $p_{\max}$ such that $d'_{p_{\max}} := \max\{d'_j \mid 1 \leq j \leq p\}$.

Now, we can consider α in the factorisation

$$\alpha = \beta \cdot y_{d'_{\widehat{p}}} \cdot \gamma \cdot y_{d'_{p_{\max}}} \cdot \delta \ .$$

By the definition of the positions $\widehat{p}$ and $p_{\max}$ above, we can conclude the following. After performing all steps d'_j with $1 \leq j \leq p$, position $d'_{\widehat{p}}$ is the position where the left head is located right now. This implies, by definition of the canonical Janus operating mode, that no head will be moved to one of the positions in β again. The position $d'_{p_{\max}}$ is the rightmost position visited by any head so far. Hence, until now, no head has reached a position in δ.

Regarding the sequence of variables $(y_{d'_1}, y_{d'_2}, \ldots, y_{d'_{k'}})$ we can observe that for every j, $1 \leq j \leq p$, $y_{d'_j} \in \text{var}(\beta \cdot y_{d'_{\widehat{p}}} \cdot \gamma \cdot y_{d'_{p_{\max}}})$, and, for every j', $p < j' \leq k'$, $y_{d'_{j'}} \in \text{var}(\gamma \cdot y_{d'_{p_{\max}}} \cdot \delta)$. This follows directly from our interpretation of $\overline{\Delta_\alpha}$ as a sequence of input head movements over α. Moreover, since for every $z \in \Gamma$, there exist j, j', $1 \leq j < p < j' \leq k'$, with $y_{d'_j} = y_{d'_{j'}} = z$, we can conclude that $\Gamma \subseteq (\text{var}(\beta \cdot y_{d'_{\widehat{p}}} \cdot \gamma \cdot y_{d'_{p_{\max}}}) \cap \text{var}(\gamma \cdot y_{d'_{p_{\max}}} \cdot \delta))$. We can further show that $\Gamma \subseteq \text{var}(\gamma \cdot y_{d'_{p_{\max}}})$. To this end, we assume that for some $z \in \Gamma$, $z \notin \text{var}(\gamma \cdot y_{d'_{p_{\max}}})$, which implies $z \in (\text{var}(\beta \cdot y_{d_{\widehat{p}}}) \cap \text{var}(\delta))$. Hence, we can conclude that there exists a matching position (l_z, r_z) in the canonical matching order, where the left element l_z is a position in $\beta \cdot y_{d_{\widehat{p}}}$ and the right element r_z is a position in δ, i. e., $1 \leq l_z \leq |\beta \cdot y_{d_{\widehat{p}}}|$ and $|\beta \cdot y_{d'_{\widehat{p}}} \cdot \gamma \cdot y_{d'_{p_{\max}}}| + 1 \leq r_z \leq |\alpha|$. By definition of the canonical Janus operating mode, this implies that the rightmost position in α, that has been visited by any input head when we reached step p in $\overline{\Delta_\alpha}$ has to be at least position r_z. Since $r_z > d'_{p_{\max}}$, this is clearly a contradiction. Consequently, we conclude that $\Gamma \subseteq \text{var}(\gamma \cdot y_{d'_{p_{\max}}})$.

We recall that position $d'_{p_{\max}}$ of α has already been reached by the right head and that in the canonical Janus operating mode, the right head is exclusively moved from the right element of some matching position (l, r) to the right element of another matching position (l', r'). Consequently, either $r \leq d'_{p_{\max}} \leq r'$ or $r' \leq d'_{p_{\max}} \leq r$ and, furthermore, the left elements l and l' must be positions in the factor $\beta \cdot y_{d'_{\widehat{p}}}$. Thus, there has to be a matching position (l, r) in the canonical matching order with $l \leq d'_{\widehat{p}}$ and $r \geq d'_{p_{\max}}$. Therefore, we can refine the factorisation from above by factorising $\beta \cdot y_{d'_{\widehat{p}}}$ into $\beta_1 \cdot y_l \cdot \beta_2$ and $y_{d'_{p_{\max}}} \cdot \delta$ into $\delta_1 \cdot y_r \cdot \delta_2$; thus, we obtain

$$\alpha = \beta_1 \cdot y_l \cdot \beta_2 \cdot \gamma \cdot \delta_1 \cdot y_r \cdot \delta_2 \ .$$

In the following, we show that the factor between the left and right element of the matching position (l, r), i. e., $\beta_2 \cdot \gamma \cdot \delta_1$, contains too many distinct variables different from $y_l = y_r$. More precisely, the number of such variables is clearly bounded by the variable distance, but, by means of the variables in Γ, we obtain a contradiction by showing that there are $\mathrm{vd}(\alpha) + 1$ such variables in the factor $\beta_2 \cdot \gamma \cdot \delta_1$. To this end, we first recall that we have already established that $\Gamma \subseteq \mathrm{var}(\gamma \cdot y_{d'_{p_{\max}}})$ and, furthermore, $y_{d'_p} \notin \Gamma$ and (l, r) is a matching position; thus, $y_l = y_r$.

By the factorisation above, we know that $d'_{p_{\max}} \leq r$. If $d'_{p_{\max}} < r$, then $\Gamma \subseteq \mathrm{var}(\gamma \cdot y_{d'_{p_{\max}}})$ implies $\Gamma \subseteq \mathrm{var}(\gamma \cdot \delta_1)$. We can further note, that y_r cannot be an element of Γ as this contradicts to the fact that (l, r) is a matching position. Thus, we have $|\Gamma|$ variables different from $y_l = y_r$ occurring in $\beta_2 \cdot \gamma \cdot \delta_1$ and we obtain the contradiction as described above.

In the following, we assume that $d'_{p_{\max}} = r$ and note that this implies $\delta_1 = \varepsilon$. We observe that there are two cases depending on whether or not $y_{d'_{p_{\max}}} \in \Gamma$. We start with the easy case, namely $y_{d'_{p_{\max}}} \notin \Gamma$, and note that in this case $\Gamma \subseteq \mathrm{var}(\gamma \cdot y_{d'_{p_{\max}}})$ implies $\Gamma \subseteq \mathrm{var}(\gamma)$. In the same way as before, this leads to a contradiction.

It remains to consider the case that $y_{d'_{p_{\max}}} \in \Gamma$. Here, $\Gamma \subseteq \mathrm{var}(\gamma)$ is not satisfied anymore, as $(l, d'_{p_{\max}})$ is a matching position (recall that we still assume $d'_{p_{\max}} = r$) and, thus, $y_{d'_{p_{\max}}} \notin \mathrm{var}(\gamma)$. In the following we consider the variable $y_{d'_p}$, for which, by definition, $y_{d'_p} \notin \Gamma$ is satisfied. Hence, in order to obtain a contradiction, it is sufficient to show that $y_{d'_p} \in \mathrm{var}(\beta_2 \cdot \gamma \cdot \delta_1)$. To this end, we need the following claim:

Claim (1). $l \leq d'_p$.

Proof. (*Claim* (1)) If $\mu'_p = \lambda$, then, by definition, $d'_{\widehat{p}} = d'_p$ and if $\mu'_p = \rho$, then $d'_{\widehat{p}} < d'_p$, since $\widehat{p}$ is the position of the left head and d'_p is the position of the right head. Hence, since $l \leq d'_{\widehat{p}}$, we conclude $l \leq d'_{\widehat{p}} \leq d'_p$. □ (*Claim* (1))

If $l < d'_p$, then $y_{d'_p} \in \mathrm{var}(\beta_2 \cdot \gamma \cdot \delta_1)$, since $y_{d'_p} = y_{d'_{p_{\max}}}$ is not possible as, by assumption, $y_{d'_{p_{\max}}} \in \Gamma$ and $y_{d'_p} \notin \Gamma$. Hence, we assume $l = d'_p$, which implies $y_l = y_{d'_p}$. We can show that this is a contradiction. First, we recall that $(l, d'_{p_{\max}})$ is a matching position, so $y_l = y_{d'_{p_{\max}}}$ and since $y_{d'_{p_{\max}}} \in \Gamma$, $y_l \in \Gamma$ as well. Furthermore, $y_{d'_p} \notin \Gamma$, which contradicts $y_l = y_{d'_p}$. We conclude that $y_{d'_p} \in \mathrm{var}(\beta_2 \cdot \gamma \cdot \delta_1)$ must be satisfied.

Hence, for each possible case, we obtain $|\mathrm{var}(\beta_2 \cdot \gamma \cdot \delta_1)| \geq \pi$, which is a contradiction.

It still remains to consider the case $\{d'_j \mid 1 \leq j \leq p, \mu'_j = \lambda\} = \emptyset$. In this case we have $\mu'_i = \rho$ for every i with $1 \leq i \leq p$. This implies that until now the left input head has not yet entered α and the right head has been moved directly from the first position of α to position d'_p without reversing direction. Furthermore, we know that the first matching position of the canonical matching order is $(1, r)$, where $d'_p \leq r$.

If $d'_p = r$, we can factorise α into

$$\alpha = y_1 \cdot \beta \cdot y_{d'_p} \cdot \gamma \,,$$

where $(1, d'_p)$ is a matching position. As for every $z \in \Gamma$ there exists an i, $1 \leq i < p$, with $y_{d_i} = z$ and since $y_{d'_p} \notin \Gamma$, we conclude $\Gamma \subseteq \text{var}(\beta)$. This directly implies $\text{vd}(\alpha) \geq \pi$, which is a contradiction.

If, on the other hand, $d'_p < r$, then we can factorise α into

$$\alpha = y_1 \cdot \beta_1 \cdot y_{d'_p} \cdot \beta_2 \cdot y_r \cdot \gamma \,.$$

In the same way as before, we can conclude that $\Gamma \subseteq \text{var}(y_1 \cdot \beta_1)$, thus, $(\Gamma / \{y_1\}) \subseteq \text{var}(\beta_1)$. Now, as $y_{d'_p} \notin \Gamma$, we have $(\Gamma / \{y_1\}) \cup \{y_{d'_p}\} \subseteq \text{var}(\beta_1 \cdot y_{d'_p} \cdot \beta_2)$, where $|(\Gamma / \{y_1\}) \cup \{y_{d'_p}\}| = \pi$ and, since $(1, r)$ is a matching position, $\text{vd}(\alpha) \geq \pi$ follows, which is a contradiction. This concludes the proof of Lemma 3.18. □

The above lemma, in conjunction with Theorems 3.10 and 3.14, shows that the canonical Janus operating mode for a pattern α can be transformed into a Janus automaton that is optimal with respect to the number of counters. We summarise this first main result in the following theorem:

Theorem 3.19. *Let α be a terminal-free pattern. There exists a* $\text{JFA}(\text{vd}(\alpha) + 1)$ *M such that $L(M) = L(\alpha)$.*

The Janus automaton obtained from the canonical Janus operating mode for a pattern α (in the way it is done in the proof of Theorem 3.10) is called the *canonical Janus automaton.* As already stated above, Theorem 3.19 shows the optimality of the canonical automaton. However, this optimality is subject to a vital assumption: we assume that the automaton needs to know the length of a factor in order to move an input head over this factor. Although this assumption is quite natural, we shall reconsider it in more detail in Section 3.3.3.

The variable distance is the crucial parameter when constructing canonical Janus automata for pattern languages. We obtain a polynomial time match test for any class of patterns with a restricted variable distance:

Theorem 3.20. *There is a computable function that, given any terminal-free pattern α and $w \in \Sigma^*$, decides on whether $w \in L(\alpha)$ in time* $\text{O}(|\alpha|^3 \, |w|^{(\text{vd}(\alpha)+4)})$.

Proof. We present an algorithm solving the membership problem with respect to terminal-free pattern languages within the time bound claimed in Theorem 3.20.

Our algorithm, on input α and w, simply constructs the canonical Janus automaton M for α and then solves the acceptance problem for M on input w. As $L(M) = L(\alpha)$, this algorithm clearly works correctly.

Regarding the time complexity we have to investigate two aspects: Firstly, the time complexity of transforming α into the canonical Janus automaton M and, secondly, the time complexity of solving the acceptance problem for M on input w. In order to simplify the estimations of time complexities, we define $n := |w|$. In the strict sense, the input has length $|w| + 2$ and there are $|w| + 1$ possible counter bounds to guess, but as we shall use the Landau notation, n is sufficiently accurate for the following analysis.

We begin with transforming $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_{n'}$ into M. To this end, we construct the canonical matching order $(m_1, m_2, \ldots, m_k)$, which can be obtained from α in time $\mathrm{O}(|\alpha|)$. Definition 3.17 shows that the canonical Janus operating mode $\Delta_\alpha := (D_1, \ldots, D_k)$ can be directly constructed from the canonical matching order and the time complexity required to do so is merely the size of Δ_α. Obviously, every D_i, $1 \leq i \leq k$, has $\mathrm{O}(|\alpha|)$ elements and $k \leq |\alpha|$. Thus, we conclude that Δ_α can be constructed in $\mathrm{O}(|\alpha|^2)$. Let $\overline{\Delta_\alpha} = ((d'_1, \mu'_1), (d'_2, \mu'_2), \ldots, (d'_{k'}, \mu'_{k'}))$ be the head movement indicator of Δ_α, and let $D_\alpha := y_{d'_1} \cdot y_{d'_2} \cdot \ldots \cdot y_{d'_{k'}}$, where, as described above, $k' \leq |\alpha|^2$. Next, we have to construct a mapping $\mathrm{co} : \mathrm{var}(\alpha) \to \{1, \ldots, \mathrm{vd}(\alpha) + 1\}$ with the required properties described in the proof of Theorem 3.10, i. e., if, for some $z, z' \in \mathrm{var}(\alpha)$, $z \neq z'$, D_α can be factorised into $D_\alpha = \beta \cdot z \cdot \gamma \cdot z' \cdot \gamma' \cdot z \cdot \delta$, then $\mathrm{co}(z) \neq \mathrm{co}(z')$. Such a mapping can be constructed in the following way. Assume that it is possible to mark counters either as free or as occupied. We move over the pattern $y_{d_1} \cdot y_{d_2} \cdot \ldots \cdot y_{d_{k'}}$ from left to right and whenever a variable x_i is encountered for the first time, we set $\mathrm{co}(x_i) := j$ for some counter j that is not occupied right now and then mark this counter j as occupied. Whenever a variable x_i is encountered for the last time, counter $\mathrm{co}(x_i)$ is marked as free. As we have to move over $\overline{\Delta_\alpha}$ in order to construct co in this way, time $\mathrm{O}(k') = \mathrm{O}(|\alpha|^2)$ is sufficient. We note that this method can be applied as it is not possible that there are more than $\mathrm{cn}(\Delta_\alpha) + 1 = \mathrm{vd}(\alpha) + 1$ variables such that for all z, z', $z \neq z'$ of them, D_α can be factorised into $D_\alpha = \beta \cdot z \cdot \gamma \cdot z' \cdot \gamma' \cdot z \cdot \delta$ or $D_\alpha = \beta \cdot z' \cdot \gamma \cdot z \cdot \gamma' \cdot z' \cdot \delta$. This can be shown in the same way as we have already done in the proof of Theorem 3.10.

Next we transform each D_p, $1 \leq p \leq k$, into a part of the automaton M, following the construction in the proof of Theorem 3.10. For the remainder of this proof, we define $\pi := \mathrm{vd}(\alpha) + 1$. We show how many states are needed to implement an arbitrary D_p with $p \geq 2$. Therefore, we define

$$D_p := ((j_1, \mu_1), (j_2, \mu_2), \ldots, (j_{k''}, \mu_{k''}), (j_r, \rho), (j_l, \lambda))$$

with $\mu_q \in \{\lambda, \rho\}$, $1 \leq q \leq k''$, and the tuples (j'_r, ρ), (j'_l, λ) to be the last two elements of D_{p-1}. We need the following sets of states.

$$Q_{p,l} := \begin{cases} \{\text{l-forth}_{p,q} \mid 1 \leq q \leq k'', \mu_q = \lambda\} & \text{if } j'_l < j_l \ , \\ \{\text{l-back}_{p,q} \mid 1 \leq q \leq k'', \mu_q = \lambda\} & \text{else} \ . \end{cases}$$

$$Q_{p,r} := \begin{cases} \{\text{r-forth}_{p,q} \mid 1 \leq q \leq k'', \mu_q = \rho\} & \text{if } j'_r < j_r \ , \\ \{\text{r-back}_{p,q} \mid 1 \leq q \leq k'', \mu_q = \rho\} & \text{else} \ . \end{cases}$$

$$Q_p := Q_{p,l} \cup Q_{p,r} \cup \{\text{match}_p\} \ .$$

The set Q_1 is defined analogously, with the only difference that only forth-states are needed. Clearly, $|Q_p| = k''+1 = \mathrm{O}(|\alpha|)$, $1 \leq p \leq k$. So as $k = \sum_{i=1}^{|\operatorname{var}(\alpha)|}(|\alpha|_{x_i}-1) = |\alpha| - |\operatorname{var}(\alpha)| \leq |\alpha|$, we can conclude that $|Q| = \mathrm{O}(|\alpha|^2)$, where $Q := \bigcup_{i=1}^{k} Q_i$. For each element y in $(|Q| \times \{0, 1, \ldots, n+1\}^2 \times \{\mathtt{t}_=, \mathtt{t}_<\}^\pi)$ we need to define $\delta(y)$, so δ can be constructed in time $\mathrm{O}(|\alpha|^2\, n^2\, 2^\pi)$. This shows that the automaton M can be constructed in time $\mathrm{O}(|\alpha|^2\, n^2\, 2^\pi)$.

Next we shall investigate the time complexity of solving the acceptance problem for M on input w. We apply the following idea. We construct a directed graph of possible configurations of M as vertices, connected by an edge if and only if it is possible to get from one configuration to the other by applying the transition function δ. Then we search this graph for a path leading from the initial configuration to a final configuration, i. e., an accepting path. For an arbitrary vertex v, we denote the number of edges starting at v by *outdegree of* v and the number of edges ending at v by *indegree of* v. The nondeterminism of the computation of M is represented by the fact that there are vertices with outdegree greater than 1, namely those configurations where a new counter bound is guessed. So the existence of an accepting path is a sufficient and necessary criterion for the acceptance of the input word w. Searching this graph for an accepting path leads to a *deterministic* algorithm correctly solving the acceptance problem for M. Let (V, E) be this graph. The problem of finding an accepting path can then be solved in time $\mathrm{O}(|V| + |E|)$. We illustrate this idea more formally and define the set of vertices, i. e., the set of all possible configurations of M on input w:

$$\widehat{C}'_{M,w} := \{(q, h_1, h_2, (c_1, C_1), \ldots, (c_\pi, C_\pi)) \mid q \in Q, 0 \leq h_1 \leq h_2 \leq n+1, \\ 0 \leq c_i \leq C_i \leq n, 1 \leq i \leq \pi\} \ .$$

Now we obtain $\widehat{C}_{M,w}$ by simply deleting all the configurations of $\widehat{C}'_{M,w}$ that cannot be reached in any computation of M on input w. How this can be done shall be explained at the end of the proof. Furthermore, we define a set of edges

$\widehat{E}_{M,w}$, connecting the configurations in $\widehat{C}_{M,w}$ as follows: for all $\widehat{c}_1, \widehat{c}_2 \in \widehat{C}_{M,w}$, $(\widehat{c}_1, \widehat{c}_2) \in \widehat{E}_{M,w}$ if and only if $\widehat{c}_1 \vdash_{M,w} \widehat{c}_2$. We call $\widehat{G}_{M,w} := (\widehat{C}_{M,w}, \widehat{E}_{M,w})$ the *full computation graph of M on input w*. To analyse the time complexity of searching $\widehat{G}_{M,w}$ for an accepting path, we have to determine the size of $\widehat{C}_{M,w}$ and $\widehat{E}_{M,w}$. By the construction given in the proof of Theorem 3.10, for all configurations $(q, h_1, h_2, (c_1, C_1), \ldots, (c_\pi, C_\pi)) \in \widehat{C}_{M,w}$, there is at most one i, $1 \leq i \leq \pi$, with $c_i \geq 1$. That is due to the fact that when M increments a counter, then this counter is incremented until the counter value jumps back to 0 again before another counter is incremented. Thus, for each i, $1 \leq i \leq \pi$, there are $|Q|\, n^{\pi+3}$, possible configurations $(q, h_1, h_2, (c_1, C_1), \ldots, (c_\pi, C_\pi))$ such that $c_i \geq 1$. Therefore, we obtain

$$|\widehat{C}_{M,w}| = \mathrm{O}(|Q|\, \pi\, n^{\pi+3}) = \mathrm{O}(|\alpha|^2\, (\mathrm{vd}(\alpha) + 1)\, n^{\pi+3}) = \mathrm{O}(|\alpha|^3\, n^{\pi+3}) \ .$$

Next, we analyse the number of edges in $\widehat{G}_{M,w}$. As already mentioned, due to the nondeterminism of Janus automata, there are vertices in $\widehat{G}_{M,w}$ with an outdegree greater than one. One such vertex is the initial configuration, as in the initial configuration, all π counters are reset. Thus, the initial configuration has outdegree of $\mathrm{O}(n^\pi)$. Furthermore, if M resets a counter by changing from one configuration $\widehat{c}_1$ to another configuration $\widehat{c}_2$, then $\widehat{c}_1$ has outdegree greater than one. However, there is at most one counter reset by changing from one configuration to another, so, for these configurations, the outdegree is bounded by n. We know that M has $|\mathrm{var}(\alpha)|$ states such that a counter is reset in this state and, furthermore, if a counter is reset, all counter values are 0. Hence the number of configurations with outdegree n is $\mathrm{O}(|\mathrm{var}(\alpha)|\, n^{\pi+2})$ and so we count $\mathrm{O}(|\mathrm{var}(\alpha)|\, n^{\pi+3})$ edges for these configurations. Finally, all the other vertices not considered so far have outdegree 1, and, as the complete number of vertices is $\mathrm{O}(|\alpha|^3\, n^{\pi+3})$, we can conclude that the number of vertices with outdegree 1 does not exceed $\mathrm{O}(|\alpha|^3\, n^{\pi+3})$. We obtain

$$|\widehat{E}_{M,w}| = \mathrm{O}(n^\pi + |\mathrm{var}(\alpha)|\, n^{\pi+3} + |\alpha|^3\, n^{\pi+3}) = \mathrm{O}(|\alpha|^3\, n^{\pi+3}) \ .$$

Consequently, $\mathrm{O}(|\widehat{C}_{M,w}| + |\widehat{E}_{M,w}|) = \mathrm{O}(|\alpha|^3\, n^{\pi+3})$ and, as $\pi = \mathrm{vd}(\alpha) + 1$, $\mathrm{O}(|\widehat{C}_{M,w}| + |\widehat{E}_{M,w}|) = \mathrm{O}(|\alpha|^3\, n^{\mathrm{vd}(\alpha)+4})$. However, it remains to explain how exactly we can search the graph for an accepting path. This can be done in the following way. We start with the initial configuration of M on input w and then we construct the graph $\widehat{G}_{M,w}$ step by step by using a Depth-First-Search approach. By this method an accepting configuration is found if there exists one and, furthermore, we do not need to construct the whole set of configurations $\widehat{C}'_{M,w}$ first. This concludes the proof. □

This main result, which is stated for terminal-free E-pattern languages, also holds for the NE case and for general patterns with terminals (see our example in Section 3.2.1). Moreover, the above developed techniques and results can be easily extended to *regular-typed* patterns[1], i. e., patterns, where for every variable $x \in \text{var}(\alpha)$ a regular language (or *type*) R_x is given and the corresponding pattern language contains then all words w that can be obtained from α by substituting every $x \in \text{var}(\alpha)$ by some word in its type R_x. This is due to the fact that any Janus automaton M that recognises a pattern language given by some pattern α can be easily transformed into a Janus automaton M' that recognises a regular-typed pattern language with respect to α in the following way. Whenever M compares two factors that correspond to occurrences of the same variable x, then M' also checks whether or not these factors are members of the type R_x. This can be easily done by using the finite state control of M'. We note that, technically, regular typed patterns constitute a subclass of extended regular expressions (see Section 7).

Since our automaton-based approach to the membership problem for pattern languages can be easily extended in the above described way and since automata provide a convenient foundation for practical implementations, the results presented in this chapter might have practical implications. We anticipate, though, that the necessary amendments to our definitions involve some technical hassle.

3.3.3 Further Improvements

In Section 2.2.1, the variable distance is introduced and the studies of the present chapter reveal that the complexity of the membership problem is essentially determined by this subtle combinatorial property. Any restriction of this parameter yields major classes of pattern languages with a polynomial-time match test.

We are also able to prove our approach to be optimal. However, this optimality is subject to the following vital assumption. We assume that a Janus automaton needs to know the length of a factor in order to move an input head over this factor and, thus, needs to store this length in form of a counter bound. Although this assumption is quite natural, it might be worthwhile to consider possibilities to weaken it. For instance, a Janus automaton is able to detect the left and right end of its input by means of the endmarkers. Therefore, it can move an input head from any position to either end of the input without using any counter. So if an input head has to be moved from one position to another, there are three ways of doing this. We can either move it directly over the intermediate factors (how it is done in the original definition of Janus operating modes) or we can move it first

[1]In Chapter 7, we shall investigate typed patterns in more detail.

to either the left or the right endmarker and then from there to the new position. In the latter two cases, only the information of the lengths of the factors between the left endmarker or the right endmarker and the target position are required. It is straightforward to extend the definition of Janus operating modes in accordance with these new ideas. Furthermore, we could again use the concept of the counter number of Janus operating modes and transform these refined Janus operating modes into Janus automata in a similar way as done in the proof of Theorem 3.10. The following example points out that, using this new approach, we can find Janus automata with less counters than the canonical Janus automata.

Example 3.21. Let $\alpha := x_1\, x_2\, x_3\, x_1\, x_2\, x_4\, x_4\, x_5\, x_5\, x_3$. Clearly, $\mathrm{vd}(\alpha) = 4$; thus the canonical Janus automaton for α needs 5 counters. We observe that there exists a JFA(4) M with $L(M) = L(\alpha)$. This automaton M matches factors according to the complete matching order $((1,4),(2,5),(6,7),(8,9),(3,10))$. The trick is that after matching the factors related to the matching position $(6,7)$, i. e., the factors corresponding to the occurrences of x_4, the counter responsible for factors corresponding to x_4 is reused to match the factors related to the matching position $(8,9)$. Hence, so far, we only needed 4 counters, but, obviously, we have lost the information of the length of factors corresponding to x_4. Now, we find the situation that it still remains to match the factors corresponding to the occurrences of x_3, i. e. the matching position $(3,10)$, but we cannot simply move the left head back to factor 3, as the automaton does not know the length of the factors corresponding to x_4 anymore. However, we can move it to the left endmarker first, and then from there, over the factors corresponding to x_1 and x_2, to factor 3. We can do this without storing the lengths of factors related to x_4 and x_5. Hence, 4 counters are sufficient.

The above illustrated amendments to our approach further complicate the definition of Janus operating modes and we do not know anymore how to efficiently compute the Janus operating mode that is optimal with respect to the counter number. In the following, we discuss this issue in a bit more detail.

By Theorem 3.14 and Lemma 3.18, it is demonstrated that a Janus operating mode that is optimal with respect to the counter number, namely the canonical Janus operating mode, has a simple structure. Thus, it can be easily computed. However, if we modify the definition of Janus operating modes such that it caters for situations as described in the above example, then we encounter several problems. First of all, we have to find an appropriate matching order from which the optimal Janus operating mode can be derived. But even if we simply assume that we are given such an optimal matching order, then there are still two problems. Firstly, for every element of the matching order and every input head, we have to choose one of three options to move the input head to the next matching position,

i. e., we can move the input head directly, via the left endmarker or via the right endmarker. Secondly, if we have decided on the movements of the input heads, it is still a question how we should interleave these two individual movements of the input heads. More precisely, it is not clear whether it is better to first move the left head and then the right head or the other way around or whether a more sophisticated strategy of alternately moving the two heads is required. This interleaving of input heads movements is not an issue for the canonical Janus operating mode, since here the left input head is monotonously moved step by step to the right. Thus, it is clear that moving the left input head first is always the best choice with respect to the counter number.

We can observe that this last aspect, i. e., the interleaving of the input head movements, can be treated as a separate computational problem. Once we have decided on a matching order and a way to move input heads from one matching position to the next one, the whole problem of computing an optimal Janus operating mode reduces to the problem of interleaving, or shuffling, two sequences in such a way that a certain parameter is optimised. This parameter is the counter number of Janus operating modes, which, applied to general words, is analogous to the scope coincidence degree (see Definition 2.4, Section 2.2.1).

In the following section, we investigate the problem of computing shuffle words with a minimum scope coincidence degree.

3.4 Computing Shuffle Words with Minimum Scope Coincidence Degree

As described in Section 3.3.3, the computational problem to be investigated in this section is motivated by ideas on improving the Janus automata based approach to the membership problem presented in Section 3.3. Nevertheless, a more general motivation for computing shuffle words with a minimum scope coincidence degree can be found in terms of a scheduling problem. In the following, we therefore explain and motivate this computational problem in a way that is independent from our considerations about the Janus operating mode.

Let us assume we have k processes and m values stored in memory cells, and all these processes need to access the stored values at some points during their execution. A process does not necessarily need all the m values at the same time, so a process might get along with less than m memory cells by, for example, first using a memory cell for a value x and then, as soon as x is not needed anymore, using the same cell for another, and previously unneeded, value y. As an example, we assume that process w_1 uses the values $\mathtt{a}, \mathtt{b}$ and $\mathtt{c}$ in the order $\mathtt{abacbc}$. This process only needs two memory cells: In the first cell, b is permanently stored, and the second cell first stores a until it is not required anymore and then stores value c. This is possible, since the part of w_1 where a occurs and the part where c occurs can be completely separated from each other. If we now assume that the k processes cannot access the shared memory simultaneously, then the question arises how we can sequentially arrange all memory accesses such that a minimum overall number of memory cells is required. For example, if we assume that, in addition to process $w_1 = \mathtt{abacbc}$, there is another process $w_2 := \mathtt{abc}$, then we can of course first execute w_1 and afterwards w_2, which results in the memory access sequence abacbcabc. It is easy to see that this requires a memory cell for each value $\mathtt{a}, \mathtt{b}$ and c. On the other hand, we can first execute aba of process w_1, then process $w_2 = \mathtt{abc}$, and finally the remaining part cbc of w_1. This results in abaabccbc, which allows us to use a single memory cell for both values a and c as before.

This scheduling problem can directly be formalised as a question on shuffle words. To this end, we merely have to interpret each of the k processes as a word over an alphabet of cardinality m, where m is the number of different values to be stored. Hence, our problem of finding the best way to organise the memory accesses of all processes directly translates into computing a shuffle word of the k processes that minimises the parameter determining the number of memory cells required. Unfortunately, even for $k = 2$, there is an exponential number of possible ways to schedule the memory accesses. However, we can present an algorithm solving this problem for arbitrary input words and a fixed alphabet size in polynomial time.

The above described problem is similar to the task of *register allocation* (see, e. g., [60, 34]), which plays an important role in compiler optimisation. However, in register allocation, the problem is to allocate a number of m values accessed by a process to a fixed number of k registers, where $k < m$, with the possibility to temporarily move values from a register into the main memory. Since accessing the main memory is a much more expensive CPU operation, the optimisation objective is to find an allocation such that the number of memory accesses is minimised. The main differences to the problem investigated in this work are that the number of registers is fixed, the periods during which the values must be accessible in registers can be arbitrarily changed by storing them in the main memory, and there is usually not the problem of sequentialising several processes.

Furthermore, this problem of computing shuffle words with minimum scope coincidence degree is not covered by any literature on scheduling (see, e. g., [15, 27]) we are aware of, and the same holds for the research on the related *common supersequence problems* (see, e. g., [49]).

3.4.1 The Problem of Computing Shuffle Words with Minimum Scope Coincidence Degree

The scope coincidence degree is introduced for patterns in Section 2.2.1 (Definition 2.4); it is straightforward to generalise it to arbitrary words. However, for the sake of completeness, we shall now explicitly define this parameter in a slightly different, yet equivalent, way.

For an arbitrary $w \in \Sigma^*$ and any $b \in \text{alph}(w)$ let l, r, $1 \leq l, r \leq |w|$, be chosen such that $w[l] = w[r] = b$ and there exists no k, $k < l$, with $w[k] = b$ and no k', $r < k'$, with $w[k'] = b$. Then the *scope of b in w* ($\text{sc}_w(b)$ for short) is defined by $\text{sc}_w(b) := (l, r)$. Note that in the case that for some word w we have $w[j] = b$ and $|w|_b = 1$, the scope of b in w is (j, j). Now we are ready to define the so called *scope coincidence degree*: Let $w \in \Sigma^*$ be an arbitrary word and, for each i, $1 \leq i \leq |w|$, let

$$\text{scd}_i(w) := |\{b \in \Sigma \mid b \neq w[i], \text{sc}_w(b) = (l, r) \text{ and } l < i < r\}| \ .$$

We call $\text{scd}_i(w)$ the scope coincidence degree of position i in w. Furthermore, the scope coincidence degree of the word w is defined by

$$\text{scd}(w) := \max\{\text{scd}_i(w) \mid 1 \leq i \leq |w|\} \ .$$

As an example, we now consider the word $w :=$ `acacbbdeabcedefdeff`. It can easily be verified that $\text{scd}_8(w) = \text{scd}_9(w) = 4$ and $\text{scd}_i(w) < 4$ if $i \notin \{8, 9\}$. Hence, $\text{scd}(w) = 4$.

In our practical motivation given above, we state that we wish to sequentially arrange parallel sequences of memory accesses. These sequences shall be modelled by words and the procedure of sequentially arranging them is described by the shuffle operation. Furthermore, our goal is to construct a shuffle word such that, for any memory access in the shuffle word, the maximum number of values that already have been accessed and shall again be accessed later on is minimal. For instance, in the shuffle word `abaabccbc` of `abacbc` and `abc`, for each position i, $1 \leq i \leq 9$, there exists at most one other symbol that has an occurrence to both sides of position i. On the other hand, with respect to the shuffle word `abacbcabc` we observe that at position 4 symbol `c` occurs while both symbols `a` and `b` have an occurrence to both sides of position 4. This number of symbols occurring to both sides of an occurrence of another symbol is precisely the scope coincidence degree. Hence, our central problem is the problem of finding, for any given set of words, a shuffle word with a minimum scope coincidence degree.

Problem 3.22. For an arbitrary alphabet Σ, let the problem SWminSCD_Σ be the problem of finding, for given $w_i \in \Sigma^*$, $1 \leq i \leq k$, a shuffle word $w \in w_1 ⧢ \ldots ⧢ w_k$ with minimum scope coincidence degree.

Note that in the definition of SWminSCD_Σ, the alphabet Σ is constant and not part of the input; hence, for each alphabet Σ, inputs for the problem SWminSCD_Σ have to consist of words over the alphabet Σ exclusively. This shall be important for complexity considerations.

A naive approach to solving SWminSCD_Σ on input $(w_1, w_2, \ldots, w_k)$ would be to enumerate all elements in $w_1 ⧢ w_2 ⧢ \ldots ⧢ w_k$ in order to find one with minimum scope coincidence degree. However, the size of this search space is too large, as the cardinality of the shuffle $w_1 ⧢ w_2 ⧢ \ldots ⧢ w_k$ is, in the worst case, given by the multinomial coefficient [18]. More precisely,

$$|w_1 ⧢ w_2 ⧢ \ldots ⧢ w_k| \leq \binom{n}{|w_1|, |w_2|, \ldots, |w_k|} = \frac{n!}{|w_1|! \times |w_2|! \times \ldots \times |w_k|!} \,,$$

where $n := \sum_{i=1}^{k} |w_i|$, and $x!$ denotes the factorial of an integer x. This demonstrates that the search space of a naive algorithm can be exponentially large. Therefore, a polynomial time algorithm cannot simply search the whole shuffle $w_1 ⧢ w_2 ⧢ \ldots ⧢ w_k$, which implies that a more sophisticated strategy is required.

Before we present a successful approach to SWminSCD_Σ in Section 3.4.3, we discuss some simple observations in Sections 3.4.1.1 and 3.4.2. First, we note that solving SWminSCD_Σ on input $w_1, w_2, \ldots, w_k$ by first computing a minimal shuffle word w of w_1 and w_2 (ignoring $w_3, \ldots, w_n$) and then solving SWminSCD_Σ on the smaller input $w, w_3 \ldots, w_n$ and so on is not possible. This can be easily comprehended by considering the words $w_1 := \mathtt{ab}$ and $w_2 := \mathtt{bc}$ and observing that $w := \mathtt{abbc}$ is a shuffle word of w_1 and w_2 that is optimal, since $\text{scd}(w) = 0$. Now, it is not possible to shuffle w with $w_3 := \mathtt{cba}$ in such a way that the resulting shuffle word has a scope coincidence degree of less than 2; however, $w' := w_2 \cdot w_3 \cdot w_1 = \mathtt{bccbaab} \in w_1 ⧢ w_2 ⧢ w_3$ and $\text{scd}(w') = 1$. We can further note that w is in fact the only optimal shuffle word of w_1 and w_2, thus, in terms of the above described approach, we necessarily have to start with a shuffle word of w_1 and w_2 that is *not* optimal in order to obtain an optimal shuffle word of all three words w_1, w_2 and w_3.

Intuitively, it seems obvious that the scope coincidence degree only depends on the leftmost and rightmost occurrences of the symbols. In other words, removing a symbol from a word that does not constitute a leftmost or rightmost occurrence should not change the scope coincidence degree of that word. For instance, if we consider a word $w := \alpha \cdot c \cdot \beta$, where c is a symbol occurring in both α and β, then all symbols in the word w that are in the scope of c are still in the scope of c with respect to the word $\alpha \cdot \beta$. Consequently, we can first remove all occurrences of symbols that are neither leftmost nor rightmost occurrences, then solve SWminSCD_Σ on these reduced words and finally insert the removed occurrences into the shuffle word in such a way that the scope coincidence degree does not increase. This reduction of the input words results in a smaller, but still exponentially large search space. Hence, this approach does not seem to help us solving SWminSCD_Σ in polynomial time. For completeness, we discuss this matter in a bit more detail in the following section.

3.4.1.1 Scope Reduced Words

As mentioned above, all symbols in the word $w := \alpha \cdot c \cdot \beta$ that are in the scope of c, where c is a symbol occurring in both α and β, are still in the scope of c with respect to the word $\alpha \cdot \beta$. However, in order to conclude $\text{scd}(w) = \text{scd}(\alpha \cdot \beta)$, we also have to consider the following situation. In case that $\text{scd}_{|\alpha|+1}(w) = \text{scd}(w)$ (i. e., the position of the symbol c under consideration has maximum scope coincidence degree in w) it is no longer as obvious that this particular occurrence of c can be removed without changing the scope coincidence degree of w.

In this case, we can show that there must exist a position i in w, different from position $|\alpha| + 1$, that also has a maximum scope coincidence degree, i. e., $\text{scd}_i(w) = \text{scd}_{|\alpha|+1}(w)$:

Lemma 3.23. *Let* $w := \alpha \cdot c \cdot \beta \in \Sigma^*$, *where* $c \in \Sigma$, $1 \le |\alpha|$, $1 \le |\beta|$. *If* $c \in (\text{alph}(\alpha) \cap \text{alph}(\beta))$, *then* $\text{scd}(w) = \text{scd}(\alpha \cdot \beta)$.

Proof. Let $w' := \alpha \cdot \beta$. Since the occurrence of c at position $|\alpha| + 1$ is neither a leftmost nor a rightmost occurrence, it is obvious that $\text{scd}_i(w) = \text{scd}_i(w')$, $1 \le i \le |\alpha|$, and $\text{scd}_{i+1}(w) = \text{scd}_i(w')$, $|\alpha| + 1 \le i \le |\alpha \cdot \beta|$. First, we observe that if $\text{scd}_{|\alpha|+1}(w) \le \text{scd}_{|\alpha|}(w)$, then we can conclude

$$\begin{aligned}
\text{scd}(w) &= \max\{\text{scd}_i(w) \mid 1 \le i \le |w|, i \ne |\alpha| + 1\} \\
&= \max\{\text{scd}_i(w') \mid 1 \le i \le |w'|\} \\
&= \text{scd}(w')\,.
\end{aligned}$$

So in order to prove the statement of the lemma, it is sufficient to show that $\text{scd}_{|\alpha|+1}(w) \le \text{scd}_{|\alpha|}(w)$. Now, as $1 \le |\alpha|$, there exists a $b \in \Sigma$ such that $\alpha = \alpha' \cdot b$. In case that $b = c$, $\text{scd}_{|\alpha|+1}(w) = \text{scd}_{|\alpha|}(w)$. Therefore, in the following, we assume that $b \ne c$ and define the set $\Gamma := \{a \mid a \in (\text{alph}(\alpha) \cap \text{alph}(\beta)) \setminus \{b, c\}\}$. There are two cases depending on whether or not $b \in \text{alph}(\beta)$.

If $b \in \text{alph}(\beta)$, then $\text{scd}_{|\alpha|+1}(w) = |\Gamma| + |\{b\}|$ and $\text{scd}_{|\alpha|}(w) = |\Gamma| + |\{c\}|$. Hence, $\text{scd}_{|\alpha|+1}(w) = \text{scd}_{|\alpha|}(w)$. If, on the other hand, $b \notin \text{alph}(\beta)$, then $\text{scd}_{|\alpha|+1}(w) = |\Gamma|$

and $\mathrm{scd}_{|\alpha|}(w) = |\Gamma| + |\{c\}|$, which implies $\mathrm{scd}_{|\alpha|+1}(w) < \mathrm{scd}_{|\alpha|}(w)$. □

By iteratively applying Lemma 3.23, it can easily be seen that all occurrences of symbols from a word that are neither leftmost nor rightmost occurrences can be removed without changing its scope coincidence degree. The next definition shall formalise that procedure.

Definition 3.24. Let $w = b_1 \cdot b_2 \cdots b_n$, $b_i \in \Sigma$, $1 \leq i \leq n$, be arbitrarily chosen and, for each i, $1 \leq i \leq n$, let $c_i := \varepsilon$ if $b_i \in (\mathrm{alph}(w[1, i-1]) \cap \mathrm{alph}(w[i+1, -]))$ and $c_i := b_i$ otherwise. The word $c_1 \cdot c_2 \cdots c_n$ (denoted by $\mathrm{sr}(w)$) is called the *scope reduced version of* w. An arbitrary word $v \in \Sigma^*$, such that, for each $b \in \mathrm{alph}(w)$, $|w|_b \leq 2$, is said to be *scope reduced.*

We can now use the previous result and Definition 3.24 in order to show that, regarding the problem $\mathrm{SWminSCD}_\Sigma$, we can restrict our considerations to input words that are scope reduced:

Lemma 3.25. *Let $w_1, w_2, \dots, w_k \in \Sigma^*$. There is a word $u \in w_1 \sqcup\!\sqcup\, w_2 \sqcup\!\sqcup \dots \sqcup\!\sqcup\, w_k$ with $\mathrm{scd}(u) = m$ if and only if there is a word $v \in \mathrm{sr}(w_1) \sqcup\!\sqcup\, \mathrm{sr}(w_2) \sqcup\!\sqcup \dots \sqcup\!\sqcup\, \mathrm{sr}(w_k)$ with $\mathrm{scd}(v) = m$.*

Proof. We prove the *only if* direction by showing that any $u \in w_1 \sqcup\!\sqcup\, w_2 \sqcup\!\sqcup \dots \sqcup\!\sqcup\, w_k$ can be transformed into a $v \in \mathrm{sr}(w_1) \sqcup\!\sqcup\, \mathrm{sr}(w_2) \sqcup\!\sqcup \dots \sqcup\!\sqcup\, \mathrm{sr}(w_k)$ with $\mathrm{scd}(u) = \mathrm{scd}(v)$. The *if* direction shall be shown in a similar way.

The basic idea is that all the symbols from the words w_i, $1 \leq i \leq k$, that are neither leftmost nor rightmost occurrences, can simply be removed from a shuffle word of $w_1, \dots, w_k$ in order to obtain a shuffle word of $\mathrm{sr}(w_1), \dots, \mathrm{sr}(w_k)$, and, analogously, inserting these symbols anywhere, but always between two occurrences of the same symbol, into a shuffle word of $\mathrm{sr}(w_1), \dots, \mathrm{sr}(w_k)$ results in a shuffle word of $w_1, \dots, w_k$. The equivalence of the scope coincidence degree can then be established by Lemma 3.23.

Let $u \in w_1 \sqcup\!\sqcup\, w_2 \sqcup\!\sqcup \dots \sqcup\!\sqcup\, w_k$ be arbitrarily chosen. By definition of a shuffle, we can assume that all symbols in u are marked with one of the numbers in $\{1, 2, \dots, k\}$ in such a way that, for each i, $1 \leq i \leq k$, by deleting all symbols from u that are not marked with i, we obtain exactly w_i. Hence, all symbols in u are of form $b^{(i)}$, where $b \in \Sigma$ and $1 \leq i \leq k$. Next, we obtain a word v from u in the following way. For each $b \in \Sigma$ and each i, $1 \leq i \leq k$, we delete all occurrences of symbols $b^{(i)}$ that are neither leftmost nor rightmost occurrences of the symbol $b^{(i)}$. After that, we unmark all symbols. Since, for each i, $1 \leq i \leq k$, we removed all symbols in u originating from w_i except the left- and rightmost occurrences in w_i, we conclude that v is a shuffle word of $\mathrm{sr}(w_1), \mathrm{sr}(w_2), \dots, \mathrm{sr}(w_k)$ and, by Lemma 3.23, we can conclude that $\mathrm{scd}(u) = \mathrm{scd}(v)$.

In order to prove the *if* direction, we arbitrarily choose a word v of the shuffle $\mathrm{sr}(w_1) \sqcup\!\sqcup\ \mathrm{sr}(w_2) \sqcup\!\sqcup\ \ldots \sqcup\!\sqcup\ \mathrm{sr}(w_k)$. Again, we may assume that all symbols in v are marked in the same way as before. Now, for every $b \in \Sigma$ and every i, $1 \leq i \leq k$, if $|w_i|_b > 2$, we define $n_{b,i} := |w_i|_b - 2$. Next, we construct a word u from v by applying the following algorithm:

```
1: Set $u \leftarrow v$
2: for all $b \in \Sigma$ do
3:     for all $i$, $1 \leq i \leq k$, do
4:         if $|w_i|_b > 2$ then
5:             Let $\alpha$, $\beta$, $\gamma$ such that $u = \alpha \cdot b^{(i)} \cdot \beta \cdot b^{(i)} \cdot \gamma$
6:             Set $u \leftarrow \alpha \cdot b \cdot b^{n_{b,i}} \cdot \beta \cdot b \cdot \gamma$
7:         end if
8:     end for
9: end for
```

For every $b \in \Sigma$ and every i, $1 \leq i \leq k$, with $|w_i|_b > 2$, we can conclude that there are exactly 2 occurrences of $b^{(i)}$ in u, thus, the factorisation in line 5 is unique. In line 6, we simply insert $n_{b,i} = |w_i|_b - 2$ occurrences of symbol b in between the two occurrences of $b^{(i)}$, which we unmark. Consequently, the word u constructed by the above given algorithm is a shuffle word of $w_1, w_2, \ldots, w_k$ and, by Lemma 3.23, we can conclude that $\mathrm{scd}(u) = \mathrm{scd}(v)$. □

The previous result also shows how to obtain a solution for $\mathrm{SWminSCD}_\Sigma$ on input words $w_1, w_2, \ldots, w_k$ from a solution for $\mathrm{SWminSCD}_\Sigma$ on the scope reduced input words $\mathrm{sr}(w_1), \mathrm{sr}(w_2), \ldots, \mathrm{sr}(w_k)$. Although the above made observations are more or less irrelevant for our main results, we shall use them at the very end of this work in order to obtain a better complexity bound.

In the following section, we shall establish basic results about the scope coincidence degree of words. These results shall then be applied later on in order to analyse the scope coincidence degree of shuffle words.

3.4.2 Further Properties of the Scope Coincidence Degree

In this section, we take a closer look at the scope coincidence degree. We are particularly interested in how words can be transformed without increasing their scope coincidence degree. First, we note that the scope coincidence degree of a single position i in some word w, i. e., $\mathrm{scd}_i(w)$, does not change if we permute the prefix $w[1, i-1]$ or the suffix $w[i+1, -]$. This follows directly from the definition, since $\mathrm{scd}_i(w)$ is the number of distinct symbols that occur to both sides of position i, which remains unchanged if $w[1, i-1]$ or $w[i+1, -]$ are permuted.

Proposition 3.26. *Let $u, v \in \Sigma^*$ with $|u| = |v|$. If, for some i, $1 \leq i \leq |u|$, $u[i] = v[i]$ and $u[1, i-1]$ is a permutation of $v[1, i-1]$ and $u[i+1, -]$ is a permutation of $v[i+1, -]$, then $\mathrm{scd}_i(u) = \mathrm{scd}_i(v)$.*

Hence, for every position in a word we can permute the part to the left or to the right of this position without changing its scope coincidence degree. The scope coincidence degree of the positions in the parts that are permuted is not necessarily stable, and thus the scope coincidence degree of the whole word may change. However, if a factor of a word w satisfies a certain property, i. e., it contains no leftmost occurrence of a symbol with respect to w (it may, however, contain rightmost occurrences of symbols), then we can arbitrarily permute this factor without changing the scope coincidence degree of the whole word:

Lemma 3.27. *Let α, β, π, $\pi' \in \Sigma^*$, where π is a permutation of π' and $\mathrm{alph}(\pi) \subseteq \mathrm{alph}(\alpha)$. Then $\mathrm{scd}(\alpha \cdot \pi \cdot \beta) = \mathrm{scd}(\alpha \cdot \pi' \cdot \beta)$.*

Proof. We prove $\mathrm{scd}(v) = \mathrm{scd}(v')$, where $v := \alpha \cdot \pi \cdot \beta$ and $v' := \alpha \cdot \pi' \cdot \beta$. By Proposition 3.26, we can conclude that, for each i, with $1 \leq i \leq |\alpha|$ or $|\alpha \cdot \pi| + 1 \leq i \leq |v|$, $\mathrm{scd}_i(v) = \mathrm{scd}_i(v')$. So it remains to examine the numbers $\mathrm{scd}_i(v)$, $\mathrm{scd}_i(v')$, where $|\alpha| + 1 \leq i \leq |\alpha \cdot \pi|$. In particular, we take a closer look at $\mathrm{scd}_{|\alpha|+1}(v)$ and $\mathrm{scd}_{|\alpha|+1}(v')$, which are determined by the number of symbols different from $\pi[1]$ ($\pi'[1]$, respectively) that occur in both factors α and $\pi[2, -] \cdot \beta$ ($\pi'[2, -] \cdot \beta$, respectively). These symbols can be divided into two sets, the set of symbols occurring in $\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta)$ but not in $\mathrm{alph}(\pi)$ ($\mathrm{alph}(\pi')$, respectively) on the one hand, and the set $\mathrm{alph}(\pi) \setminus \{\pi[1]\}$ ($\mathrm{alph}(\pi') \setminus \{\pi'[1]\}$, respectively) on the other hand. This is due to the fact that $\mathrm{alph}(\pi) \subseteq \mathrm{alph}(\alpha)$; thus, all symbols in $\mathrm{alph}(\pi) \setminus \{\pi[1]\}$ ($\mathrm{alph}(\pi') \setminus \{\pi'[1]\}$, respectively) have an occurrence to the left and to the right of position $|\alpha| + 1$ in v (v', respectively). Therefore, $\mathrm{scd}_{|\alpha|+1}(v) = \mathrm{scd}_{|\alpha|+1}(v') = (m-1) + r$, where $m := |\mathrm{alph}(\pi)|$ and $r := |(\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta)) \setminus \mathrm{alph}(\pi)|$. If we consider the numbers $\mathrm{scd}_i(v)$, $\mathrm{scd}_i(v')$, $|\alpha| + 2 \leq i \leq |\alpha \cdot \pi|$ we encounter the same situation with the only difference that not necessarily all $m-1$ symbols in $\mathrm{alph}(\pi) \setminus \{v[i]\}$ ($\mathrm{alph}(\pi') \setminus \{v'[i]\}$, respectively) have to occur to the right of position i. Hence, $\mathrm{scd}_{|\alpha|+i}(v) = r + m'$ and $\mathrm{scd}_{|\alpha|+i}(v) = r + m''$, where $m' \leq (m-1)$ and $m'' \leq (m-1)$. In conclusion,

- $\max\{\mathrm{scd}_i(v) \mid |\alpha| + 1 \leq i \leq |\alpha \cdot \pi|\} = \mathrm{scd}_{|\alpha|+1}(v)$,
- $\max\{\mathrm{scd}_i(v') \mid |\alpha| + 1 \leq i \leq |\alpha \cdot \pi'|\} = \mathrm{scd}_{|\alpha|+1}(v')$, and
- $\mathrm{scd}_{|\alpha|+1}(v) = \mathrm{scd}_{|\alpha|+1}(v')$.

Thus, $\mathrm{scd}(v) = \mathrm{scd}(v')$. $\square$

The next two lemmas show that if certain conditions hold, then we can move one or several symbols in a word to the left without increasing the scope coincidence degree. The first result of that kind is related to the situation where only one symbol is moved, and the second lemma describes the case where several symbols are moved and therefore makes use of the first lemma.

We can informally summarise the first lemma in the following way. We assume that at position i in a word w a certain symbol b occurs and, furthermore, this is not the leftmost occurrence of b. Then we can move this symbol to the left without increasing the scope coincidence degree of w as long as it is not moved to the left of the leftmost occurrence of a b in w. This seems plausible, as such an operation shortens the scope of symbol b or leaves it unchanged. However, we might move this certain b into a region of the word where many scopes coincide; thus, it is possible that the scope coincidence degree of the new position of b increases compared to its old position. We can show that this increase of the scope coincidence degree of that certain position does not affect the scope coincidence degree of the whole word:

Lemma 3.28. *For all $\alpha, \beta, \gamma \in \Sigma^*$ and for each $b \in \Sigma$ with $b \in \mathrm{alph}(\alpha)$,*

$$\mathrm{scd}(\alpha \cdot b \cdot \beta \cdot \gamma) \leq \mathrm{scd}(\alpha \cdot \beta \cdot b \cdot \gamma)\,.$$

Proof. Let $w := \alpha \cdot b \cdot \beta \cdot \gamma$ and $w' := \alpha \cdot \beta \cdot b \cdot \gamma$. Furthermore, let $j := |\alpha \cdot b|$. We prove the statement of the lemma by showing that $\mathrm{scd}_i(w) \leq \mathrm{scd}(w')$, for each i, $1 \leq i \leq |w|$.

By applying Proposition 3.26, we can conclude that for each i with $1 \leq i \leq j-1$ or $|\alpha \cdot b \cdot \beta| + 1 \leq i \leq |w|$, $\mathrm{scd}_i(w) = \mathrm{scd}_i(w')$. For the positions in w that are in factor β, i. e. the positions i with $j+1 \leq i \leq j + |\beta|$, we observe the following. For each i, $j+1 \leq i \leq j+|\beta|$, there is a certain number of symbols different from symbol $w[i]$ that occur to the left and to the right of position i in w. Regarding w', as in w' the factor β is simply shifted one position to the left, the very same symbols occur to the left and to the right of position $i-1$ in w'. In addition to that, we know that symbol b has an occurrence to the left and to the right of position $i-1$ in w', whereas in w and with respect to position i, this is not necessarily the case. Therefore, we can conclude that $\mathrm{scd}_i(w) \leq \mathrm{scd}_{i-1}(w')$, $j+1 \leq i \leq j + |\beta|$.

So far, we showed that $\mathrm{scd}_i(w) \leq \mathrm{scd}(w')$ for each i with $1 \leq i \leq |w|$ and $i \neq j$. Thus, it only remains to take a closer look at position j in w and, in particular, at the number $\mathrm{scd}_j(w)$. In general, it is possible that $\mathrm{scd}_j(w) > \mathrm{scd}_{|\alpha \cdot \beta|+1}(w')$, but we shall see that always $\mathrm{scd}_j(w) \leq \mathrm{scd}(w')$ holds. We consider the symbol y at position $j-1$ in w, i. e. the last symbol of the factor α (recall that $|\alpha| \geq 1$). Now we can write w as $w := \alpha' \cdot y \cdot b \cdot \beta \cdot \gamma$, where $\alpha = \alpha' \cdot y$. If $y = b$, then obviously

$\mathrm{scd}_j(w) = \mathrm{scd}_{j-1}(w)$ and we already know that $\mathrm{scd}_{j-1}(w) = \mathrm{scd}_{j-1}(w')$, hence, $\mathrm{scd}_j(w) \leq \mathrm{scd}(w')$. We assume that, on the other hand, $y \neq b$. Furthermore, we assume to the contrary that $\mathrm{scd}_j(w) = m > \mathrm{scd}(w')$. This implies that $|\Gamma| = m$, where $\Gamma := |(\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta \cdot \gamma)) \setminus \{b\}|$. Next we consider the set $\Gamma' = (\mathrm{alph}(\alpha') \cap \mathrm{alph}(\beta \cdot b \cdot \gamma)) \setminus \{y\}$ and note that, since $b \in \mathrm{alph}(\alpha')$, $b \in \Gamma'$. We observe now that we have $|\Gamma| = |\Gamma'|$ if $y \in \Gamma$, and $|\Gamma| < |\Gamma'|$ if $y \notin \Gamma$, hence, $|\Gamma| \leq |\Gamma'|$ and, as $|\Gamma'| = \mathrm{scd}_{j-1}(w')$, $m \leq \mathrm{scd}_{j-1}(w')$ is implied, which is a contradiction. $\square$

Obviously, if for some word w the condition of Lemma 3.28 is satisfied not only for one symbol b but for several symbols $d_1, d_2, \dots, d_n$, then we can separately move each of these d_i, $1 \leq i \leq n$, to the left and conclude that the scope coincidence degree of the resulting word does not increase compared to w. This observation is described by the following lemma.

Lemma 3.29. *Let α, γ, $\beta_i \in \Sigma^*$, $0 \leq i \leq n$, $n \in \mathbb{N}$, and let $d_i \in \Sigma$, $1 \leq i \leq n$, such that $d_i \in \mathrm{alph}(\alpha)$, $1 \leq i \leq n$. Then*

$$\mathrm{scd}(\alpha \cdot d_1 \cdot d_2 \cdots d_n \cdot \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma) \leq \mathrm{scd}(\alpha \cdot \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma) .$$

Proof. We prove $\mathrm{scd}(w) \leq \mathrm{scd}(w')$, where

- $w := \alpha \cdot d_1 \cdot d_2 \cdots d_n \cdot \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma$,
- $w' := \alpha \cdot \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma$.

We can obtain a word u_1 from w' by moving d_1 to the left until it is positioned directly to the left of factor β_1. Furthermore, for each i, $2 \leq i \leq n$, we can obtain a word u_i from u_{i-1} by moving the symbol d_i to the left in the same way (i. e. d_i is then positioned between d_{i-1} and β_1). Obviously, $u_n = w$ and, by Lemma 3.28, $\mathrm{scd}(u_1) \leq \mathrm{scd}(w')$ and $\mathrm{scd}(u_i) \leq \mathrm{scd}(u_{i-1})$, $2 \leq i \leq n$, hence, $\mathrm{scd}(w) \leq \mathrm{scd}(w')$. $\square$

Concerning the previous lemma, we observe that we can as well position the symbols d_i, $1 \leq i \leq n$, in any order other than $d_1 \cdot d_2 \cdots d_n$ and would still obtain a word with a scope coincidence degree that has not increased. Furthermore, with Lemma 3.27, we can conclude that the scope coincidence degree is exactly the same, no matter in which order the symbols d_i, $1 \leq i \leq n$, occur between α and β_1.

3.4.3 Solving the Problem $\mathrm{SWminSCD}_\Sigma$

In this section, we present an efficient way to solve $\mathrm{SWminSCD}_\Sigma$. Our approach is established by identifying a certain set of well-formed shuffle words which contains at least one shuffle word with minimum scope coincidence degree and, moreover, is considerably smaller than the set of all shuffle words. To this end, we shall first introduce a general concept for constructing shuffle words, and then a simpler and standardised way of constructing shuffle words is defined. By applying the lemmas given in the previous section, we are able to show that there exists a shuffle word with minimum scope coincidence degree that can be constructed in this simple way.

Let $w_1, w_2, \ldots, w_k \in \Sigma^*$ be arbitrary words. We consider these words as stack-like data structures where the leftmost symbol is the topmost stack element. Now we can empty these stacks by successively applying the pop operation and every time we pop a symbol from a stack, we append this symbol to the end of an initially empty word w. Thus, as soon as all stacks are empty, we obtain a word built up of symbols from the stacks, and this word is certainly a shuffle word of $w_1, w_2, \ldots, w_k$.

It seems useful to reason about different ways of constructing a shuffle word rather than about actual shuffle words, as this allows us to ignore the fact that in general a shuffle word can be constructed in several completely different ways. In particular the following unpleasant situation seems to complicate the analysis of shuffle words. If we consider a shuffle word w of the words $w_1, w_2, \ldots, w_k$, it might be desirable to know, for a symbol b on a certain position j, which w_i, $1 \leq i \leq k$, is the origin of that symbol. Obviously, this depends on how the shuffle word has been constructed from the words w_i, $1 \leq i \leq k$, and for different ways of constructing w, the symbol b on position j may originate from different words w_i, $1 \leq i \leq k$. In particular, if we want to alter shuffle words by moving certain symbols, it is essential to know the origin words w_i, $1 \leq i \leq k$, of the symbols, as this determines how they can be moved without destroying the shuffle properties.

We now formalise the way to construct a shuffle word by utilising the stack analogy introduced above. An arbitrary configuration (of the content) of the stacks corresponding to words w_i, $1 \leq i \leq k$, can be given as a tuple $(v_1, \ldots, v_k)$ of suffixes, i.e. $w_i = u_i \cdot v_i$, $1 \leq i \leq k$. Such a configuration $(v_1, \ldots, v_k)$ is then changed into another configuration $(v_1, \ldots, v_{i-1}, v'_i, v_{i+1}, \ldots, v_k)$, by a pop operation, where $v_i = b \cdot v'_i$ for some i, $1 \leq i \leq k$, and for some $b \in \Sigma$. Initially, we start with the stack content configuration $(w_1, \ldots, w_k)$ and as soon as all the stacks are empty, which can be represented by $(\varepsilon, \ldots, \varepsilon)$, our shuffle word is complete. Hence, we can represent a way to construct a shuffle word by a sequence of these tuples of stack contents:

Definition 3.30. A *construction sequence* for words $w_1, w_2, \ldots, w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, is a sequence $s := (s_0, s_1, \ldots, s_m)$, $m := |w_1 \cdots w_k|$ such that

- $s_i = (v_{i,1}, v_{i,2}, \dots, v_{i,k})$, $0 \leq i \leq m$, where, for each i, $0 \leq i \leq m$, and for each j, $1 \leq j \leq k$, $v_{i,j}$ is a suffix of w_j ,
- $s_0 = (w_1, \dots, w_k)$ and $s_m = (\varepsilon, \varepsilon, \dots, \varepsilon)$,
- for each i, $0 \leq i \leq m-1$, there exists a j_i, $1 \leq j_i \leq k$, and a $b_i \in \Sigma$ such that $v_{i,j_i} = b_i \cdot v_{i+1,j_i}$ and $v_{i,j'} = v_{i+1,j'}$, $j' \neq j_i$.

The shuffle word $w = b_0 \cdot b_1 \cdots b_{m-1}$ is said to *correspond* to s. In a step from s_i to s_{i+1}, $0 \leq i \leq m-1$, of s, we say that the symbol b_{i+1} is *consumed*.

To illustrate the definition of construction sequences, we consider an example construction sequence $s := (s_0, s_1, \dots, s_9)$ corresponding to a shuffle word of the words $w_1 :=$ `abacbc` and $w_2 :=$ `abc`:

$$s := ((\mathtt{abacbc},\ \mathtt{abc}), (\mathtt{bacbc},\ \mathtt{abc}), (\mathtt{bacbc},\ \mathtt{bc}), (\mathtt{bacbc},\ \mathtt{c}),$$
$$(\mathtt{acbc},\ \mathtt{c}), (\mathtt{acbc},\ \varepsilon), (\mathtt{cbc},\ \varepsilon), (\mathtt{bc},\ \varepsilon), (\mathtt{c},\ \varepsilon), (\varepsilon,\ \varepsilon))\,.$$

The shuffle word corresponding to s is $w :=$ `aabbcacbc`, and it is easy to see that $\mathrm{scd}(w) = 2$.

In the next definition, we introduce a certain property of construction sequences that can be easily described in an informal way. Recall that in an arbitrary step from s_i to s_{i+1} of a construction sequence s, exactly one symbol b is consumed. Hence, at each position $s_i = (v_1, \dots, v_k)$ of a construction sequence, we have a part u of already consumed symbols, which is actually a prefix of the shuffle word we are about to construct and some suffixes $v_1, \dots, v_k$ that remain to be consumed. A symbol b that is consumed can be an *old* symbol that already occurs in the part u or it can be a *new* symbol that is consumed for the first time. Now the special property to be introduced next is that this consumption of symbols is greedy with respect to old symbols: Whenever a new symbol b is consumed in a step from s_i to $s_{i+1} = (v_1, \dots, v_k)$, we require the construction sequence to first consume as many old symbols as possible from the remaining $v_1, \dots, v_k$ before another new symbol is consumed. For the sake of uniqueness, this greedy consumption of old symbols shall be defined in a canonical order, i. e. we first consume all the old symbols from v_1, then all the old symbols from v_2 and so on. However, this consumption is canonical only with respect to old symbols. Thus, there are still several possible greedy construction sequences for some input words w_i, $1 \leq i \leq k$, since whenever a new symbol is consumed, we have a choice of k possible suffixes to take this symbol from. We formally define this greedy property of construction sequences.

Definition 3.31. Let $w \in w_1 \sqcup\!\sqcup\, w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup\, w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, and let $s := (s_0, s_1, \ldots, s_{|w|})$ with $s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq |w|$, be an arbitrary construction sequence for w. An element s_i, $1 \leq i \leq |w| - 1$, of s satisfies the *greedy property* if and only if $w[i] \notin \text{alph}(w[1, i-1])$ implies that for each j, $1 \leq j \leq k$, $s_{i+|u_1 \cdots u_j|} = (\overline{v}_{i,1}, \ldots, \overline{v}_{i,j}, v_{i,j+1}, \ldots, v_{i,k})$, where $v_{i,j} = u_j \cdot \overline{v}_{i,j}$ and u_j is the longest prefix of $v_{i,j}$ such that $\text{alph}(u_j) \subseteq \text{alph}(w[1, i])$.

A construction sequence $s := (s_0, s_1, \ldots, s_{|w|})$ for some $w \in \Sigma^*$ is a *greedy construction sequence* if and only if, for each i, $1 \leq i \leq |w| - 1$, s_i satisfies the greedy property. A shuffle word w that corresponds to a greedy construction sequence is a *greedy shuffle word.*

As an example, we again consider the words $w_1 = \texttt{abacbc}$ and $w_2 = \texttt{abc}$. This time, we present a greedy construction sequence $s := (s_0, s_1, \ldots, s_9)$ for w_1 and w_2:

$$\begin{aligned} s := (&(\texttt{abacbc},\ \texttt{abc}), (\texttt{bacbc},\ \texttt{abc}), (\texttt{bacbc},\ \texttt{bc}), (\texttt{bacbc},\ \texttt{c}), \\ &(\texttt{acbc},\ \texttt{c}), (\texttt{cbc},\ \texttt{c}), (\texttt{cbc},\ \varepsilon), (\texttt{bc},\ \varepsilon), (\texttt{c},\ \varepsilon), (\varepsilon,\ \varepsilon))\,. \end{aligned}$$

Obviously, the shuffle word $w := \texttt{aabbaccbc}$ corresponds to the construction sequence s and $\text{scd}(w) = 1$. To show that s is a greedy construction sequence, it is sufficient to observe that s_1, s_3 and s_6 (the elements where a new symbol is consumed) satisfy the greedy property. We only show that s_3 satisfies the greedy property as s_1 and s_6 can be handled analogously. First, we recall that $s_3 = (\texttt{bacbc},\ \texttt{c})$ and note that, in terms of Definition 3.31, we have $u_1 := \texttt{ba}$, $\overline{v}_{3,1} := \texttt{cbc}$, $u_2 := \varepsilon$ and $\overline{v}_{3,2} := \texttt{c}$. By definition, s_3 only satisfies the greedy property if $s_{3+|u_1|} = (\overline{v}_{3,1}, v_{3,2})$ and $s_{3+|u_1 \cdot u_2|} = (\overline{v}_{3,1}, \overline{v}_{3,2})$. Since $|u_1| = |u_1 \cdot u_2| = 2$, $\overline{v}_{3,1} = \texttt{cbc}$, $v_{3,2} = \overline{v}_{3,2} = \texttt{c}$ and $s_5 = (\texttt{cbc},\ \texttt{c})$, this clearly holds.

In the following, we show how we can transform an arbitrary construction sequence $s := (s_0, s_1, \ldots, s_m)$ into a greedy one. Informally speaking, this is done by determining the first element s_i that does not satisfy the greedy property and then we simply redefine all the elements s_j, $i + 1 \leq j \leq m$, in a way such that s_i satisfies the greedy property. If we apply this method iteratively, we can obtain a greedy construction sequence. Next, we introduce the formal definition of that transformation and explain it in more detail later on.

Definition 3.32. We define an algorithm G that transforms a construction sequence. Let $s := (s_0, s_1, \ldots, s_m)$ with $s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq m$, be an arbitrary construction sequence that corresponds to a shuffle word w. In the case that s is a greedy construction sequence, we define $G(s) := s$. If s is not a greedy construction sequence, then let p, $1 \leq p \leq m$, be the smallest number such that

s_p does not satisfy the greedy property. Furthermore, for each j, $1 \leq j \leq k$, let u_j be the longest prefix of $v_{p,j}$ with $\text{alph}(u_j) \subseteq \text{alph}(w[1,p])$ and let $v_{p,j} = u_j \cdot \overline{v}_{p,j}$. For each j, $1 \leq j \leq k$, let $\sigma_j : \Sigma^* \to \Sigma^*$ be a mapping defined by $\sigma_j(x) := \overline{v}_{p,j}$ if $|x| > |\overline{v}_{p,j}|$ and $\sigma_j(x) := x$ otherwise, for each $x \in \Sigma^*$. Furthermore, let the mapping $\sigma : (\Sigma^*)^k \to (\Sigma^*)^k$ be defined by $\sigma((v_1, \ldots, v_k)) := (\sigma_1(v_1), \ldots, \sigma_k(v_k))$, $v_j \in \Sigma^*$, $1 \leq j \leq k$. Finally, we define $G(s) := (s'_0, s'_1, \ldots, s'_{m'})$, where the elements s'_i, $0 \leq i \leq m'$, are defined by the following procedure.

```
1:  $s'_i := s_i$, $0 \leq i \leq p$
2:  for all $j$, $1 \leq j \leq k$, do
3:      $s'_{p+|u_1 \cdots u_j|} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$
4:      for all $l_j$, $2 \leq l_j \leq |u_j|$, do
5:          $s_{p+|u_1 \cdots u_{j-1}|+l_j-1} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j-1}, u_j[l_j, -] \cdot \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$
6:      end for
7:  end for
8:  $q' \leftarrow p + 1$
9:  $q'' \leftarrow p + |u_1 \cdots u_k| + 1$
10: while $q' \leq m$ do
11:     if $\sigma(s_{q'-1}) \neq \sigma(s_{q'})$ then
12:         $s'_{q''} := \sigma(s_{q'})$
13:         $q'' \leftarrow q'' + 1$
14:     end if
15:     $q' \leftarrow q' + 1$
16: end while
```

As mentioned above, we explain the previous definition in an informal way and shall later consider an example. Let $s := (s_0, s_1, \ldots, s_m)$ be an arbitrary construction sequence and let p and the u_j, $1 \leq j \leq k$, be defined as in Definition 3.32. The sequence $s' := (s'_0, s'_1, \ldots, s'_{m'}) := G(s)$ is obtained from s in the following way. We keep the first p elements and then redefine the next $|u_1 \cdots u_k|$ elements in such a way that s'_p satisfies the greedy property as described by Definition 3.31. This is done in lines 1 to 9 of the algorithm. Then, in order to build the rest of s', we modify the elements s_i, $p+1 \leq i \leq m$. First, for each component $v_{i,j}$, $p+1 \leq i \leq m$, $1 \leq j \leq k$, if $|\overline{v}_{p,j}| < |v_{i,j}|$ we know that $v_{i,j} = \overline{u}_j \cdot \overline{v}_{p,j}$, where $\overline{u}_j$ is a suffix of u_j. In s', this part $\overline{u}_j$ has already been consumed by the new elements s'_i, $p+1 \leq i \leq p+|u_1 \cdots u_k|$, and is, thus, simply cut off and discarded by the mapping σ in Definition 3.32. More precisely, if a component $v_{i,j}$, $p+1 \leq i \leq m$, $1 \leq j \leq k$, of an element s_i is longer than $\overline{v}_{p,j}$, then $\sigma_j(v_{i,j}) = \overline{v}_{i,j}$. If on the other hand $|v_{i,j}| \leq |\overline{v}_{p,j}|$, then $\sigma(v_{i,j}) = v_{i,j}$. This is done in lines 10 to 16 of the algorithm.

The following proposition shows that $G(s)$ actually satisfies the conditions to be a proper construction sequence:

Proposition 3.33. *For each construction sequence s of some words $w_1, \dots, w_k$, $G(s)$ is also a construction sequence of the words $w_1, \dots, w_k$.*

Proof. Let $s := (s_0, s_1, \dots, s_m)$ and $s' := (s'_0, s'_1, \dots, s'_{m'}) := G(s)$, where $s_i := (v_{i,1}, \dots, v_{i,k})$, $0 \leq i \leq m$, $s'_{i'} := (v'_{i',1}, \dots, v'_{i',k})$, $0 \leq i' \leq m'$. We assume that s is not greedy, as otherwise $G(s) = s$ and the statement of the proposition trivially holds. Hence, let p, $1 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. In order to show that s' is a construction sequence, we need to show that the following conditions hold:

1. $s'_0 = (w_1, w_2, \dots, w_k)$.
2. $s'_{m'} = (\varepsilon, \varepsilon, \dots, \varepsilon)$.
3. For each i, $0 \leq i \leq m' - 1$, there exists a j_i, $1 \leq j_i \leq k$, and a $b_i \in \Sigma$, such that $v'_{i,j_i} = b_i \cdot v'_{i+1,j_i}$ and $v'_{i,j'} = v'_{i+1,j'}$, $j' \neq j_i$.

Condition 1 is clearly satisfied as $s'_0 = s_0 = (w_1, \dots, w_k)$. We note that it is sufficient to prove that condition 3 is satisfied, as this implies condition 2 and, furthermore, $m = m'$. For each i, $0 \leq i \leq p + |u_1 \cdots u_k| - 1$, condition 3 is clearly satisfied. To show the same for each i, $p + |u_1 \cdots u_k| \leq i \leq m'$, we consider the mapping σ from Definition 3.32. This mapping is defined in a way that, for an arbitrary $s_i = (v_{i,1}, \dots, v_{i,k})$, $\sigma(s_i) = (\widetilde{v}_{i,1}, \dots, \widetilde{v}_{i,k})$, with, for each j, $1 \leq j \leq k$, either $\widetilde{v}_{i,j} = v_{i,j}$, if $|v_{i,j}| \leq |\overline{v}_{p,j}|$ or $\widetilde{v}_{i,j} = \overline{v}_{p,j}$, if $|v_{i,j}| > |\overline{v}_{p,j}|$, where $\overline{v}_{p,j}$ is defined as in Definition 3.32. Consequently, for each i, $p + 1 \leq i \leq m$, we have either $\sigma(s_{i-1}) = \sigma(s_i)$ or $\sigma(s_{i-1}) = (\widetilde{v}_{i,1}, \dots, \widetilde{v}_{i,j-1}, b \cdot \widetilde{v}_{i,j}, \widetilde{v}_{i,j+1}, \dots, \widetilde{v}_{i,k})$ and $\sigma(s_i) = (\widetilde{v}_{i,1}, \dots, \widetilde{v}_{i,k})$, for some j, $1 \leq j \leq k$. In lines 10 to 16 of the algorithm, we ignore the $\sigma(s_i)$ with $\sigma(s_i) = \sigma(s_{i-1})$ and only keep $\sigma(s_i)$ in the sequence s' if $\sigma(s_i) \neq \sigma(s_{i-1})$. Hence, condition 1 holds for all i, $0 \leq i \leq m - 1$, and, moreover, this implies $m' = m$. □

Now, as an example for Definition 3.32, we consider the construction sequence

$$s := ((\mathtt{abacbc},\ \mathtt{abc}), (\mathtt{bacbc},\ \mathtt{abc}), (\mathtt{bacbc},\ \mathtt{bc}), (\mathtt{bacbc},\ \mathtt{c}),$$
$$(\mathtt{acbc},\ \mathtt{c}), (\mathtt{acbc},\ \varepsilon), (\mathtt{cbc},\ \varepsilon), (\mathtt{bc},\ \varepsilon), (\mathtt{c},\ \varepsilon), (\varepsilon,\ \varepsilon))$$

of the words $w_1 = \mathtt{abacbc}$ and $w_2 = \mathtt{abc}$, as given below Definition 3.30. The shuffle word that corresponds to this construction sequence is $w := \mathtt{aabbcacbc}$. We now illustrate how the construction sequence $s' := (s'_0, s'_1, \dots, s'_m) := G(s)$ is constructed by the algorithm G. First, we note that $s_3 = (\mathtt{bacbc},\ \mathtt{c})$ is the first element that does not satisfy the greedy property, since in the step from s_4 to s_5, the symbol $\texttt{c}$ is consumed before the leftmost (and old) symbol $\texttt{a}$ from $v_{4,1}$ is consumed. Thus, $s'_i = s_i$, $1 \leq i \leq 3$. As $w[1,3] = \texttt{aab}$, we conclude that $u_1 := \texttt{ba}$ and $u_2 := \varepsilon$. So the next two elements s'_4 and s'_5 consume the factor u_1 from $\texttt{bacbc}$, hence, $s'_4 = (\texttt{acbc},\, \texttt{c})$ and $s'_5 = (\texttt{cbc},\, \texttt{c})$. Now let σ be defined as in Definition 3.32, thus,

$$\sigma(s_3) = (\texttt{cbc},\, \texttt{c}), \sigma(s_4) = (\texttt{cbc},\, \texttt{c}), \sigma(s_5) = (\texttt{cbc},\, \varepsilon),$$
$$\sigma(s_6) = (\texttt{cbc},\, \varepsilon), \sigma(s_7) = (\texttt{bc},\, \varepsilon), \sigma(s_8) = (\texttt{c},\, \varepsilon), \sigma(s_9) = (\varepsilon,\, \varepsilon)\,.$$

Since $\sigma(s_3) = \sigma(s_4)$ and $\sigma(s_5) = \sigma(s_6)$, we ignore $\sigma(s_4)$ and $\sigma(s_6)$; hence,

$$s'_6 = \sigma(s_5) = (\texttt{cbc},\, \varepsilon), s'_7 = \sigma(s_7) = (\texttt{bc},\, \varepsilon),$$
$$s'_8 = \sigma(s_8) = (\texttt{c},\, \varepsilon), s'_9 = \sigma(s_9) = (\varepsilon,\, \varepsilon)\,.$$

In conclusion,

$$s' = ((\texttt{abacbc},\, \texttt{abc}), (\texttt{bacbc},\, \texttt{abc}), (\texttt{bacbc},\, \texttt{bc}), (\texttt{bacbc},\, \texttt{c}),$$
$$(\texttt{acbc},\, \texttt{c}), (\texttt{cbc},\, \texttt{c}), (\texttt{cbc},\, \varepsilon), (\texttt{bc},\, \varepsilon), (\texttt{c},\, \varepsilon), (\varepsilon,\, \varepsilon))\,.$$

Next, we show that if in a construction sequence $s := (s_0, s_1, \dots, s_m)$ the element s_p is the first element that does not satisfy the greedy property, then in $G(s) := (s'_0, s'_1, \dots, s'_m)$ the element s'_p satisfies the greedy property. This follows from Definition 3.32 and has already been explained informally.

Proposition 3.34. *Let $s := (s_0, s_1, \dots, s_m)$ be any construction sequence that is not greedy, and let p, $0 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. Let $s' := (s'_0, s'_1, \dots, s'_m) := G(s)$ and, if s' is not greedy, let q, $0 \leq q \leq m$, be the smallest number such that s'_q does not satisfy the greedy property. Then $p < q$.*

Proof. Let $s_i := (v_{i,1}, v_{i,2}, \dots, v_{i,k})$, $0 \leq i \leq m$. We assume that s' is not greedy and note that, by Definition 3.32, $s'_i = s_i$, $1 \leq i \leq p$. Hence, all the elements s'_i, $1 \leq i \leq p-1$, satisfy the greedy property. To prove $p < q$ it is sufficient to show that s'_p satisfies the greedy property.

Since $w[1,p] = w'[1,p]$, we can conclude that $w'[p] \notin \mathrm{alph}(w'[1,p-1])$. Furthermore, line 5 of the algorithm given in Definition 3.32 makes sure that, for every j, $1 \leq j \leq k$,

$$s'_{p+|u_1 \cdots u_j|} = (\overline{v}_{p,1}, \dots, \overline{v}_{p,j}, v_{p,j+1}, \dots, v_{p,k})\,,$$

where the $\overline{v}_{p,j}$ are defined as in Definition 3.32. Consequently, s'_p satisfies the greedy property, and therefore $p < q$. □

More importantly, we can also state that the scope coincidence degree of the shuffle word corresponding to $G(s)$ does not increase compared to the shuffle word that corresponds to s. To this end, we shall employ the lemmas introduced in Section 3.4.2.

Lemma 3.35. *Let s be an arbitrary construction sequence that corresponds to the shuffle word w and let w' be the shuffle word corresponding to $G(s)$. Then* $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

Proof. Let $s := (s_0, s_1, \ldots, s_m)$, $s' := G(s) := (s'_0, s'_1, \ldots, s'_m)$ and, for each i, $0 \leq i \leq m$, $s_i := (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $s'_i := (v'_{i,1}, v'_{i,2}, \ldots, v'_{i,k})$. In this proof we shall use a special terminology: If for some i, i' with $1 \leq i < i' \leq m$, $v_{i,j} := u_{i,j} \cdot v_{i',j}$, $1 \leq j \leq k$, then we say that the $u_{i,j}$, $1 \leq j \leq k$, are consumed from the $v_{i,j}$, $1 \leq j \leq k$, by the part $s_i, s_{i+1}, \ldots, s_{i'}$.

If s is a greedy construction sequence, then $G(s) = s$ and we are done. Therefore, we assume that s is not a greedy construction sequence and let p, $0 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. For each $v_{p,j}$, $1 \leq j \leq k$, we define $v_{p,j} = u_j \cdot \overline{v}_{p,j}$, where u_j is the longest prefix of $v_{p,j}$ with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w[1,p])$.

To prove $\mathrm{scd}(w') \leq \mathrm{scd}(w)$, we have to consider two possible cases. The first case is that $\mathrm{alph}(w[p+1,-]) \subseteq \mathrm{alph}(w[1,p])$, i. e. $w[p]$ is the last new symbol that is consumed in s; thus $\overline{v}_{p,j} = \varepsilon$, $1 \leq j \leq k$. The second case is that this property is not satisfied, so there exists a $c \in \Sigma$, such that $w[p+1,-] = \alpha \cdot c \cdot \beta$ with $c \notin \mathrm{alph}(w[1, p+|\alpha|])$. In other words, c is the next new symbol that is consumed in s after b is consumed in the step from s_{p-1} to s_p.

We start with the latter case and note that we can write w as follows:

$$w = \alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta \,,$$

where $|\alpha_1| = p - 1$, $c \notin \mathrm{alph}(\alpha_1 \cdot b \cdot \alpha_2)$ and $\mathrm{alph}(\alpha_2) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$. Before we continue, we explain the main idea of the proof. By definition of the transformation G, we know that the shuffle word w' begins with the same prefix as w, i.e. $w' = \alpha_1 \cdot b \cdot \delta$, but the suffix δ may differ from $\alpha_2 \cdot c \cdot \beta$. In the following we show that the suffix $\alpha_2 \cdot c \cdot \beta$ from w can be gradually transformed into δ without increasing the scope coincidence degree of w.

Next, we take a closer look at w and notice that α_2 exclusively consists of symbols from the prefixes u_j, $1 \leq j \leq k$. That is due to the fact that $\mathrm{alph}(\alpha_2) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$ and, for each j, $1 \leq j \leq k$, u_j is the longest prefix of $v_{p,j}$ with

$\mathrm{alph}(u_j) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$. Consequently, we can consider the prefixes u_j, $1 \leq j \leq k$, as being factorised into $u_j = \widetilde{u}_j \cdot \widehat{u}_j$ such that

$$s_{p+|\alpha_2|} = (\widehat{u}_1 \cdot \overline{v}_{p,1}, \widehat{u}_2 \cdot \overline{v}_{p,2}, \ldots, \widehat{u}_k \cdot \overline{v}_{p,k}) \,.$$

In other words, as $s_p = (\widetilde{u}_1 \cdot \widehat{u}_1 \cdot \overline{v}_{p,1}, \ldots, \widetilde{u}_k \cdot \widehat{u}_k \cdot \overline{v}_{p,k})$, exactly the prefixes $\widetilde{u}_j$ are consumed by the part $s_p, s_{p+1}, \ldots, s_{p+|\alpha_2|}$ of s, and, thus, $\alpha_2 \in \widetilde{u}_1 \sqcup\!\sqcup \widetilde{u}_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup \widetilde{u}_k$. Moreover, the suffix $c \cdot \beta$ exclusively consists of symbols consumed in steps from s_i to s_{i+1}, $p + |\alpha_2| \leq i \leq m - 1$. Thus, $c \cdot \beta \in \widehat{u}_1 \cdot \overline{v}_{p,1} \sqcup\!\sqcup \widehat{u}_2 \cdot \overline{v}_{p,2} \sqcup\!\sqcup \ldots \sqcup\!\sqcup \widehat{u}_k \cdot \overline{v}_{p,k}$. Now let $n := |\widehat{u}_1 \cdot \widehat{u}_2 \cdots \widehat{u}_k|$. We can conclude that the n symbols from the factors $\widehat{u}_j$, $1 \leq j \leq k$, occur somewhere in $c \cdot \beta$, and, furthermore, since $c \notin \mathrm{alph}(\widehat{u}_j)$, $1 \leq j \leq k$, we also know that all these n symbols occur in β. Thus we can write

$$\beta = \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma \,,$$

where the symbols d_j, $1 \leq j \leq n$, are exactly the symbols consumed from the $\widehat{u}_j$, $1 \leq j \leq k$, i. e., for each $i \in \{|\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta_1 \cdot d_1 \cdots \beta_{i'}| \mid 1 \leq i' \leq n\}$ there exists a j_i, $1 \leq j_i \leq k$, such that $v_{i,j_i} = d_i \cdot v_{i+1,j_i}$ and $v_{i,j'} = v_{i+1,j'}$, $j' \neq j_i$, and, furthermore, $|v_{i+1,j_i}| \geq |\overline{v}_{p,j_i}|$. This means, in particular, that $d_1 \cdot d_2 \cdots d_n \in \widehat{u}_1 \sqcup\!\sqcup \widehat{u}_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup \widehat{u}_k$ and $c \cdot \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma \in \overline{v}_{p,1} \sqcup\!\sqcup \overline{v}_{p,2} \sqcup\!\sqcup \ldots \sqcup\!\sqcup \overline{v}_{p,k}$, and therefore,

$$\alpha_2 \cdot d_1 \cdot d_2 \cdots d_n \in \widetilde{u}_1 \cdot \widehat{u}_1 \sqcup\!\sqcup \widetilde{u}_2 \cdot \widehat{u}_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup \widetilde{u}_k \cdot \widehat{u}_k = u_1 \sqcup\!\sqcup u_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup u_k \,.$$

On the other hand, by Definition 3.32, we know that s' is constructed such that the prefixes u_j, $1 \leq j \leq k$, are consumed by $s'_p, s'_{p+1}, \ldots, s'_{|u_1 \cdot u_2 \cdots u_k|}$ in a canonical way, i. e. we can write w' as

$$w' = \alpha_1 \cdot b \cdot u_1 \cdot u_2 \cdots u_k \cdot c' \cdot \beta' \,.$$

Since, for each j, $1 \leq j \leq k$, u_j is the longest prefix of $v_{p,j}$ with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$, we know that $c' \notin \mathrm{alph}(\alpha_1 \cdot b \cdot u_1 \cdots u_k)$. In the following, we show that $c' = c$. To this end, we recall that $s_{p+|\alpha_2|} = (\widehat{u}_1 \cdot \overline{v}_{p,1}, \ldots, \widehat{u}_k \cdot \overline{v}_{p,k})$ and the symbol c is consumed in the step from $s_{p+|\alpha_2|}$ to $s_{p+|\alpha_2|+1}$. More precisely, for some j', $1 \leq j' \leq k$, $\widehat{u}_{j'} = \varepsilon$, $\overline{v}_{p,j'}[1] = c$ and $v_{p+|\alpha_2|+1,j'} = \overline{v}_{p,j'}[2,-]$. Since $|\widehat{u}_j \cdot \overline{v}_{p,j}| \geq |\overline{v}_{p,j}|$, $1 \leq j \leq k$, we can conclude that $\sigma(s_{p+|\alpha_2|}) = (\overline{v}_{p,1}, \ldots, \overline{v}_{p,k})$ and, for the same reason, $\sigma(s_i) = (\overline{v}_{p,1}, \ldots, \overline{v}_{p,k})$, for each i, $p \leq i \leq p+|\alpha_2|$. Hence, $\sigma(s_{i-1}) = \sigma(s_i)$, $p+1 \leq i \leq p+|\alpha_2|$, and $\sigma(s_{p+|\alpha_2|}) \neq \sigma(s_{p+|\alpha|+1})$. By recalling lines 10 to 16 of Definition 3.32, we can observe that this implies $s'_{p+|u_1 \cdots u_k|} = (v_{p,1}, \ldots, v_{p,k})$ and, furthermore, $s'_{p+|u_1 \cdots u_k|+1} = \sigma(s_{p+|\alpha_2|+1}) = (v_{p,1}, \ldots, v_{p,j'-1}, v_{p,j'}[2,-], v_{p,j'+1}, \ldots, v_{p,k})$, where $v_{p,j'}[1] = c$. This directly implies $w'[p + |u_1 \cdots u_k| + 1] = c$ and thus,

$c = c'$.

Next, we show that $\beta' = \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma$. We already know that $\beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma \in \overline{v}_{p,1} \sqcup\!\sqcup \overline{v}_{p,2} \sqcup\!\sqcup \ldots \sqcup\!\sqcup \overline{v}_{p,k}$ and clearly $\beta' \in \overline{v}_{p,1} \sqcup\!\sqcup \overline{v}_{p,2} \sqcup\!\sqcup \ldots \sqcup\!\sqcup \overline{v}_{p,k}$, too. Now we recall that $\beta = \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma$ is constructed by the part $t := (s_{p+|\alpha_2|+1}, s_{p+|\alpha_2|+2}, \ldots, s_m)$ of the construction sequence s and β' is constructed by $t' := (s'_{p+|u_1 \cdots u_k|+1}, s'_{p+|u_1 \cdots u_k|+2}, \ldots, s'_m)$. By Definition 3.32, t' is the same as $(\sigma(s_{p+|\alpha_2|+1}), \sigma(s_{p+|\alpha_2|+2}), \ldots, \sigma(s_m))$ with the only difference, that duplicate elements have been removed. These duplicate elements are exactly the elements that consume the symbols d_i, $1 \leq i \leq n$, and therefore, we can conclude that t and t' construct the same shuffle word.

We consider now the scope coincidence degree of w. Obviously,

$$\mathrm{scd}(w) = \mathrm{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta) = \mathrm{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta_1 \cdot d_1 \cdots \beta_n \cdot d_n \cdot \gamma)\,.$$

Next, we recall that $d_i \in \mathrm{alph}(\alpha_1 \cdot b \cdot \alpha_2)$, $1 \leq i \leq n$, and therefore, by applying Lemma 3.29, we can move all the symbols d_i, $1 \leq i \leq n$, to the left, directly next to symbol c, without increasing the scope coincidence degree, i. e.

$$\mathrm{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta_1 \cdot d_1 \cdots \beta_n \cdot d_n \cdot \gamma) \geq \mathrm{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot d_1 \cdots d_n \cdot c \cdot \beta_1 \cdots \beta_n \cdot \gamma)\,.$$

Now we recall that $\alpha_2 \cdot d_1 \cdots d_n \in u_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup u_k$, and, thus, is actually a permutation of $u_1 \cdots u_k$. Moreover, by definition, for each j, $1 \leq j \leq k$, $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$. Consequently, by Lemma 3.27, we can substitute $u_1 \cdots u_k$ for $\alpha_2 \cdot d_1 \cdots d_n$ without changing the scope coincidence degree and, furthermore, we can substitute β' for $\beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma$:

$$\mathrm{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot d_1 \cdots d_n \cdot c \cdot \beta_1 \cdots \beta_n \cdot \gamma) = \mathrm{scd}(\alpha_1 \cdot b \cdot u_1 \cdots u_k \cdot c \cdot \beta') = \mathrm{scd}(w')\,.$$

Hence, $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

It remains to prove $\mathrm{scd}(w') \leq \mathrm{scd}(w)$ for the case that $\mathrm{alph}(w[p+1,-]) \subseteq \mathrm{alph}(w[1,p])$. In this case, the situation is not as difficult as before. We can write w' as

$$w' = \alpha_1 \cdot b \cdot u_1 \cdots u_k\,.$$

Furthermore, $\mathrm{alph}(w[p+1,-]) \subseteq \mathrm{alph}(w[1,p])$ implies $s_p = (u_1, u_2, \ldots, u_k)$; thus,

$$w = \alpha_1 \cdot b \cdot \alpha_2\,,$$

where α_2 is a permutation of $u_1 \cdots u_k$. As $\mathrm{alph}(\alpha_2) = \mathrm{alph}(u_1 \cdots u_k) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$, we can apply Lemma 3.27 and conclude $\mathrm{scd}(w') = \mathrm{scd}(w)$. □

The previous lemma is very important, as it implies our next result, which can be stated as follows. By iteratively applying the algorithm G, we can transform each construction sequence, including the ones corresponding to shuffle words with minimum scope coincidence degree, into a greedy construction sequence that corresponds to a shuffle word with a scope coincidence degree that is the same or even lower:

Theorem 3.36. *Let $w \in w_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, be an arbitrary shuffle word. There exists a greedy shuffle word w' such that* $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

Proof. Let s be an arbitrary construction sequence of w. We define $s' := G^{|\Sigma|}(s)$, where $G^k(s)$ is the k-fold application of the mapping G on s, i. e. $G^k(s) = G(G(\ldots G(s))\ldots)$. Obviously, in w there exist $|\Sigma|$ positions i, $1 \leq i \leq |w|$, such that $w[i] \notin \mathrm{alph}(w[1, i-1])$. Thus, in s there exist at most $|\Sigma|$ elements s_i, $1 \leq i \leq |w|$, that do not satisfy the greedy property. Therefore, by Proposition 3.34, we conclude that s' is a greedy construction sequence and Lemma 3.35 implies that $\mathrm{scd}(w') \leq \mathrm{scd}(w)$, where w' is the shuffle word corresponding to s'. □

This particularly implies that there exists a greedy shuffle word with minimum scope coincidence degree. Hence, $\mathrm{SWminSCD}_\Sigma$ reduces to the problem of finding a greedy shuffle word with minimum scope coincidence degree.

The following algorithm – referred to as SolveSWminSCD – applies the above established way to construct greedy shuffle words and enumerates all possible greedy shuffle words in order to solve $\mathrm{SWminSCD}_\Sigma$.

Algorithm 1 SolveSWminSCD

```
1:  optShuffle := $\varepsilon$, minscd := $|\Sigma|$, push $(\varepsilon, (w_1, \ldots, w_k))$
2:  while the stack is not empty do
3:      Pop element $(w, (v_1, \ldots, v_k))$
4:      if $|v_1 \cdot v_2 \cdots v_k| = 0$ and $\mathrm{scd}(w) <$ minscd then
5:          optShuffle := $w$
6:          minscd := $\mathrm{scd}(w)$
7:      else
8:          for all $i$, $1 \leq i \leq k$, with $v_i \neq \varepsilon$ do
9:              $b := v_i[1]$
10:             $v_i := v_i[2, -]$
11:             Let $u_j$, $1 \leq j \leq k$, be the longest prefix of $v_j$ with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w \cdot b)$
12:             Push $(w \cdot b \cdot u_1 \cdot u_2 \cdots u_k, (v_1[|u_1|+1, -], v_2[|u_2|+1, -], \ldots, v_k[|u_k|+1, -]))$
13:         end for
14:     end if
15: end while
16: Output optShuffle
```

As a central data structure in our algorithm, we use a stack that is able to store tuples of the form $(w, (v_1, v_2, \dots, v_k))$, where $w, v_i \in \Sigma^*$, $1 \le i \le k$. In the following, all push or pop operations refer to this stack. Initially, the stack stores $(\varepsilon, (w_1, w_2, \dots, w_k))$ (line 1), where $(w_1, w_2, \dots, w_k)$ is the input of the algorithm. We shall see that throughout the whole execution of the algorithm, the stack exclusively stores elements $(w, (v_1, v_2, \dots, v_k))$, where, for each i, $1 \le i \le k$, either $v_i[1] \notin \mathrm{alph}(w)$ or $v_i = \varepsilon$. For the initial element $(\varepsilon, (w_1, w_2, \dots, w_k))$, this property is clearly satisfied. In the main part of the algorithm, we first pop an element $(w, (v_1, v_2, \dots, v_k))$ (line 3) and then, for each i, $1 \le i \le k$, with $v_i \ne \varepsilon$, we carry out the following steps (lines 7 to 12). First we append $b := v_i[1]$ to the end of w, i. e. $w := w \cdot b$ and $v_i := v_i[2, -]$ (lines 8 and 9), then, for each j, $1 \le j \le k$, we compute the longest prefix u_j of v_j, such that $\mathrm{alph}(u_j) \in \mathrm{alph}(w \cdot v_i[1])$ (line 11). After that, we append all these factors u_j, $1 \le j \le k$, to w, i. e. $w := w \cdot u_1 \cdot u_2 \cdots u_k$ and $v_j := v_j[|u_j| + 1, -]$. Finally, $(w, (v_1, v_2, \dots, v_k))$ is pushed on the stack (line 12). When this is done for each i, $1 \le i \le k$, with $v_i \ne \varepsilon$, we pop another element and repeat these steps. Sooner or later, we necessarily pop a tuple $(w, (\varepsilon, \varepsilon, \dots, \varepsilon))$ and according to how the algorithm constructs the new elements that are pushed on the stack, we can conclude that w is a greedy shuffle word of the words $w_1, w_2, \dots, w_k$. Thus, we compute $\mathrm{scd}(w)$ and save both w and $\mathrm{scd}(w)$ in case that $\mathrm{scd}(w)$ is smaller than our current minimum (lines 5 and 6). The algorithm terminates as soon as the stack is completely empty.

We note that in lines 4 and 6 of the algorithm SolveSWminSCD the number $\mathrm{scd}(w)$ needs to be computed, which, by the following proposition, can be done efficiently:

Proposition 3.37. *Let $w \in \Sigma$ be arbitrarily chosen. Then the number $\mathrm{scd}(w)$ can be computed in time $\mathrm{O}(|w| \times |\Sigma|)$.*

Proof. We illustrate a procedure that computes $\mathrm{scd}(w)$. First of all, we move over the word w from left to right, determining the scopes of the symbols in $\mathrm{alph}(w) := \{b_1, b_2, \dots, b_m\}$, i. e. for each b_i, $1 \le i \le m$, we obtain $(l_i, r_i) := \mathrm{sc}_w(b_i)$. Then we initialise $|w|$ counters $c_1 := 0, c_2 := 0, \dots, c_{|w|} := 0$, and, for each i, $1 \le i \le m$, j, $l_i < j < r_i$, we increment c_j if $w[j] \ne b_i$. Finally, $\mathrm{scd}(w) = \max\{c_i \mid 1 \le i \le n\}$. □

Next, we state that algorithm SolveSWminSCD works correctly and establish its time complexity.

Theorem 3.38. *On an arbitrary input $(w_1, w_2, \dots, w_k) \in (\Sigma^*)^k$, the algorithm* SolveSWminSCD *computes a $w \in w_1 \sqcup\!\sqcup\, w_2 \sqcup\!\sqcup \dots \sqcup\!\sqcup\, w_k$ in time $\mathrm{O}(|w_1 \cdots w_k| \times |\Sigma| \times k^{|\Sigma|})$, and there exists no $w' \in w_1 \sqcup\!\sqcup\, w_2 \sqcup\!\sqcup \dots \sqcup\!\sqcup\, w_k$ with $\mathrm{scd}(w') < \mathrm{scd}(w)$.*

Proof. We shall first prove the correctness of the algorithm SolveSWminSCD, i. e., SolveSWminSCD computes a shuffle word with minimum scope coincidence degree, and then we take a closer look at its runtime.

By definition of the algorithm SolveSWminSCD, it is obvious that the output is a greedy shuffle word of the input words $w_1, w_2, \ldots, w_k$. From Theorem 3.36, we can derive that, in order to prove that w is a shuffle word with minimum scope coincidence degree, it is sufficient to show that the algorithm SolveSWminSCD considers all possible greedy shuffle words and therefore outputs a greedy shuffle word with minimum scope coincidence degree. To this end, let $s := (s_0, s_1, \ldots, s_m)$ be an arbitrary greedy construction sequence that corresponds to the shuffle word $w \in w_1 \sqcup\!\sqcup\, w_2 \sqcup\!\sqcup\, \ldots \sqcup\!\sqcup\, w_k$. We can factorise w into $w = b_1 \cdot \alpha_1 \cdot b_2 \cdot \alpha_2 \cdots b_{|\Sigma|} \cdot \alpha_{|\Sigma|}$, where, for each i, $2 \leq i \leq |\Sigma|$, $b_i \notin \text{alph}(b_1 \cdot \alpha_1 \cdot b_2 \cdot \alpha_2 \cdots b_{i-1} \cdot \alpha_{i-1})$. Let, for each i, $1 \leq i \leq |\Sigma|$, $p_i := |b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}|$. We observe, furthermore, that in the construction sequence s, for each i, $1 \leq i \leq |\Sigma|$, we can associate the element s_{p_i} with the symbol b_i at position $|b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}| + 1$ in w, as the symbol b_i is consumed in the step from s_{p_i} to s_{p_i+1}. More precisely, for each i, $1 \leq i \leq |\Sigma|$, there exists a q_i, $1 \leq q_i \leq k$, such that $s_{p_i} = (v_{p_i,1}, \ldots, v_{p_i,k})$, where $v_{p_i,q_i} = b_i \cdot v_{p_i+1,q_i}$. Moreover, since s is a greedy construction sequence, we know that for each j, $1 \leq j \leq k$, either $v_{p_i,j}[1] \notin \text{alph}(b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1})$ or $v_{p_i,j} = \varepsilon$. Consequently, by definition of the algorithm and since s is a greedy construction sequence, we can conclude that, for each i, $1 \leq i \leq |\Sigma| - 1$, if we pop the tuple $(b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}, (v_{p_i,1}, \ldots, v_{p_i,k}))$ from the stack in line 3, then in iteration q_i of the loop in lines 7 to 12, we push the element $(b_1 \cdot \alpha_1 \cdots b_i \cdot \alpha_i, (v_{p_{i+1},1}, \ldots, v_{p_{i+1},k}))$ on the stack. Moreover, if we pop the tuple $(b_1 \cdot \alpha_1 \cdots b_{|\Sigma|-1} \cdot \alpha_{|\Sigma|-1}, (v_{p_{|\Sigma|-1},1}, \ldots, v_{p_{|\Sigma|-1},k}))$ in line 3, then the tuple $(b_1 \cdot \alpha_1 \cdots b_{|\Sigma|} \cdot \alpha_{|\Sigma|}, (\varepsilon, \varepsilon, \ldots, \varepsilon))$ is pushed on the stack in iteration $q_{|\Sigma|}$ of the loop in lines 7 to 12. As $(\varepsilon, (v_{p_1,1}, \ldots, v_{p_1,k})) = (\varepsilon, (w_1, \ldots, w_k))$ and $(\varepsilon, (w_1, \ldots, w_k))$ is pushed on the stack in line 1, we can conclude that all the tuples $(b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}, (v_{p_i,1}, \ldots, v_{p_i,k}))$, $1 \leq i \leq |\Sigma|$, are pushed on the stack and thus, also popped from it, at some point of the execution of the algorithm. As shown above, this implies that in particular the tuple $(b_1 \cdot \alpha_1 \cdots b_{|\Sigma|} \cdot \alpha_{|\Sigma|}, (\varepsilon, \varepsilon, \ldots, \varepsilon)) = (w, (\varepsilon, \varepsilon, \ldots, \varepsilon))$ is popped from the stack.

Since w has been arbitrarily chosen, we can conclude that each possible greedy shuffle word of the words $w_1, w_2, \ldots, w_k$ is considered by SolveSWminSCD. Thus, SolveSWminSCD computes a shuffle word with minimum scope coincidence degree.

Next, we consider the runtime of SolveSWminSCD. First, we determine the total number of elements that are pushed on the stack during the execution of algorithm SolveSWminSCD. To this end, we note that if we pop an element $(w, (v_1, v_2, \ldots, v_k))$ from the stack in line 3, then in lines 7 to 12 we push at most k elements $(w', (v'_1, v'_2, \dots, v'_k))$ on the stack and, furthermore, $|\operatorname{alph}(w')| = |\operatorname{alph}(w)| + 1$. Hence, we cannot push more than $k^{|\Sigma|}$ elements on the stack. We conclude the proof by estimating the time complexity caused by a single stack element $(w, (v_1, v_2, \dots, v_k))$. The lines 8 to 13 as well as line 3 can each be executed in time $\mathrm{O}(|w \cdot v_1 \cdots v_k|)$. In lines 4 and 6, we have to know the number $\operatorname{scd}(w)$, which, by Proposition 3.37, can be computed in time $\mathrm{O}(|w| \times |\Sigma|)$. Hence, for each element that is pushed on the stack at some point of the algorithm, we require time $\mathrm{O}(|w \cdot v_1 \cdots v_k| \times |\Sigma|) = \mathrm{O}(|w_1 \cdot w_2 \cdots w_k| \times |\Sigma|)$. Since, as explained initially, at most $k^{|\Sigma|}$ elements are pushed on the stack, we can conclude that the total runtime of the algorithm SolveSWminSCD is $\mathrm{O}(|w_1 \cdots w_k| \times |\Sigma| \times k^{|\Sigma|})$. □

By applying the observation from Section 3.4.1.1 – i. e., $\mathrm{SWminSCD}_\Sigma$ can be solved by first deleting all the occurrences of symbols in the input words that are neither leftmost nor rightmost occurrences and then solving $\mathrm{SWminSCD}_\Sigma$ for the reduced input words – we can prove the following result about the time complexity of $\mathrm{SWminSCD}_\Sigma$:

Theorem 3.39. *The problem* $\mathrm{SWminSCD}_\Sigma$ *on an arbitrary input* $(w_1, w_2, \dots, w_k)$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, *can be solved in time* $\mathrm{O}(\max\{|w_1 \cdot w_2 \cdots w_k|, |\Sigma|^2 \times k^{|\Sigma|+1}\})$.

Proof. We observe that we can solve the problem $\mathrm{SWminSCD}_\Sigma$ on an input $w_1, w_2, \dots, w_k$ in the following way. First, we use the algorithm SolveSWminSCD to compute a $w' \in \operatorname{sr}(w_1) \sqcup\!\sqcup \operatorname{sr}(w_2) \sqcup\!\sqcup \dots \sqcup\!\sqcup \operatorname{sr}(w_k)$ with minimum scope coincidence degree. After that, from w', we obtain a $w \in w_1 \sqcup\!\sqcup w_2 \sqcup\!\sqcup \dots \sqcup\!\sqcup w_k$ with $\operatorname{scd}(w) = \operatorname{scd}(w')$ by inserting the symbols into w' that have been removed in order to scope reduce the words $w_1, w_2, \dots, w_k$. By the proof of Lemma 3.25, it is obvious that both, scope reducing the input words and obtaining w from w' by inserting the removed symbols, can be done in time $\mathrm{O}(|w_1 \cdot w_2 \cdots w_k|)$. Since $|\operatorname{sr}(w_1) \cdot \operatorname{sr}(w_2) \cdots \operatorname{sr}(w_k)| = \mathrm{O}(2|\Sigma|k)$, we can conclude that, in case that the input words are scope reduced, the runtime of SolveSWminSCD is $\mathrm{O}(|\Sigma|^2 \times k^{|\Sigma|+1})$. Hence, with the assumption that $|w_1 \cdot w_2 \cdots w_k| = \mathrm{O}(|\Sigma|^2 \times k^{|\Sigma|+1})$, it follows that $\mathrm{SWminSCD}_\Sigma$ can be solved in time $\mathrm{O}(|\Sigma|^2 \times k^{|\Sigma|+1})$. □

3.4.4 A Remark on the Lower Complexity Bound

We have introduced and investigated the problem $\mathrm{SWminSCD}_\Sigma$, i. e., the problem of computing a shuffle word for given input words over the alphabet Σ that is optimal with respect to the scope coincidence degree. We have presented an algorithm solving $\mathrm{SWminSCD}_\Sigma$, which makes use of the fact that there necessarily exists a shuffle word with a minimum scope coincidence degree that can be constructed in a canonical way. Consequently, we obtain an upper bound for the time complexity of this problem, which is dominated by the number of input words and the alphabet size; the length of the input words, on the other hand, is not a crucial factor. Since we have assumed the alphabet to be a constant, the problem is solvable in polynomial time, but the complexity of the problem remains open for the general case, i. e., if the alphabet is considered part of the input (we denote this problem by SWminSCD). We further note that if SWminSCD is NP-complete, then our algorithm is of special interest as it demonstrates the fixed-parameter tractability of this problem, with respect to the parameters of the number of input words and the alphabet size.

Chapter 4

Interlude

This chapter is devoted to a formal study of nondeterministically bounded modulo counter automata (Section 4.1) and nondeterministically initalised automata (Section 4.2). Nondeterministically bounded modulo counter automata are introduced in Chapter 3, where a special version of them, Janus automata, is successfully applied to identify subclasses of pattern languages with a polynomial time membership problem. Nondeterministically initalised automata (IFA) are multi-head automata with restricted nondeterminism. More precisely, an IFA is a deterministic multi-head automaton, the input heads of which are initially nondeterministically distributed on the input word.

4.1 A Formal Study of NBMCA

Regarding NBMCA, two aspects seem to be particularly worth studying: Firstly, all additional resources the automaton is equipped with, namely the counters, are tailored to storing positions in the input word. Secondly, the nondeterminism of NBMCA, which merely allows positions in the input word to be guessed, differs quite substantially from the common nondeterminism of automata, which provides explicit computational alternatives.

If we study the first aspect in more detail, then we can see that it is not really new. For example, even when regarding a common multi-head automaton M with input word w, in every step of a computation every input head *implicitly* represents a number between 1 and $|w|$, namely its position. We can therefore, without a loss of expressive power, turn all input heads into *blind* heads, that cannot scan the input anymore, and successively move an additional reading input head to their positions in order to store the corresponding input symbols in the finite state control. Thus, we can separate the mechanisms of storing positions from the functionality of actually processing the input. This idea is formalised in the model of partially blind multi-head automata (see, e. g., Ibarra and Ravikumar [39]). Another respective variant of multi-head automata are Pebble Automata (see, e. g., Chang et al. [13]), where again only one input head can scan symbols and also place pebbles on the input tape in order to mark certain positions. Finally, automata with sensing heads (see, e. g., Petersen [61]) are multi-head automata where each head is able to sense all other heads currently located at the same input tape cell. Given this similarity between NBMCA and established automata models regarding their emphasis on storing positions in the input word, there is still one difference: the counters of NBMCA are quite limited in their ability to change the positions they represent, since their values can merely be incremented, and their bounds are guessed.

Regarding the second aspect, as mentioned above, standard nondeterminism is designed to provide the automata with computational alternatives. Nevertheless, these automata often use their nondeterminism to actually guess a certain position of the input. For example, as already mentioned in Section 3.2 on page 31, a push-down automaton that recognises $\{ww^R \mid w \in \Sigma^*\}$ needs to perform an unbounded number of guesses even though only one specific position, namely the middle one, of the input needs to be found. Despite this observation, the nondeterminism of NBMCA might be weaker, as it seems to *solely* refer to positions in the input.

In order to understand the character of these novel, and seemingly limited, resources NBMCA can use, in the present section we compare the expressive power of these automata to that of the well-established, and seemingly less restricted, models of multi-head and counter automata. Furthermore, we study some basic decision problems for NBMCA and present a hierarchy result with respect to the number of counters.

The second part of this section is concerned with the role of the finite state control of NBMCA in case that the nondeterminism, i. e., the number of counter resets, is restricted. To this end, we investigate stateless NBMCA. Stateless automata, i. e., automata with only one internal state, have first been considered by Yang et al. [88], where they are compared to P-Systems. This comparison is appropriate, as it is a feature of P-Systems that they are not controlled by a finite state control. Ibarra et al. [38] and Frisco and Ibarra [24] mainly investigate stateless multi-head automata, whereas Ibarra and Eğecioğlu [37] consider stateless counter machines. In Kutrib et al. [47] stateless restarting automata are studied. Intuitively, the lack of states results in a substantial loss of possible control mechanisms for the automaton. For instance, the task to recognise exactly the singleton language $\{a^k\}$ for some fixed constant k, which is easily done by any automaton with states, suddenly seems difficult, as we somehow need to count k symbols without using any states. In [38] an example of a stateless multi-head automaton that recognises $\{a^k\}$ can be found.

The question of whether or not states are really necessary for a model, i. e., whether it is possible to simulate automata by their stateless counterparts, is probably the most fundamental question about stateless automata. Obviously, the models of DFA and NFA degenerate if the number of states is restricted to at most one. On the other hand, we know that the power of nondeterministic PDA is not dependent on the number of states and, thus, every PDA can be turned into a PDA with only a single state (see, e. g., Hopcroft and Ullman [33]). Intuitively, the pushdown store compensates for the loss of states. Regarding *deterministic* pushdown automata, we find a different situation; here, the expressive power strictly increases with the number of states (see, e. g., Harrison [29]).

Our first main result shows that every NBMCA with states can be turned into an equivalent one without states. Hence, the loss of the finite state control does not lead to a reduced expressive power of the model. NBMCA are tailored to a restriction of nondeterminism, since we can simply limit the number of possible resets of counters. If the number of resets for an NBMCA M is bounded by k, then this means that in every computation of M the first k resets for every counter actually reset the counters and all further resets are ignored, i. e., after a counter is reset for at least k times, every further reset leaves the counter unchanged. Furthermore, we focus on stateless NBMCA with a single one-way input head and only one counter, the resets of which are restricted as defined above. For this class of automata, we establish our second main result, which, on the one hand, states that there exist languages that can be recognised by stateless NBMCA with k but not with $k - 1$ resets and, on the other hand, that there exist languages that can be recognised with k but not with $k + 1$ resets. Hence, for this class of NBMCA, a non-existent finite state control can turn nondeterminism into a handicap with respect to the expressive power.

4.1.1 Expressive Power

In this section we investigate the expressive power of NBMCA in comparison to multi-head automata and counter automata. Since an NBMCA can be regarded as a finite state control with additional resources the model is still sufficiently similar to classical multi-head automata and counter automata, so that a comparison in terms of expressive power is appropriate. On the other hand, NBMCA show non-standard restrictions that seem difficult to classify in terms of the classical models and may be worthwhile to investigate.

One question that springs to mind is whether or not the fact that the counters of an NBMCA can only count modulo a certain bound is a restriction, in terms of expressive power, compared to automata with counters that can be decremented as well. As already described in Section 4.1, another interesting aspect of NBMCA is their specific use of nondeterminism and it is not obvious whether this special nondeterminism is somehow weaker than nondeterminism defined via a transition relation.

In order to address these questions, we now study the problem of whether classical multi-head automata and counter automata can simulate NBMCA and vice versa. It is almost obvious that NBMCA can be simulated by these other models as NBMCA can be interpreted as multi-head automata (or counter automata, respectively) with further restrictions. So multi-head and counter automata intuitively seem to be more powerful. In contrast, the converse question, i. e., whether or not arbitrary multi-head and counter automata, and particularly their unrestricted nondeterminism, can be simulated by NBMCA, is more challenging.

In our comparisons we have to make a restriction to the model of counter automata. The problem is that the counters of counter automata are unrestricted and can, thus, store arbitrarily large values. This yields an immense expressive power, and in fact even two counters are sufficient to simulate Turing machines (cf. Minsky [52]). As multi-head automata and NBMCA are restricted to exclusively operate on the original input, they are strictly weaker than Turing machines. Hence, it seems uninteresting to compare the model of a counter automaton to NBMCA or multi-head automata in terms of expressive power. Since we are particularly interested in the special restrictions of NBMCA and their impact on the expressive power, we still consider it worthwhile to compare NBMCA to a model that also uses counters, but without the modulo restriction and with classical nondeterminism. To this end, we slightly alter the definition of counter automata. More precisely, we define counter automata the counters of which are bounded by a function of the input length:

Definition 4.1. Let $k \in \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$. An *$f(n)$-bounded nondeterministic* or *deterministic two-way counter automaton with k counters (*$2\text{CNFA}_{f(n)}(k)$ *or* $2\text{CDFA}_{f(n)}(k)$ *for short)* is a nondeterministic (or deterministic, respectively) two-way automaton with k counters that can be incremented and decremented within the bounds of 0 and $f(n)$, where n is the current input length. It can be checked whether a counter stores 0, $f(n)$ or a value in between.

We shall mainly consider the models of $2\text{CNFA}_{f(n)}(k)$ and $2\text{CDFA}_{f(n)}(k)$, where $f(n) = n$, denoted by $2\text{CNFA}_n(k)$ and $2\text{CDFA}_n(k)$. We note that we can interpret a $2\text{CNFA}_n(k)$ as an NBMCA that has static counter bounds equal to the input length and that can decrement counters as well. Furthermore, the transition function is nondeterministic. Hence, the model of a $2\text{CNFA}_n(k)$ can be regarded as an unrestricted version of an NBMCA.

We proceed with a basic observation that shall prove useful for our further results.

Proposition 4.2. *For every $k \in \mathbb{N}$, $\mathcal{L}(\text{2CNFA}_n(k)) \subseteq \mathcal{L}(\text{2NFA}(k+1))$.*

This proposition can be easily comprehended by observing that we can use k input heads of the 2NFA$(k+1)$ in such a way that they ignore the input, thus, they behave exactly like counters that are bounded by the input length.

In the following we show that for every $k \in \mathbb{N}$ there exists a $k' \in \mathbb{N}$ such that an arbitrary NBMCA(k) can be simulated by a 2CNFA$_n(k')$. For this simulation, we apply the following idea. For each counter of the NBMCA(k) we simply use two counters of the 2CNFA$_n(k')$, one of which stores the current counter value and the other stores the distance between counter value and counter bound. With Proposition 4.2 we can extend this simulation to 2NFA$(k'+1)$ as well.

Lemma 4.3. *For every $k \in \mathbb{N}$ and for every $M \in$ NBMCA(k), there exists an $M' \in$ 2CNFA$_n(2k)$ and an $M'' \in$ 2NFA$(2k+1)$ with $L(M) = L(M') = L(M'')$.*

Proof. Let $M \in$ NBMCA(k) be arbitrarily chosen. We show how an $M' \in$ 2CNFA$_n(2k)$ can simulate M. The input head of M' is used in exactly the same way M uses its input head. Hence, it is sufficient to illustrate how M' simulates the modulo counters of M. The idea is that the modulo counter i, $1 \leq i \leq k$, of M is simulated by the counters $2i-1$ and $2i$ of M', i. e., counter $2i-1$ represents the counter value and counter $2i$ represents the counter bound of the modulo counter i of M. A reset of modulo counter i is simulated by M' in the following way. First, both counters $2i-1$ and $2i$ of M' are decremented to 0. Then counter $2i-1$ is incremented and after every increment, M' can nondeterministically guess whether it keeps on incrementing or it stops. If the counter reaches value n, it must stop. The value counter $2i-1$ stores after that procedure is interpreted as the new counter bound.

The actual counting of the modulo counter i of M is then simulated in the following way. Whenever M increments counter i, then M' increments counter $2i$ and decrements counter $2i-1$. When counter $2i-1$ reaches 0, then this is interpreted as reaching the counter bound. In order to enable a new incrementing cycle of the modulo counter i of M from 0 to its counter bound, the counters $2i-1$ and $2i$ simply change their roles and can then be used again in the same way.

Using Proposition 4.2 we can conclude that there exists an $M'' \in$ 2NFA$(2k+1)$ with $L(M'') = L(M')$.

□

Now Lemma 4.3 directly implies the following result.

Theorem 4.4. *For every $k \in \mathbb{N}$,*

- $\mathcal{L}(\text{NBMCA}(k)) \subseteq \mathcal{L}(\text{2CNFA}_n(2k))$ *and*
- $\mathcal{L}(\text{NBMCA}(k)) \subseteq \mathcal{L}(\text{2NFA}(2k+1))$.

Next, we investigate the problem of whether NBMCA can be used to simulate $\text{2CNFA}_n(k)$ and $\text{2NFA}(k)$. It turns out that this is possible, but the constructions are a bit more involved since we have to simulate input heads that can be moved in both direction (counters that may count in both directions, respectively) by the restricted counters of NBMCA. Furthermore, the nondeterminism of these models has to be handled by the special nondeterminism of NBMCA.

It is sufficient to show how $\text{2NFA}(k)$ can be simulated by NBMCA and then use Proposition 4.2 to conclude that we can simulate $\text{2CNFA}_n(k)$ by NBMCA as well. In the following simulation, the NBMCA uses one modulo counter in order to store the positions of two input heads of the $\text{2NFA}(k)$, i. e., the counter value and the counter bound each represent an input head position. A step of the $\text{2NFA}(k)$ is then simulated by first moving the input head of the NBMCA successively to all these positions stored by the counters and record the scanned input symbols in the finite state control. After that, all these positions stored by the counters must be updated according to the transition function of the $\text{2NFA}(k)$. Since counter values cannot be decremented and counter bounds cannot be changed directly, this updating step requires some technical finesse. Furthermore, we need an additional counter which is also used in order to simulate the possible nondeterministic choices of the $\text{2NFA}(k)$.

Lemma 4.5. *For every $k \in \mathbb{N}$ and for every $M \in \text{2NFA}(k)$, there exists an $M' \in \text{NBMCA}(\lceil \frac{k}{2} \rceil + 1)$ with $L(M) = L(M')$. For every $\widehat{M} \in \text{2CNFA}_n(k)$ there exists an $\widehat{M}' \in \text{NBMCA}(\lceil \frac{k+1}{2} \rceil + 1)$ with $L(\widehat{M}) = L(\widehat{M}')$.*

Proof. We first show how an $\text{NBMCA}(\lceil \frac{k}{2} \rceil + 1)$ M' can be constructed that, on any input $w := a_1 \cdot a_2 \cdots a_n$, $a_i \in \Sigma$, $1 \leq i \leq n$, simulates M. For the sake of convenience, we assume that k is even, the case that k is odd can be handled analogously. The general idea is that the first $\lceil \frac{k}{2} \rceil$ modulo counters of M' are used to store the positions of the k input heads of M. Thus, one modulo counter of M' has to store the positions of two input heads of M, i. e., one position is represented by the counter value and the other one by the counter bound of the modulo counter. In addition to that, M' has an auxiliary counter that is used to temporarily store data. More precisely, if M is able to perform the move $[q, h_1, \ldots, h_k] \vdash_{M,w} [p, h'_1, \ldots, h'_k]$, then M' can perform a sequence of moves $[q, 0, (h_1, h_2), \ldots, (h_{k-1}, h_k), (0, c)] \vdash^*_{M',w}$ $[p, 0, (h'_1, h'_2), \ldots, (h'_{k-1}, h'_k), (0, c')]$, for some c, c', $1 \leq c, c' \leq n$. The role of the counter bounds c and c' of the auxiliary counter are not important right now and shall be explained later on.

We shall now informally explain the basic idea of how a step of M can be simulated by a sequence of moves of M' and formally prove all the technical details afterwards. A transition of M depends on k input symbols and a state. Therefore, M' records in its finite state control all the symbols at the positions determined by the counter values and counter bounds. More precisely, if h_1 and h_2 are the counter value and counter bound of the first counter and M' is in state q right now, then M' moves its input head to position h_1, changes into state $q_{a_{h_1}}$, moves the input head to position h_2 and changes into state $q_{a_{h_1},a_{h_2}}$. The same procedure is applied to all counters $2, 3, \dots, \lceil \frac{k}{2} \rceil$ until M finally reaches a state $q_{a_{h_1},a_{h_2},\dots,a_{h_k}}$. We note that in order to prove that all these steps can be carried out by M', it is sufficient to show that M' can perform the following sequences of moves:

$$\widetilde{c} \vdash^*_{M',w} [q_{a_{h_{2i-1}}}, 0, (h_1, h_2), \dots, (h_{2i-1}, h_{2i}), \dots, (h_{k-1}, h_k), (0, c')] \,, \tag{4.1}$$

$$\widetilde{c} \vdash^*_{M',w} [q_{a_{h_{2i}}}, 0, (h_1, h_2), \dots, (h_{2i-1}, h_{2i}), \dots, (h_{k-1}, h_k), (0, c')] \,, \tag{4.2}$$

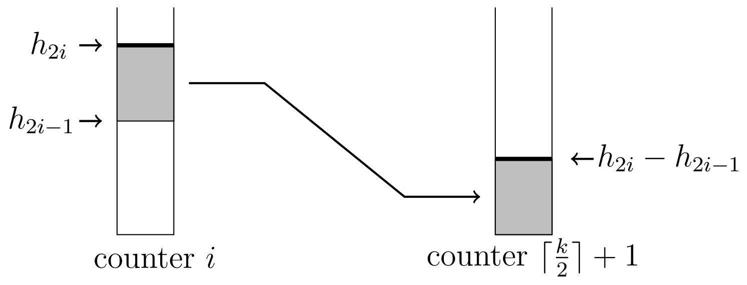
where $\widetilde{c} := [q, 0, (h_1, h_2), \dots, (h_{k-1}, h_k), (0, c'')]$ is an arbitrary configuration of M' and $0 \leq c', c'' \leq n$.

The next step is now determined by q, the symbols $a_{h_1}, a_{h_2}, \dots, a_{h_k}$ and δ, the transition function of M, which is possibly nondeterministic and can choose one of several possible steps. However, it is possible to transform an arbitrary 2NFA(k) into a 2NFA(k), where for every nondeterministic step there are exactly two possible choices. This can be done by substituting a nondeterministic transition with $l > 2$ choices by $l - 2$ transitions that have exactly two nondeterministic choices. Obviously, this requires $l - 2$ new states and, thus, the number of states increases, but this is not a problem as the number of states does not play any role in the statement of the lemma. In order to simulate this nondeterministic choice between two options, M' resets counter $\lceil \frac{k}{2} \rceil + 1$ and checks whether or not the newly guessed counter bound equals 0, which is only the case if the counter message is $\mathtt{t}_1$ right after reseting it. The transition function of M' can then be defined such that the first option of the two possible transitions of M is carried out if 0 is guessed as new counter bound and the second option is chosen otherwise. We assume that the transition chosen by M is $(q, a_{h_1}, \dots, a_{h_k}) \rightarrow_\delta (p, d_1, \dots, d_k)$, where p is the new state and $(d_1, \dots, d_k)$ are the input head movements, so next all counter values and counter bounds need to be updated according to $(d_1, \dots, d_k)$. To this end, M' changes into state $p_{d_1,\dots,d_k}$ where the counter value of counter 1 is changed to $h_1 + d_1$ and after that M changes into state $p_{d_2,\dots,d_k}$. Next, the counter bound of counter 1 is changed to $h_2 + d_2$ while the state changes into $p_{d_3,\dots,d_k}$ and so on. Eventually, M' reaches state p and the configurations of the counters are $(h_1 + d_1, h_2 + d_2), \dots, (h_{k-1} + d_{k-1}, h_k + d_k)$. Again, in order to prove that this procedure can be carried out by M', it is sufficient to show that M' can perform the following sequences of moves:

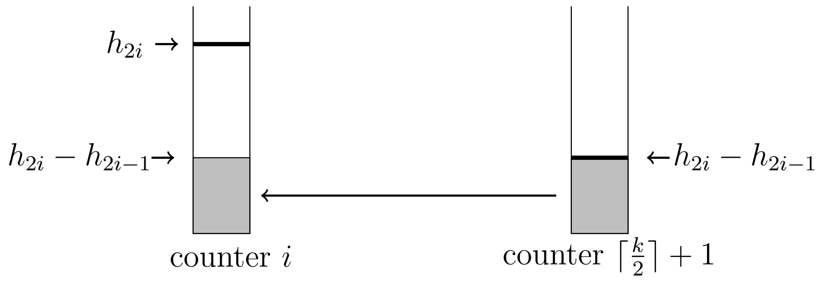
$$\widetilde{c} \vdash^*_{M',w} [q, 0, (h_1, h_2), \dots, (h_{2i-1} + d, h_{2i}), \dots, (h_{k-1}, h_k), (0, c')] \,, \tag{4.3}$$
$$\widetilde{c} \vdash^*_{M',w} [q, 0, (h_1, h_2), \dots, (h_{2i-1}, h_{2i} + d'), \dots, (h_{k-1}, h_k), (0, c')] \,, \tag{4.4}$$

where $\widetilde{c} := [q, 0, (h_1, h_2), \dots, (h_{k-1}, h_k), (0, c'')]$ is an arbitrary configuration of M', $0 \leq c', c'' \leq n$, $d, d' \in \{1, -1\}$, $h_{2i-1} + d \leq h_{2i}$ and $h_{2i-1} \leq h_{2i} + d'$.

In order to conclude the proof, it remains to show that the transition function δ' of M' can be defined in a way such that the sequences of moves (4.1) - (4.4) can be performed. We begin with the sequences of moves (4.1) and (4.2). First, M' resets counter $\lceil \frac{k}{2} \rceil + 1$ and then increments counter $\lceil \frac{k}{2} \rceil + 1$ and counter i simultaneously. If these two counters reach their counter bounds at exactly the same time, then we can conclude that the newly guessed counter bound of counter $\lceil \frac{k}{2} \rceil + 1$ equals $h_{2i} - h_{2i-1}$ and we proceed. In case that a different counter bound is guessed, M' changes into a non-accepting trap state. This procedure is illustrated by the following diagram.

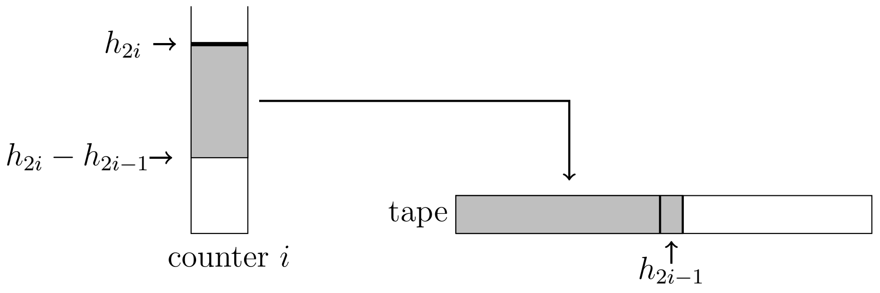

$h_{2i} \rightarrow$
$h_{2i-1} \rightarrow$
$\leftarrow h_{2i} - h_{2i-1}$
counter i
counter $\lceil \frac{k}{2} \rceil + 1$

Counters i and $\lceil \frac{k}{2} \rceil + 1$ are then set back to 0 by incrementing them once more. Then they are incremented simultaneously until counter $\lceil \frac{k}{2} \rceil + 1$ reaches its counter bound. After this step, counter i stores value $h_{2i} - h_{2i-1}$ as pointed out by the following illustration.


$h_{2i} \rightarrow$
$h_{2i} - h_{2i-1} \rightarrow$
$\leftarrow h_{2i} - h_{2i-1}$
counter i
counter $\lceil \frac{k}{2} \rceil + 1$

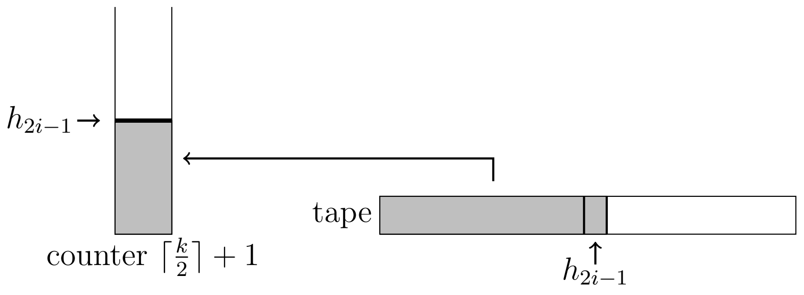
Now it is possible to increment counter i and simultaneously move the input head to the right until counter i reaches its bound of h_{2i}. Clearly, this happens after

h_{2i-1} increments, so the input head is then located at position h_{2i-1} of the input tape.

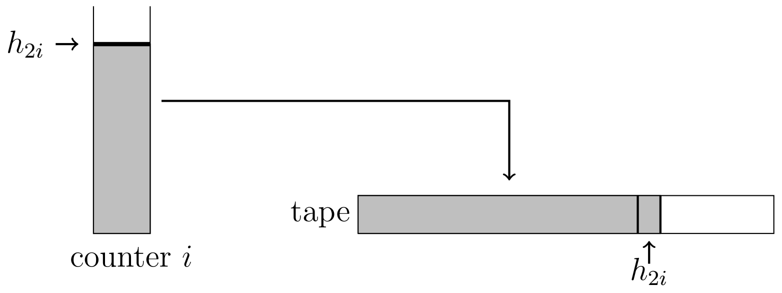


Now, in case of (4.1), M' changes into state $q_{a_{h_{2i-1}}}$ and sets the value of counter i back to 0. Finally, by moving the input head from position h_{2i-1} to the left until it reaches the left endmarker and simultaneously incrementing counter i, we set the input head back to position 0 and the counter value of counter i back to h_{2i-1}. Furthermore, we set the value of counter $\lceil \frac{k}{2} \rceil + 1$ back to 0.

In case of (4.2), a few more steps are required. We recall that the input head is located at position h_{2i-1}. M resets counter $\lceil \frac{k}{2} \rceil + 1$ and checks whether or not the new counter bound equals h_{2i-1}. This is done by moving the input head to the left and simultaneously incrementing counter $\lceil \frac{k}{2} \rceil + 1$.



Next, we set the value of counter i back to 0 and then increment it until the counter bound of h_{2i} is reached and simultaneously move the input head to the right. Obviously, the input head is then located at position h_{2i}. Thus, M' can change into state $q_{a_{h_{2i}}}$.



As counter $\lceil \frac{k}{2} \rceil + 1$ has a counter bound of h_{2i-1}, we can easily set the value of counter i back to h_{2i-1}. Finally, the input head is moved back to position 0 and the value of counter $\lceil \frac{k}{2} \rceil + 1$ is set back to 0.

Next, we consider case (4.3). If $d = 1$, we can simply increment counter i. If, on the other hand, $d = -1$, we first move the input head to position h_{2i-1} in the same way we did in case (4.1), and then one step to the left, i. e., to position $h_{2i-1}+d$. Now we can set counter i to 0, and then increment it and simultaneously move the input head to the left until it reaches the left endmarker. After that step, counter i stores value $h_{2i-1} + d$.

In order to implement case (4.4), we first move the input head to position h_{2i} in the same way we did in case (4.2), i. e., we first move it to position h_{2i-1} as done for cases (4.1) and (4.2) and then, by reseting counter $\lceil \frac{k}{2} \rceil + 1$, we store h_{2i-1} in the counter bound of counter $\lceil \frac{k}{2} \rceil + 1$ and finally use counter i in order to move the input head to position h_{2i}. Next, we move the input head to position $h_{2i} + d'$, reset counter i and, by moving the input head back to position 0, check whether $h_{2i} + d'$ is guessed as new counter bound. Finally, we use counter $\lceil \frac{k}{2} \rceil + 1$, which has a counter bound of h_{2i-1}, to set the counter value of counter i back to h_{2i-1}.

It remains to show how we can handle the cases where we have $h_{2i-1} = h_{2i}$ and either h_{2i-1} should be incremented or h_{2i} should be decremented. Clearly, this is not possible, so in this case we simply change the roles of the counter bound and counter value to avoid this problem. If we do this, we need to store in the finite state control that from now on the counter value stores the position of input head $2i$ and the counter bound stores the position of input head $2i - 1$.

This shows that M' can perform the sequences of moves (4.1) - (4.4), which implies that M' can simulate M in the way described at the beginning of this proof.

It remains to show that an arbitrary $\widehat{M} \in \text{2CNFA}_n(k')$, $k' \in \mathbb{N}$, can be simulated by some $\widehat{M}' \in \text{NBMCA}(\lceil \frac{k'+1}{2} \rceil + 1)$. This can be directly concluded from Proposition 4.2. □

Before we proceed to the main result of this section, we wish to briefly discuss some particularities of the proof of the above lemma: Originally, the counters of NBMCA were designed for a special application (see Chapter 3). In this context, the purpose of the counter bound is to store the length of a factor or a position of the input and the counter value is incremented in order to move the input head over a whole factor or to a distinct position of the input. So intuitively, every counter bound can be interpreted as an anchor on the input tape and the functionality of the counters is merely a mechanism to move the input head to these anchored positions. However, the result above is obtained by showing that the counters of NBMCA can be used in a completely different and rather counter-intuitive way. In the proof of Lemma 4.5 it is vital to overcome the strong dependency between a counter value and its counter bound such that both of them can be fully exploited as mere storages for input positions that can be arbitrarily updated. The immediate result of this is then that the counter value as well as the counter bound are each as powerful as an input head.

From Lemma 4.5, we conclude the following theorem.

Theorem 4.6. *For every $k \in \mathbb{N}$,*

- $\mathcal{L}(\text{2NFA}(k)) \subseteq \mathcal{L}(\text{NBMCA}(\lceil \frac{k}{2} \rceil + 1))$ *and*
- $\mathcal{L}(\text{2CNFA}_n(k)) \subseteq \mathcal{L}(\text{NBMCA}(\lceil \frac{k+1}{2} \rceil + 1))$.

By definition, NBMCA and 2NFA are different in several regards. The input heads of 2NFA can be freely moved on the tape whereas the counter values of NBMCA can only be incremented from 0 to the counter bound, which in turn can only be changed by nondeterministically guessing a new one and, thus, losing the old bound. Furthermore, the nondeterministic transitions of 2NFA are defined by the scanned input symbols and the state, whereas the deterministic step of an NBMCA is merely defined by the state and the predicate whether the counter values have reached their bounds, i. e., neither counter bounds nor counter values directly control the automaton. Given these substantial differences between the models, the previous result seems surprising.

In the following corollary, NL denotes the complexity class of nondeterministic logarithmic space. The characterisation $\bigcup_{k \in \mathbb{N}} \mathcal{L}(\text{2NFA}(k)) = \text{NL}$ is a well know fact (see Hartmanis [30]) and the other equalities follow from Theorems 4.4 and 4.6.

Corollary 4.7.

$$\bigcup_{k \in \mathbb{N}} \mathcal{L}(\text{2CNFA}_n(k)) = \bigcup_{k \in \mathbb{N}} \mathcal{L}(\text{2NFA}(k)) = \bigcup_{k \in \mathbb{N}} \mathcal{L}(\text{NBMCA}(k)) = \text{NL} \, .$$

In a simulation of 2NFA by NBMCA as used in the proof of Lemma 4.5, the counters of the NBMCA can be employed quite economically, i. e., we use one counter of the NBMCA to handle two input heads of the 2NFA. Intuitively, a simulation is also possible, and most likely much simpler, if we allow one counter of the NBMCA per input head of the 2NFA. However, we shall see that this tight use of the modulo counters is worth the effort, as it allows us to prove a hierarchy result on the class NBMCA. This insight follows from the classical result in automata theory that adding an input head to a 2NFA(k) strictly increases its expressive power (see Holzer et al. [31] for a summary and references of the original papers). More precisely, the classes $\mathcal{L}(\text{2NFA}(k))$, $k \in \mathbb{N}$, describe a hierarchy with respect to k:

Theorem 4.8 (Monien [53])**.** *For every $k \in \mathbb{N}$, $\mathcal{L}(\text{2NFA}(k)) \subset \mathcal{L}(\text{2NFA}(k+1))$.*

Theorem 4.8 together with Theorems 4.4 and 4.6 can be used to show a similar result on NBMCA with respect to the number of counters. However, we obtain a hierarchy with a gap, i. e., we can only show that NBMCA($k + 2$) are strictly more powerful than NBMCA(k).

Corollary 4.9. *For every $k \in \mathbb{N}$, $\mathcal{L}(\text{NBMCA}(k)) \subset \mathcal{L}(\text{NBMCA}(k + 2))$.*

Proof. By Theorems 4.4, 4.8 and 4.6, we know that, for every $k \in \mathbb{N}$,

- $\mathcal{L}(\text{NBMCA}(k)) \subseteq \mathcal{L}(\text{2NFA}(2k + 1))$,
- $\mathcal{L}(\text{2NFA}(2k + 1)) \subset \mathcal{L}(\text{2NFA}(2k + 2))$ and
- $\mathcal{L}(\text{2NFA}(2k + 2)) \subseteq \mathcal{L}(\text{NBMCA}(k + 2))$.

Consequently, NBMCA(k) $\subset$ NBMCA($k + 2$). □

In this section, we investigated the expressive power of NBMCA in relation to classical multi-head automata. Next, we shall take a closer look at decidability properties of NBMCA.

4.1.2 Decidability

In this section, we investigate the decidability of the emptiness, infiniteness, universe, equivalence, inclusion and disjointness problem with respect to languages given by NBMCA. All these problems are undecidable even for 1DFA(2) (cf., Holzer et al. [31]) and since NBMCA can simulate 1DFA(2) (Lemma 4.5) these negative results carry over to the class of NBMCA. However, it is a common approach to further restrict automata models with undecidable problems in order to obtain subclasses with decidable problems (see, e.g., Ibarra [36]). One respective option is to require the automata to be reversal bounded. The following definitions are according to [36].

In a computation of some two-way automaton model, an *input head reversal* describes the situation that the input head is moved a step to the right (to the left, respectively) and the last time it has been moved it was moved a step to the left (to the right, respectively), so it reverses directions. A *counter reversal* is defined in a similar way just with respect to the increments and decrements of a counter. We say that an automaton is *input head reversal bounded* or *counter reversal bounded* if there exists a constant m such that, for every accepting computation, the number of input head reversals (counter reversals, respectively) is at most m.

We now formally define classes of reversal bounded automata.

Definition 4.10. For all $m_1, m_2, k \in \mathbb{N}$, we define (m_1, m_2)-REV-CNFA(k) and (m_1, m_2)-REV-CDFA(k) to be the class of 2CNFA(k) and 2CDFA(k), respectively, that perform at most m_1 input head reversals and every counter performs at most m_2 counter reversals in every accepting computation.

For the above defined reversal bounded automata, there is no need anymore to distinguish between the one-way and the two-way case as this aspect is covered by the number of input head reversals, i. e., one-way automata coincide with those that are input head reversal bounded by 0. Next, we cite a classical result about reversal bounded counter automata:

Theorem 4.11 (Ibarra [36]). *The emptiness, infiniteness and disjointness problems for the class* (m_1, m_2)-REV-CNFA(k) *are decidable. The emptiness, universe, infiniteness, inclusion, equivalence and disjointness problem for the class* (m_1, m_2)-REV-CDFA(k) *are decidable.*

Our goal is to transfer these results to reversal bounded NBMCA. With respect to NBMCA, a counter reversal is interpreted as an increment of the counter in case that it has already reached its counter bound. Furthermore, we need to bound the number of resets as well.

Definition 4.12. For all $m_1, m_2, l, k \in \mathbb{N}$, let (m_1, m_2, l)-REV-NBMCA(k) denote the class of NBMCA(k) that perform at most m_1 input head reversals, at most m_2 counter reversals and resets every counter at most l times in every accepting computation.

We can show that an (m_1, m_2, l)-REV-NBMCA(k) can be simulated by an (m'_1, m'_2)-REV-CNFA(k'), which implies that the results of Theorem 4.11 carry over to (m_1, m_2, l)-REV-NBMCA(k).

Lemma 4.13. *For every* $M \in (m_1, m_2, l)$-REV-NBMCA(k)*, there exists an* $M' \in (m'_1, m'_2)$-REV-CNFA$(4k)$ *such that* $L(M) = L(M')$.

Proof. Let $M \in (m_1, m_2, l)$-REV-NBMCA(k). We first recall that, by Lemma 4.3, an NBMCA(k) can be simulated by a CNFA$_n(2k)$. Furthermore, in this simulation, the input head of the CNFA$_n(2k)$ is used in the same way as the input head of NBMCA(k), and every counter reversal and reset of a modulo counter of the NBMCA(k) causes the two corresponding counters of the CNFA$_n(2k)$ to perform a reversal. Consequently, in the simulation of an NBMCA(k) by a CNFA$_n(2k)$, the input head reversals of the NBMCA(k) are preserved and the counter reversals of the CNFA$_n(2k)$ are bounded in the number of counter resets and counter reversals of the NBMCA(k). We conclude that there exists an

(m'_1, m'_2)-REV-CNFA$_n(2k)$ M', i. e., an (m'_1, m'_2)-REV-CNFA$(2k)$ the counters of which are bounded by the input length, with $L(M) = L(M')$. This M' can be simulated by an $(m'_1 + 2, m'_2 + 1)$-REV-CNFA$(4k)$ M'' in the following way. At the beginning of the computation M'' increments the first $2k$ counters to the input length by moving the input head over the input. After that step, for every i, $1 \leq i \leq 2k$, counter i stores the input length n and counter $i + 2k$ stores 0. Counters i and $i + 2k$ of M'' can simulate counter i of M' by decrementing (or incrementing) counter i and incrementing (or decrementing, respectively) counter $i + 2k$ for every increment (or decrement, respectively) of counter i of M'. Hence, when counter i of M'' reaches 0, then counter i of M' reaches n and when counter $i + 2k$ of M'' reaches 0, then counter i of M' reaches 0 as well. This requires two additional input head reversals and an additional counter reversal for the first k counters. □

Corollary 4.14. *The emptiness, infiniteness and disjointness problem for the class* (m_1, m_2, l)-REV-NBMCA *are decidable.*

In the following, we study the question of whether it is possible to ease the strong restriction of (m_1, m_2, l)-REV-NBMCA a little without losing the decidability results. More precisely, we investigate the decidability of the emptiness, infiniteness, universe, equivalence, inclusion and disjointness problems for the class (m, ∞, l)-REV-NBMCA, i. e., the number of counter reversals is not bounded anymore. We shall explain our motivation for this in a bit more detail. To this end we cite the following result.

Theorem 4.15 (Ibarra [36])**.** *The emptiness, infiniteness, universe, equivalence, inclusion and disjointness problems are undecidable for* $(1, \infty)$-REV-CDFA(1).

Consequently, with respect to CDFA (and, thus, CNFA) the typical decision problems remain undecidable when the restriction on the counter reversals is abandoned. However, regarding (m, ∞, l)-REV-NBMCA we observe a slightly different situation. While a counter reversal of a counter automaton can happen anytime in the computation and for any possible counter value, a counter reversal of an NBMCA strongly depends on the current counter bound, i. e., as long as a counter is not reset, all the counter reversals of that counter happen at exactly the same counter value. So while for $(1, \infty)$-REV-CDFA the counters are not restricted at all, the modulo counters of (m, ∞, l)-REV-NBMCA can still be considered as restricted, since the number of resets is bounded. Intuitively, this suggests that the restrictions of (m, ∞, l)-REV-NBMCA are still stronger than the restrictions of $(1, \infty)$-REV-CDFA.

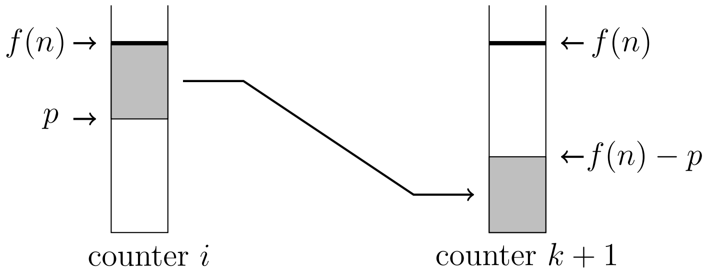
In order to answer the question about the decidability of the above mentioned problems with respect to (m, ∞, l)-REV-NBMCA, we first find another way to simulate counter automata by NBMCA. The simulation that can be used to prove Lemma 4.5 has the advantage of requiring a relatively small number of modulo counters, but pays the price of a large number of input head reversals and counter resets. In fact, in the simulation of Lemma 4.5 even if the 2CNFA(k) is input head reversal bounded and counter reversal bounded, the number of counter resets as well as the input head reversals of the NBMCA($\lceil \frac{k+1}{2} \rceil + 1$) are not necessarily bounded anymore. Hence, it is our next goal to find a simulation of counter automata by NBMCA that preserves the number of input head reversals and requires only a constant number of resets. Before we can give such a simulation, we need the following technical lemma, which shows that we can transform an arbitrary 2CNFA$(k)_{f(n)}$ or 2CDFA$_{f(n)}(k)$ into an equivalent 2CNFA$_{f(n)}(k)$ (or 2CDFA$_{f(n)}(k)$, respectively) that only reverses counters at value 0 or $f(n)$.

Lemma 4.16. *For every $M \in$ 2CNFA$_{f(n)}(k)$ (or $M \in$ 2CDFA$_{f(n)}(k)$) there exists an $M' \in$ 2CNFA$_{f(n)}(k+2)$ (or $M' \in$ 2CDFA$_{f(n)}(k+2)$, respectively) such that $L(M) = L(M')$ and every counter of M' reverses from decrementing to incrementing only at value 0 and from incrementing to decrementing only at value $f(n)$. Furthermore, for every $w \in \Sigma^*$, if M reverses the input head m times and reverses every counter at most q times on input w, then M' reverses the input head m times and reverses every counter at most $2kq$ times on w.*

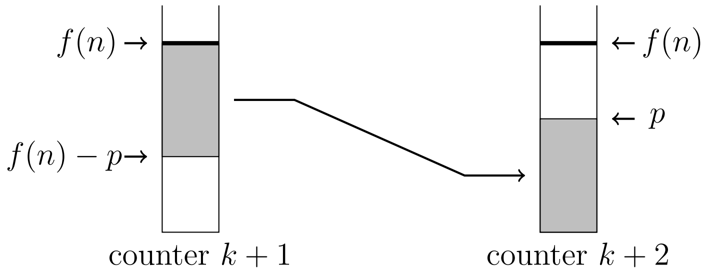
Proof. Let $M \in$ 2CNFA$_{f(n)}(k)$ (or $M \in$ 2CDFA$_{f(n)}(k)$), $k \in \mathbb{N}$, be arbitrarily chosen. We shall show how M can be changed such that all counters only reverse at value 0 or $f(n)$. All the following constructions are completely deterministic, so determinism of M is preserved. We can assume that, for every counter, M stores in its finite state control whether this counter is in incrementing or decrementing mode. Thus, for any counter, M can identify a change from incrementing to decrementing and vice versa. Furthermore, by using additional states, every 2CNFA$_{f(n)}(k)$ (or 2CDFA$_{f(n)}(k)$, respectively) can be transformed into one that increments or decrements at most one counter in any transition. Hence, we can assume M to have this property.

We define how an $M' \in$ 2CNFA$_{f(n)}(k+2)$ (or $M' \in$ 2CDFA$_{f(n)}(k+2)$, respectively), the counters of which reverse only at values 0 or $f(n)$, can simulate M. In this simulation, the counters 1 to k of M' exactly correspond to the counters 1 to k of M, and the counters $k+1$ and $k+2$ of M' are auxiliary counters. We now consider a situation where counter i of M is decremented from value p to $p-1$ and this decrement constitutes a reversal. We show how M' simulates this step such that its counter i reverses at $f(n)$. The main idea is to use the auxiliary counters $k+1$ and $k+2$ to temporarily store values, but, since these counter can only reverse at values 0 or $f(n)$ as well, the construction is not completely straightforward.

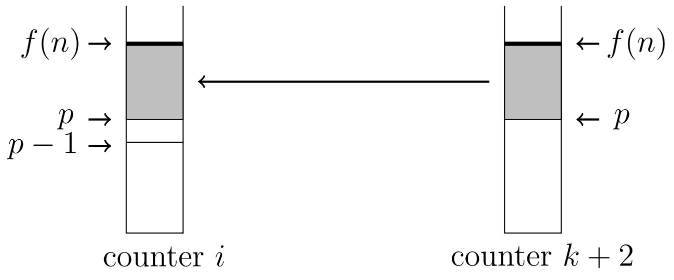
M' simulates the above described step in the following way. Instead of decrementing counter i from p to $p-1$, M' performs further dummy increments until value $f(n)$ is reached and simultaneously increments counter $k+1$. Hence, counter $k+1$ stores exactly $f(n)-p$ when counter i reaches $f(n)$. This situation is illustrated below.



Next, we increment counters $k+1$ and $k+2$ simultaneously until counter $k+1$ reaches $f(n)$. This implies that counter $k+2$ stores now p.



We can now decrement counter i, thus performing the reversal at value $f(n)$, and simultaneously increment counter $k+2$ until it reaches $f(n)$. After these steps, counter i stores value p again, but is now in decrementing mode. M finally decrements counter i to value $p-1$.



Both counters $k+1$ and $k+2$ store now value $f(n)$ and therefore are simply decremented until value 0 is reached. We note that in the above described procedure, counter i is incremented from p to $f(n)$ and then decremented from $f(n)$ to $p-1$. Furthermore, both auxiliary counters $k+1$ and $k+2$ are incremented from 0 to $f(n)$ and then again decremented from $f(n)$ to 0, so they reverse only at 0 or $f(n)$. We conclude that M satisfies the required conditions.

A reversal from decrementing to incrementing can be handled in an analogous way. The only difference is that counter i keeps on decrementing until 0 is reached and is then incremented again. The two auxiliary counters $k+1$ and $k+2$ can be used in exactly the same way.

We assume that M reverses the input head m times and every counter reverses at most q times on some input w. Obviously, M' also reverses the input head m times on input w. Furthermore, for every reversal of a counter of M that is not done at either value 0 or value $f(n)$, M' reverses counters $k+1$ and $k+2$ twice. Hence, the two auxiliary counters reverse at most $2kq$ times on input w. □

We are now ready to show how $\text{CDFA}_n(k)$ can be simulated by NBMCA such that the number of input head reversals is preserved and no counter is reset.

Lemma 4.17. *For every $M \in 2\text{CDFA}_n(k)$ there exists an $M' \in \text{NBMCA}(k+2)$ with $L(M) = L(M')$. Furthermore, M' resets none of its counters and, for every $w \in \Sigma^*$, if M reverses the input head m times on input w, then M' reverses the input head $m+2$ times on input w.*

Proof. We show how to define an automaton M' that simulates M. First, we transform M into an equivalent $2\text{CDFA}_n(k+2)$ $\widehat{M}$ that reverses counters only at value 0 or value n. By Lemma 4.16 we know that such an automaton exists and, furthermore, that $\widehat{M}$ reverses the input head m times. We shall now show how this automaton $\widehat{M}$ is simulated by M'. At the beginning of a computation, M' checks whether or not *all* modulo counters are initialised with a counter bound of n, where n is the current input length. This can be done by moving the input head from the left endmarker to the right endmarker and simultaneously incrementing all $k+2$ modulo counters. After that, the input head needs to be moved back to the left end of the input, so M' makes two additional input head reversals. Next, we show how M' simulates a step of $\widehat{M}$. The input head of M' is used in exactly the same way $\widehat{M}$ uses its input head, and every counter of M' simulates one counter of $\widehat{M}$. Since the modulo counters of M' have bound n, we can simulate both, an incrementing sequence from 0 to n and a decrementing sequence from n to 0 of a counter of $\widehat{M}$ by an incrementing cycle from 0 to n of a modulo counter of M'. However, we need to keep track in the finite state control on whether the counters of M' simulate an incrementing or a decrementing cycle of a counter of $\widehat{M}$ at the moment, i. e., M' keeps track on whether reaching the counter bound with some modulo counter is interpreted as the situation that the corresponding counter of $\widehat{M}$ reaches 0 or it reaches n.

From our considerations above, we can conclude that, on any input, M' reverses the input head exactly two times more often than M. Furthermore, none of the modulo counters is reset. □

Next, we take a closer look at the model 2CDFA(1), i. e., a deterministic two-way counter automaton with only one counter that is not restricted by the input length. We can observe that in accepting computations, a 2CDFA(1) cannot reach arbitrarily large values with its counter.

Lemma 4.18. *Let M be an arbitrary* 2CDFA(1)*. During the computation of M on an arbitrary $w \in L(M)$, the counter never reaches a value $m \geq 2\,|Q|\,(|w|+2)$.*

Proof. By definition, M stops as soon as an accepting state is reached. Let Q be the set of states of M, let F be the set of accepting states and let $C_{M,w} := \{[q,h,d] \mid q \in Q, 0 \leq h \leq |w|+1, d \in \mathbb{N}\}$ be the set of possible configurations of M on input $w \in L(M)$. Furthermore, for every $[q,h,d] \in C_{M,w}$ let the mapping g be defined by

$$g([q,h,d]) := \begin{cases} [q,h,0] & \text{if } d = 0, \\ [q,h,1] & \text{else.} \end{cases}$$

In order to prove the statement of the lemma, we assume to the contrary that the counter of M reaches a value $m \geq 2|Q|(|w|+2)$ in the computation of M on w. This implies that in the computation of M on w there must be a sequence of at least $m+1$ configurations such that the first of these configurations has a counter value of 0, the last configuration has a counter value of m and all the configurations in between have a counter value strictly between 0 and m. More precisely, there exists a sequence of configurations $c_1, c_2, \ldots, c_{m'}$, $m' > m$, where $c_i := [q_i, h_i, d_i]$, $1 \leq i \leq m'$, $d_1 = 0$, $d_{m'} = m$ and $1 \leq d_i \leq m-1$, $2 \leq i \leq m'-1$. Furthermore, $q_i \notin F$, $1 \leq i < m'$, as otherwise the automaton stops in a configuration c_i, $1 \leq i < m'$. As $|\{g(c) \mid c \in C_{M,w}\}| = 2|Q|(|w|+2)$ and $m' > 2|Q|(|w|+2)$, we can conclude that there exist j, j', $1 \leq j < j' \leq m'$, with $g(c_j) = g(c_{j'})$. Since M is deterministic, this implies that in the computation for c_j and $c_{j'}$ the transition function applies the same transition; thus, the computation may enter a loop. We consider two possible cases depending on the counter values d_j and $d_{j'}$ of the configuration c_j and $c_{j'}$:

- $d_j \leq d_{j'}$: This implies that M has entered an infinite loop and all states in this loop are non-accepting. Thus, $w \notin L(M)$, which is a contradiction.
- $d_j > d_{j'}$: This implies that M decrements the counter to value 0 before reaching value m, which contradicts the fact that $1 \leq d_i \leq m-1$, $2 \leq i \leq m'-1$, and $d_{m'} = m$.

Consequently, the assumption that the counter reaches a value $m \geq 2|Q|(|w|+2)$ implies $w \notin L(M)$, which is a contradiction. □

The above given result, that shows that the counter of 2CDFA(1) is generally bounded, and Lemma 4.17 can be used in order to obtain the following result.

Lemma 4.19. $\mathcal{L}(\text{2CDFA}(1)) \subseteq \mathcal{L}((\infty, \infty, 0)\text{-REV-NBMCA}(3))$.

Proof. For every $M \in \text{2CDFA}(1)$ with a set of states Q_M, in every accepting computation of M on any w, the counter value does not reach the value $2\,|Q_M|\,(|w|+2)$. This implies that there exists a constant c_M depending on Q_M such that in every accepting computation of M on any w the counter value does not reach the value $c_M\,|w|$. This implies that we can construct an $M' \in \text{2CDFA}(1)$ with a set of states $Q_{M'} := \{q_i \mid q \in Q_M, 1 \leq i \leq c_M\}$ and $L(M) = L(M')$. More precisely, whenever the counter of M is in state q and has a counter value of $(k\,c_M)+k'$, $k, k' \in \mathbb{N}$, then M' is in state $q_{k'}$ and has a counter value of k. In other words, M' uses the subscript in the states as a counter bounded by c_M and increments the actual counter only every c_M increments. This implies that in every accepting computation of M' on some w the counter value does not reach the value $|w|$. Hence, its counter is bounded by $|w|$. Consequently, we can simulate M' by an $M'' \in \text{2CDFA}_n(1)$: On any input w, we simulate M' by M'' and abort the current computation in a non-accepting state in the case that the counter reaches a value of $|w|$. Furthermore, by Lemma 4.17, we can simulate $\text{2CDFA}_n(1)$ by $(\infty, \infty, 0)$-REV-NBMCA(3) and, thus, the statement of the lemma is implied. □

We note that both, the above sketched constructions and the construction of Lemma 4.17, increase the number of input head reversals only by 2, i. e., a 2CDFA(1) that makes at most a constant number of m input head reversals can be simulated by a $(m+2, \infty, 0)$-REV-NBMCA(3). Together with Theorem 4.15 we can conclude the following corollary.

Corollary 4.20. *The emptiness, finiteness, universe, equivalence, inclusion and disjointness problems are undecidable for* $(3, \infty, 0)$-REV-NBMCA(3).

Hence, our above question is answered in the negative: NBMCA with an unbounded number of counter reversals have undecidable problems even in the case of only three counters and a strongly bounded number of input head reversals and resets.

4.1.3 NBMCA without States

In this section we introduce *stateless* NBMCA (SL-NBMCA for short).

As already mentioned at the beginning of Section 4.1, stateless versions of automata have first been considered just a few years ago by Yang et al. [88]. Since then, various kinds of automata such as multi-head automata, counter machines and restarting automata have been investigated regarding their stateless variants (cf. [38, 24, 37, 47]).

A stateless NBMCA can be regarded as an NBMCA with only one internal state that is never changed. Hence, the component referring to the state is removed from the transition function and transitions do not depend anymore on the state. As a result, the acceptance of inputs by accepting state is not possible anymore. So for stateless NBMCA we define the input to be accepted by a special accepting transition, i. e., the transition that does not change the configuration of the automaton anymore. On the other hand, if the automaton enters a configuration for which no transition is defined, then the input is rejected and the same happens if an infinite loop is entered. For example, $(b, s_1, \ldots, s_k) \rightarrow (r, d_1, \ldots, d_k)$ is a possible transition for an SL-NBMCA(k) and $(b, s_1, \ldots, s_k) \rightarrow (0, 0, 0, \ldots, 0)$ is an accepting transition. For the sake of convenience we shall denote an accepting transition by $(b, s_1, \ldots, s_k) \rightarrow \mathbf{0}$. An SL-NBMCA($k$) can be given as a tuple (k, Σ, δ) comprising the number of counters, the input alphabet and the transition function.

As already mentioned at the beginning of Section 4.1, in Ibarra et al. [38] an example of a stateless multi-head automaton that recognises $\{a^k\}$ can be found. We shall now consider a similar example with respect to SL-NBMCA, i. e., we show how the following languages can be recognised by SL-NBMCA.

Definition 4.21. For every $k \in \mathbb{N}$, let $S_k := \{a^k, \varepsilon\}$.

We introduce an SL-NBMCA(5) that recognises S_3 and prove its correctness.

Definition 4.22. Let $M_{S_3} := (5, \{a\}, \delta) \in$ SL-NBMCA(5), where δ is defined by

1. $(\texttt{¢}, \texttt{t}_0, \texttt{t}_0, \texttt{t}_0, \texttt{t}_0, \texttt{t}_0) \rightarrow_\delta (\texttt{1}, \texttt{1}, \texttt{1}, \texttt{1}, \texttt{1}, \texttt{r})$,
2. $(\texttt{a}, \texttt{t}_1, \texttt{t}_1, \texttt{t}_1, \texttt{t}_1, \texttt{t}_0) \rightarrow_\delta (-\texttt{1}, \texttt{1}, \texttt{1}, \texttt{1}, \texttt{1}, \texttt{1})$,
3. $(\texttt{¢}, \texttt{t}_0, \texttt{t}_0, \texttt{t}_0, \texttt{t}_0, \texttt{t}_1) \rightarrow_\delta (\texttt{1}, \texttt{1}, \texttt{0}, \texttt{0}, \texttt{0}, \texttt{0})$,
4. $(\texttt{a}, \texttt{t}_1, \texttt{t}_0, \texttt{t}_0, \texttt{t}_0, \texttt{t}_1) \rightarrow_\delta (\texttt{1}, \texttt{0}, \texttt{1}, \texttt{0}, \texttt{0}, \texttt{0})$,
5. $(\texttt{a}, \texttt{t}_1, \texttt{t}_1, \texttt{t}_0, \texttt{t}_0, \texttt{t}_1) \rightarrow_\delta (\texttt{1}, \texttt{0}, \texttt{0}, \texttt{1}, \texttt{0}, \texttt{0})$,
6. $(\texttt{a}, \texttt{t}_1, \texttt{t}_1, \texttt{t}_1, \texttt{t}_0, \texttt{t}_1) \rightarrow_\delta (\texttt{1}, \texttt{0}, \texttt{0}, \texttt{0}, \texttt{1}, \texttt{1})$,
7. $(\$, \texttt{t}_1, \texttt{t}_1, \texttt{t}_1, \texttt{t}_1, \texttt{t}_0) \rightarrow_\delta \mathbf{0}$.

Proposition 4.23. $L(M_{S_3}) = S_3$.

Proof. Before we prove in a formal manner that $L(M_{S_3}) = S_3$, we give an intuitive understanding of how M_{S_3} recognises S_3. The first 4 counters are used to count the 4 steps that are necessary to move the input head from the left to the right endmarker in case that aaa is the input. However, this is only possible if all counter bounds of these counters are 1. So initially M_{S_3} checks whether or not all the first 4 counters are initialised with counter bounds of 1. To this end, the input head is moved one step to the right while the first 4 counters are incremented. After that it is checked whether all these counters have reached their bounds after this increment and then the input head is moved back to the left endmarker. Then, M_{S_3} uses the counters in order to count the occurrences of symbols a on the input tape. Hence, the computations of M_{S_3} comprise two parts: a first part of checking whether or not the right counter bounds are guessed and a second part where the symbols on the tape are counted. However, since there are no states, the automaton is not able to distinguish between these two parts. This is mainly due to the fact that in the first part we move the input head and, thus, necessarily scan an input symbol. So it is possible that the counter bounds are initialised in a way such that in the first part of the computation M_{S_3} accidentally enters a configuration that is also reached in the second part. In order to separate these two parts of the computation, we need an additional counter. In the following we formally prove the correctness of M_{S_3}.

We first show that both inputs ε and aaa are accepted by M_{S_3} if all the counters $1, 2, 3$ and 4 are initialised with 1, counter 5 is initialised with some bound $C \geq 1$ and guesses 1 as counter bound when reset by transition 1. More formally, if ε is the input and the counters are initialised as described above, M_{S_3} first applies transition 1 and reaches a configuration $[1, (1, 1), (1, 1), (1, 1), (1, 1), (0, 1)]$. As the input is empty, at position 1 the right endmarker \$ occurs; hence, M_{S_3} applies transition 7 and accepts. If, on the other hand, aaa is the input, then M_{S_3} again first applies transition 1 and reaches the configuration $[1, (1, 1), (1, 1), (1, 1), (1, 1), (0, 1)]$. Now it is easy to verify that transitions 2, 3, 4, 5, 6 and 7 apply in exactly this order, and this implies that M_{S_3} accepts aaa.

Next, we have to show that no $a^{k'}$, $k' \neq 3$, $k' \neq 0$, can be accepted by M_{S_3}. To this end, we first take a closer look at the counter bounds and observe that in every possible accepting computation, each of the counters $1, 2, 3$ and 4 must be initialised with counter bound 1 and for counter 5, which can be reset several times, the *last* guessed bound must be 1 as well. This can be concluded from the following considerations.

For any input, M_{S_3} can start off with either transition 1 or 3. If transition 3 is first applicable, then the counter message of counter 5 is initially $\mathtt{t_1}$, which implies that it is initialised with bound 0. Furthermore, no matter how often counter 5 is incremented, its message cannot change unless it is reset. Since there does not exist any transition that resets counter 5 when its message is $\mathtt{t}_1$ and since transition 7 requires the counter message of counter 5 to be $\mathtt{t}_0$, M_{S_3} cannot accept the input in this case. In a similar way we can also conclude that in an accepting computation it is not possible that 0 is guessed as counter bound for counter 5 by a reset. This implies that every accepting computation starts with transition 1, which in turn means that no counter $1, 2, 3$ or 4 can be initialised with 0. After transition 1, the input head scans a symbol $b \in \{a, \$\}$, the value of the first 4 counters is 1 and counter 5 has message $\mathtt{t}_0$. For such a configuration only transitions 2 and 7 are possible, which both require the counter messages of the first 4 counters to be $\mathtt{t}_1$, thus, the first 4 counters must have a counter bound of 1. We recall that we assume that the input is not empty, thus, $b = a$, and the only next possible transition is transition 2, which moves the input head back to the left endmarker and, by incrementing, sets counters $1, 2, 3$ and 4 back to value 0. Furthermore, counter 5 is incremented, which has been reset in the previous transition. We observe two possible cases. If for counter 5 a bound greater than 1 is guessed, then M_{S_3} reaches configuration $[0, (0, 1), (0, 1), (0, 1), (0, 1), (1, C_5)]$, $1 < C_5$, which implies that M_{S_3} is in its initial configuration again and transitions 1 and 2 apply in the same way as before. So the only possible case in an accepting computation is that eventually 1 is guessed as counter bound for counter 5 by transition 1 and, thus, a configuration $[0, (0, 1), (0, 1), (0, 1), (0, 1), (1, 1)]$ is reached.

Now, if $a^{k'}$ with $k' > 3$ is the input, then, by applying transitions $3, 4, 5$ and 6 in this order, M_{S_3} reaches the configuration $[4, (1, 1), \dots, (1, 1), (0, 1)]$ and, since at position 4 the symbol a occurs, M_{S_3} rejects. If $k' < 3$, then the input head reaches the right endmarker in either configuration $[k' + 1, (1, 1), (1, 1), (0, 1), (0, 1), (1, 1)]$ or $[k' + 1, (1, 1), (1, 1), (1, 1), (0, 1), (1, 1)]$ for which both no transition is defined. Consequently, M_{S_3} accepts exactly S_k.

We conclude this proof by noting that the reset of counter 5 in transition 1 is not necessary, but convenient for the proof of the correctness. □

By generalising Definition 4.22 and Proposition 4.23 it can be shown that for every $k \in \mathbb{N}$, there exists an $M_{S_k} \in \text{SL-NBMCA}(k + 2)$ with $L(M_{S_k}) = S_k$:

Proposition 4.24. *For every $k \in \mathbb{N}$, $S_k \in \mathcal{L}(\text{SL-NBMCA}(k + 2))$.*

Next, we answer the question of whether or not SL-NBMCA are as powerful as their counterparts with states. It has been shown in [38] that two-way stateless multi-head automata can easily simulate a finite state control by using additional input heads to encode a state. More precisely, a two-way multi-head automaton with a number of n states can be simulated by a stateless two-way multi-head automaton with $\log(n)$ auxiliary input heads. Each of these additional input heads is interpreted as representing 0 if it scans the left endmarker and 1 if it scans the first symbol of the input (or the right endmarker in case that the input is empty). In this way these $\log(n)$ auxiliary input heads can be used in order to encode a state as a binary number.

Regarding SL-NBMCA it is not completely obvious how this idea of simulating states can be applied. This is mainly due to the fact that the counters of an SL-NBMCA can have any counter bound, and it is not possible to control these bounds. So it seems more difficult to use a counter as some sort of a bit that can be flipped between 0 and 1. However, as we shall see, it is possible to define an SL-NBMCA such that in an accepting computation certain counters must be initialised with a counter bound of 1. Informally speaking, this is done by simply using the input head to check whether or not certain counters have counter bounds of 1. In order to do this, the input head has to be moved in the input, hence, as we cannot use any states, the problem is how to separate this initial phase from the main part of the computation.

Theorem 4.25. *For every $M \in \text{NBMCA}(k)$, $k \in \mathbb{N}$, with a set of states Q, there exists an $M' \in \text{SL-NBMCA}(k + \lceil \log(|Q| + 1) \rceil + 2)$ with $L(M) = L(M')$.*

Proof. Let $M := (k, Q, \Sigma, \delta, q_0, F)$. We show how an $M' := (k + \lceil \log(|Q| + 1) \rceil + 2, \Sigma, \delta') \in \text{SL-NBMCA}(k + \lceil \log(|Q| + 1) \rceil + 2)$ can be constructed from M with $L(M) = L(M')$. Without loss of generality we assume that, for any input $w \in \Sigma^*$, M cannot reach a configuration where the state is an accepting state and the input head scans the left endmarker. Moreover, we assume that $q_0 \notin F$ and that in every computation at least 2 steps are performed. Let the mapping $f_1 : \{\mathtt{t_0}, \mathtt{t_1}\} \to \{0, 1\}$ be defined by $f_1(\mathtt{t_0}) := 0$ and $f_1(\mathtt{t_1}) := 1$. Furthermore, let the mapping $f_2 : \mathbb{N} \times \mathbb{N} \to \{\mathtt{t_0}, \mathtt{t_1}\}$ be defined by $f_2(n_1, n_2) := \mathtt{t_0}$ if $n_1 \neq n_2$ and $f_2(n_1, n_2) := \mathtt{t_1}$ if $n_1 = n_2$. Hence, f_2 translates a counter configuration into the resulting counter message.

First, we define M', and we shall prove its correctness afterwards. For the construction of M' we use an encoding of states $g : Q \to \{\mathtt{t_0}, \mathtt{t_1}\}^m$, where $m := \lceil \log(|Q| + 1) \rceil$, that satisfies $g(q_0) = (\mathtt{t_1}, \mathtt{t_1}, \ldots, \mathtt{t_1})$ and, for every $q \in Q$, $g(q) \neq (\mathtt{t_0}, \mathtt{t_0}, \ldots, \mathtt{t_0})$. Obviously, such an encoding exists. We are now ready to define the transition function δ' of M'.

First, for every transition of M, we define a transition for M' that performs the same step, but instead of changing from one state into the other, it changes the first m counter messages from the encoding of one state into the encoding of another state. So for every transition $(p, c, s_1, \ldots, s_k) \to_\delta (q, r, d_1, \ldots, d_k)$, with

$q \notin F$, we define the transition

$$(c, \widehat{s}_1, \ldots, \widehat{s}_m, \mathtt{t}_1, \mathtt{t}_0, s_1, \ldots, s_k) \rightarrow_{\delta'} (r, \widehat{d}_1, \ldots, \widehat{d}_m, \mathtt{0}, \mathtt{0}, d_1, \ldots, d_k)\,, \tag{4.1}$$

where $g(q) = (f_1^{-1}((f_1(\widehat{s}_1) + \widehat{d}_1) \mod 2), \ldots, f_1^{-1}((f_1(\widehat{s}_m) + \widehat{d}_m) \mod 2))$ and $g(p) = (\widehat{s}_1, \ldots, \widehat{s}_m)$. Regarding the transitions of M that change the state into an accepting state, we use slightly different transitions for M'. For every transition $(p, c, s_1, \ldots, s_k) \rightarrow_\delta (q, r, d_1, \ldots, d_k)$, with $q \in F$, we define the transition

$$(c, \widehat{s}_1, \ldots, \widehat{s}_m, \mathtt{t}_1, \mathtt{t}_0, s_1, \ldots, s_k) \rightarrow_{\delta'} (r, \widehat{d}_1, \ldots, \widehat{d}_m, \mathtt{1}, \mathtt{1}, d_1, \ldots, d_k)\,, \tag{4.2}$$

where, again, $g(q) = (f_1^{-1}((f_1(\widehat{s}_1) + \widehat{d}_1) \mod 2), \ldots, f_1^{-1}((f_1(\widehat{s}_m) + \widehat{d}_m) \mod 2))$ and $g(p) = (\widehat{s}_1, \ldots, \widehat{s}_m)$. At this point, we observe that the above defined transitions of type (1) and (2) only work correctly, i. e., they change the encodings of states according to the transitions of M, if the counter bound of the first m counters is 1. This shall be a crucial point for the correctness of our approach.

By definition, if M enters an accepting state, then the input is accepted. Hence, if the first m counter messages of M' encode an accepting state of M, then M' can apply an accepting transition. So for every $q \in F$, $b \in \Sigma \cup \{\$\}$ and for every $s_i \in \{\mathtt{t}_0, \mathtt{t}_1\}$, $1 \leq i \leq k$, we define

$$(b, \widehat{s}_1, \ldots, \widehat{s}_m, \mathtt{t}_0, \mathtt{t}_1, s_1, \ldots, s_k) \rightarrow_{\delta'} \mathbf{0}\,, \tag{4.3}$$

where $g(q) = (\widehat{s}_1, \ldots, \widehat{s}_m)$. The next two transitions are used by M' to start the computation and to check whether the first $m + 2$ counters are initialised with a bound of 1. For every $s_i \in \{\mathtt{t}_0, \mathtt{t}_1\}$, $1 \leq i \leq k$, and for every $b \in \Sigma \cup \{\$\}$ we define

$$(¢, \mathtt{t}_0, \ldots, \mathtt{t}_0, \mathtt{t}_0, \mathtt{t}_0, s_1, \ldots, s_k) \rightarrow_{\delta'} (\mathtt{1}, \mathtt{1}, \ldots, \mathtt{1}, \mathtt{0}, \mathtt{0}, d_1 \ldots, d_k) \text{ and} \tag{4.4}$$

$$(b, \mathtt{t}_1, \ldots, \mathtt{t}_1, \mathtt{t}_0, \mathtt{t}_0, s_1, \ldots, s_k) \rightarrow_{\delta'} (-\mathtt{1}, \mathtt{0}, \ldots, \mathtt{0}, \mathtt{1}, \mathtt{0}, d_1, \ldots, d_k)\,, \tag{4.5}$$

where $d_i = \mathtt{0}$, $1 \leq i \leq k$. This concludes the definition of M'.

As an immediate consequence of the definition of δ', we observe the following:

Claim (1). For every $w \in \Sigma^*$, for all $p \in Q/F, q \in F$, for all $h, h' \in \{0, 1, \ldots, |w| + 1\}$, for all $C_i, C'_i \in \{0, 1, \ldots, |w|\}$, $1 \leq i \leq k$, for every $c_i \in \{0, 1, \ldots, C_i\}$, $1 \leq i \leq k$, and for every $c'_i \in \{0, 1, \ldots, C'_i\}$, $1 \leq i \leq k$,

$$[p, h, (c_1, C_1), \ldots, (c_k, C_k)] \vdash^*_{M,w} [q, h', (c'_1, C'_1), \ldots, (c'_k, C'_k)]$$

if and only if

$$\begin{aligned}&[h, (\widehat{c}_1, 1), \ldots, (\widehat{c}_m, 1), (1, 1), (0, 1), (c_1, C_1), \ldots, (c_k, C_k)] \vdash^*_{M',w}\\ &[h', (\widetilde{c}_1, 1), \ldots, (\widetilde{c}_m, 1), (0, 1), (1, 1), (c'_1, C'_1), \ldots, (c'_k, C'_k)]\,,\end{aligned}$$

where $g(p) = (f_2(\widehat{c}_1, 1), \ldots, f_2(\widehat{c}_m, 1))$ and $g(q) = (f_2(\widetilde{c}_1, 1), \ldots, f_2(\widetilde{c}_m, 1))$.

Proof. (*Claim* (1)) Before we are able to prove the statement of the claim, we have to prove the following analogous statement. For every $w \in \Sigma^*$, for all $p, q \in Q/F$, for all $h, h' \in \{0, 1, \ldots, |w| + 1\}$, for all $C_i, C'_i \in \{0, 1, \ldots, |w|\}$, $1 \leq i \leq k$, for every $c_i \in \{0, 1, \ldots, C_i\}$, $1 \leq i \leq k$, and for every $c'_i \in \{0, 1, \ldots, C'_i\}$, $1 \leq i \leq k$,

$$[p, h, (c_1, C_1), \ldots, (c_k, C_k)] \vdash^*_{M,w} [q, h', (c'_1, C'_1), \ldots, (c'_k, C'_k)]$$

if and only if

$$\begin{aligned}&[h, (\widehat{c}_1, 1), \ldots, (\widehat{c}_m, 1), (1, 1), (0, 1), (c_1, C_1), \ldots, (c_k, C_k)] \vdash^*_{M',w}\\ &[h', (\overline{c}_1, 1), \ldots, (\overline{c}_m, 1), (1, 1), (0, 1), (c'_1, C'_1), \ldots, (c'_k, C'_k)]\,,\end{aligned}$$

where $g(p) = (f_2(\widehat{c}_1, 1), \ldots, f_2(\widehat{c}_m, 1))$ and $g(q) = (f_2(\overline{c}_1, 1), \ldots, f_2(\overline{c}_m, 1))$. This can be concluded by observing that, by definition of the transitions of type (1), if the first $m+2$ counters are initialised with a counter bound of 1, then the counter instructions for the first m counters are chosen such that the encoding of states are changed exactly according to how M changes its states. Furthermore, the input head and the remaining counters are used in exactly the same way as it is done by M. Thus, the above statement is correct.

Next, we consider the statement of the claim and observe that since we assume that in every computation of M at least 2 steps are performed,

$$[p, h, (c_1, C_1), \ldots, (c_k, C_k)] \vdash^*_{M,w} [q, h', (c'_1, C'_1), \ldots, (c'_k, C'_k)]$$

if and only if

$$\begin{aligned}&[p, h, (c_1, C_1), \ldots, (c_k, C_k)] \vdash^*_{M,w}\\ &[p', h'', (c''_1, C''_1), \ldots, (c''_k, C''_k)] \vdash_{M,w}\\ &[q, h', (c'_1, C'_1), \ldots, (c'_k, C'_k)]\,,\end{aligned}$$

and there exists $(p', w[h''], f_2(c''_1, C''_1), \ldots, f_2(c''_k, C''_k)) \rightarrow_\delta (q, r, d_1, \ldots, d_k)$, with $p' \notin F$, $h'' + r = h'$ and, for every i, $1 \leq i \leq k$, $c'_i = c''_i + d_i \mod (C''_i + 1)$ if $d_i \neq r$ and $c'_i = 0$ otherwise. By the statement from above and the definition of the transitions of type (2), we conclude that this holds if and only if

$$\begin{aligned}&[h,(\widehat{c}_1,1),\ldots,(\widehat{c}_m,1),(1,1),(0,1),(c_1,C_1),\ldots,(c_k,C_k)] \vdash^*_{M',w}\\ &[h'',(\overline{c}_1,1),\ldots,(\overline{c}_m,1),(1,1),(0,1),(c''_1,C''_1),\ldots,(c''_k,C''_k)] \vdash_{M',w}\\ &[h',(\widetilde{c}_1,1),\ldots,(\widetilde{c}_m,1),(0,1),(1,1),(c'_1,C'_1),\ldots,(c'_k,C'_k)]\,.\end{aligned}$$

where $g(p)=\widehat{c}_1,\ldots,\widehat{c}_m$, $g(p')=\overline{c}_1,\ldots,\overline{c}_m$ and $g(q)=\widetilde{c}_1,\ldots,\widetilde{c}_m$. This concludes the proof of the claim. □ (*Claim* (1))

In order to prove the correctness of M' we claim the following:

Claim (2). Every *accepting* computation of M' on some $w \in L(M')$ starts with

$$\begin{aligned}&[0,(0,1),\ldots,(0,1),(0,1),(0,1),(0,C_1),\ldots,(0,C_k)] \vdash_{M',w}\\ &[1,(1,1),\ldots,(1,1),(0,1),(0,1),(0,C_1),\ldots,(0,C_k)] \vdash_{M',w}\\ &[0,(1,1),\ldots,(1,1),(1,1),(0,1),(0,C_1),\ldots,(0,C_k)]\,,\end{aligned}$$

for some $C_i \in \{0,1,\ldots,|w|\}$, $1 \le i \le k$.

Proof. (*Claim* (2)) If the first $m+2$ counters are initialised with counter bounds of 1, then at the beginning of the computation transitions (4) and (5) apply first, which implies that exactly the above mentioned configurations are reached. So it remains to show that for every accepting computation, the first $m+2$ counters must be initialised with 1.

First, we observe that an accepting computation cannot start with an transition of either type (1) or type (2), as this implies that counter $m+1$ is initialised with 0, which means that the counter message of this counter stays $\mathtt{t_1}$ for the entire computation and there is no accepting transition defined for the case that counter $m+1$ has a message of $\mathtt{t_1}$. Furthermore, it is not possible that a transition of type (3) or transition (5) is the first transition of an accepting computation as these transitions are not defined for the symbol ¢. Here, we use the fact that the transitions of type (3) are only defined for input symbols different from ¢, which is only possible because we assume that M cannot reach a configuration where the state is an accepting state and the input head scans the left endmarker. We conclude that an accepting computation must start with transition (4).

This implies that none of the first $m+2$ counters are initialised with 0. Transition 4 increments the first m counters, so in case that at least one of them is not initialised with a counter bound of 1, a configuration is reached where at least one of the first m counters has counter message $\mathtt{t_0}$ and counter $m+1$ and $m+2$ also have counter message $\mathtt{t_0}$ while the input head scans some symbol $b \in \Sigma \cup \{\$\}$. For such a configuration no transition is defined; thus, we can assume that the first

m counters are initialised with a counter bound of 1. For the configuration that is reached by applying transition (4), transition (5) is the only next applicable transition. By applying transition (5), the input head is moved back to the left endmarker and counter $m+1$ is incremented. If counter $m+1$ has a counter bound strictly greater than 1, its counter bound stays $\mathtt{t_0}$ and a configuration is reached where the first m counters have message $\mathtt{t_1}$, counters $m+1$ and $m+2$ have messages $\mathtt{t_0}$ and the input head scans ¢. Again, there is no transition defined for such a configuration. Hence, we can conclude that the counter bound of counter $m+1$ is 1 as well. So far, we have shown that an accepting computation of M' on some input w starts with

$$\begin{aligned}&[0, (0, 1), \ldots, (0, 1), (0, 1), (0, C'), (0, C_1), \ldots, (0, C_k)] \vdash_{M',w}\\ &[1, (1, 1), \ldots, (1, 1), (0, 1), (0, C'), (0, C_1), \ldots, (0, C_k)] \vdash_{M',w}\\ &[0, (1, 1), \ldots, (1, 1), (1, 1), (0, C'), (0, C_1), \ldots, (0, C_k)]\,,\end{aligned}$$

for some $C' \in \{1, 2, \ldots, |w|\}$. Hence, it remains to show that $C' = 1$.

Obviously, for an accepting computation, at some point there must a transition of type (3) be applied. Since all transitions of type (3) require the counter message of counter $m+1$ to be $\mathtt{t_0}$, it is necessary that a transition of type (2) is applied. We assume now that the counter bound of counter $m+2$ is strictly greater than 1. So by applying a transition of type (2) a configuration is reached where counters $m+1$ and $m+2$ have message $\mathtt{t_0}$ and the first m counters have messages $\widehat{s}_1, \ldots, \widehat{s}_m$ with $g(q) = (\widehat{s}_1, \ldots, \widehat{s}_m)$ for some $q \in F$. There are only two possible sequences of messages $\widehat{s}_1, \ldots, \widehat{s}_m$ such that a transition is defined. If $\widehat{s}_i = \mathtt{t_1}$, $1 \leq i \leq m$, then transition (5) might be applicable next. However, since $g(q_0) = (\mathtt{t_1}, \ldots, \mathtt{t_1})$, this implies $q_0 \in F$, which contradicts our assumption that q_0 is not an accepting state. If, on the other hand, $\widehat{s}_i = \mathtt{t_0}$, $1 \leq i \leq m$, then transition (4) might be applicable but this implies that there exists a state q with $g(q) = (\mathtt{t_0}, \ldots, \mathtt{t_0})$, which, by definition of g, is not possible. Consequently, we can conclude that in an accepting computation the counter bound of counter $m+2$ is 1, which concludes the proof. □ (*Claim* (2))

From Claims (1) and (2) we can directly conclude that $w \in L(M)$ if and only if $w \in L(M')$. Thus, $L(M) = L(M')$. □

As demonstrated by the above results SL-NBMCA can simulate NBMCA by using additional counters. For this simulation as well as for the automaton M_{S_3} (see Definition 4.22) recognising S_3, it is a vital point that certain counters have a counter bound of 1. The automaton M_{S_3} uses these counters in order to count input symbols and in the simulation of NBMCA by SL-NBMCA we interpret them as a binary number encoding a state. Due to the lack of states, this need for counters to be initialised with a counter bound of 1 involves considerable technical challenges.

4.1.4 Stateless NBMCA with Bounded Resets

As described at the beginning of Section 4.1, it is our goal to investigate the role of nondeterminism in the case that there does not exist a finite state control that can be used to control the nondeterminism. Since, as demonstrated by the previous section, SL-NBMCA can simulate states using their counters, we now consider SL-NBMCA(1). However, since it is not possible to simulate arbitrarily large finite state controls with a fixed number of counters, we anticipate that for any class SL-NBMCA(k), $k \in \mathbb{N}$, similar results to the following ones exist.

We take a closer look at SL-NBMCA(1) the input heads of which operate in a one-way manner, i. e., for every transition $(b, x) \rightarrow_\delta (y, z)$, we have $y \in \{\mathtt{0}, \mathtt{1}\}$. Furthermore, the number of counter resets is bounded. These classes of SL-NBMCA shall be denoted by 1SL-NBMCA$_k$(1), where k is the maximum number of resets allowed. The number of resets of an 1SL-NBMCA$_k$(1) is bounded in the following way. In any computation of a 1SL-NBMCA$_k$(1), the first k applications of a transition of form $(b, x) \rightarrow (y, \mathtt{r})$, $b \in \Sigma$, $x \in \{\mathtt{t_0}, \mathtt{t_1}\}$, $y \in \{\mathtt{0}, \mathtt{1}\}$, reset the counter in accordance with the definition of NBMCA. Every further application of a transition of that form simply ignores the counter, i. e., the counter value and counter bound remain unchanged. More precisely, if in a computation a transition $(a, x) \rightarrow (y, \mathtt{r})$ is applied after the counter has already been reset for at least k times, then this transition is interpreted as $(a, x) \rightarrow (y, \mathtt{0})$.

This way of restricting automata is unusual compared to the common restrictions that are found in the literature. We shall explain this in a bit more detail and consider input head reversal bounded automata as an example (see, e. g., Ibarra [36]). An input head reversal bounded automaton is an automaton that can recognise each word of a language in such a way that the number of input head reversals is bounded. There is no need to require the input head reversals to be bounded in the non-accepting computations as well, as this does not constitute a further restriction. This is due to the fact that we can always use the finite state control to count the number of input head reversals in order to interrupt a computation in a non-accepting state as soon as the bound of input head reversals is exceeded. However, regarding stateless automata this is not necessarily possible anymore and it seems that it is a difference whether a restriction is defined for all possible computations or only for the accepting ones. Our definition of bounded resets from above avoids these problems by slightly changing the model itself, i. e., in every computation it loses the ability to reset the counter after a number of resets.

We recall that in a computation of an NBMCA, the counters are already nondeterministically initialised. Hence, in a computation of a 1SL-NBMCA$_k(1)$ the counter can have $k+1$ distinct counter bounds. Since the input head of 1SL-NBMCA$_k(1)$ is a one-way input head, we require all accepting transitions to be of form $(\$, x) \to \mathbf{0}$, $x \in \{\mathtt{t}_0, \mathtt{t}_1\}$.

The main question is whether or not the classes $\mathcal{L}(\text{1SL-NBMCA}_k(1))$, $k \in \mathbb{N}$, describe a hierarchy with respect to k. First, for every $k \in \mathbb{N}$, we separate the classes $\mathcal{L}(\text{1SL-NBMCA}_k(1))$ and $\mathcal{L}(\text{1SL-NBMCA}_{k+1}(1))$ by identifying a language L that can be recognised by an 1SL-NBMCA$_{k+1}(1)$, but there exists no $M \in$ 1SL-NBMCA$_k(1)$ with $L(M) = L$. The words of the separating language for 1SL-NBMCA$_k(1)$ and 1SL-NBMCA$_{k+1}(1)$ are basically concatenations of $k+2$ words u_i, $1 \leq i \leq k+2$, where each u_i comprises unary factors of the same length n_i, $1 \leq i \leq k+2$. A possible 1SL-NBMCA$_{k+1}(1)$ for this language can be initialised with a counter bound of n_1, it can guess n_i, $2 \leq i \leq k+2$, as counter bounds in the computation and it can use the counter to check the unary factors for equality. Since it is possible that $n_i \neq n_{i+1}$, the automaton needs $k+2$ distinct counter bounds. Hence, a 1SL-NBMCA$_k(1)$, which can only use at most $k+1$ different counter bounds in any computation, is not able to recognise this language. Next, we shall formally define these languages and use them in the above illustrated way in order to separate the classes 1SL-NBMCA$_k(1)$, $k \in \mathbb{N}$.

In the remainder of this section, we exclusively consider languages and automata defined over the alphabet $\Sigma := \{a, \#_1, \#_2\}$. Next, for every $k \in \mathbb{N}$, we define a language over Σ that shall then be shown to be in $\mathcal{L}(\text{1SL-NBMCA}_k(1))$ but not in $\mathcal{L}(\text{1SL-NBMCA}_{k-1}(1))$.

Definition 4.26. For every $n \in \mathbb{N}_0$ let $\widetilde{\mathbf{L}}_n := \{a^n\} \cdot \{\#_1 \cdot a^n\}^*$, and let $\widetilde{\mathbf{L}} := \bigcup_{n \in \mathbb{N}_0} \widetilde{\mathbf{L}}_n$. Furthermore, for every $k \in \mathbb{N}$, let

$$\begin{aligned}
\mathbf{L}_{k,1} &:= \{u_1 \cdot \#_2 \cdot u_2 \cdot \#_2 \cdot \cdots \cdot \#_2 \cdot u_{k'} \mid u_i \in \widetilde{\mathbf{L}}, 1 \leq i \leq k' \leq k\}\,,\\
\mathbf{L}_{k,2} &:= \{u_1 \cdot \#_2 \cdot \cdots \cdot \#_2 \cdot u_k \cdot \#_2 \cdot \cdots \cdot \#_2 \cdot u_{k'} \mid u_i \in \widetilde{\mathbf{L}}, 1 \leq i \leq k \leq k',\\
&\qquad \text{if } u_k \in \widetilde{\mathbf{L}}_0, \text{ then } u_{i'} \in \widetilde{\mathbf{L}}_0, k+1 \leq i' \leq k',\\
&\qquad \text{if } u_k \in \widetilde{\mathbf{L}}_n, n \in \mathbb{N}, \text{ then } u_{i'} \in (\{\#_1\} \cdot \widetilde{\mathbf{L}}_n), k+1 \leq i' \leq k'\}\,,
\end{aligned}$$

and let $\mathbf{L}_k := \mathbf{L}_{k,1} \cup \mathbf{L}_{k,2}$.

Thus, the words of language $\widetilde{\mathbf{L}}$ consist of concatenations of factors over $\{a\}$ of the same length that are separated by occurrences of $\#_1$. The words of the language $\mathbf{L}_k$ are basically concatenations of words in $\widetilde{\mathbf{L}}$ separated by occurrences of $\#_2$. However, in a word from $\mathbf{L}_k$, only the first k of these elements from $\widetilde{\mathbf{L}}$ can be arbitrarily chosen, for all the others the length of the factors over $\{a\}$ must be the same as for the k^{th} word, with the only difference that they start with an additional occurrence of $\#_1$. For example,

$$aa \cdot \#_1 \cdot aa \cdot \#_2 \cdot \#_2 \cdot aaa \cdot \#_1 \cdot aaa \cdot \#_1 \cdot aaa \cdot \#_2 \cdot \#_1 \cdot aaa \cdot \#_1 \cdot aaa \in \mathbf{L}_3\,,$$
$$aaaaaa \cdot \#_2 \cdot a \cdot \#_1 \cdot a \cdot \#_1 \cdot a \cdot \#_2 \cdot aaa \cdot \#_1 \cdot aaa \cdot \#_2 \cdot \#_2 \cdot \#_1 \cdot \#_1 \cdot \#_1 \in \mathbf{L}_4\,,$$
$$aaaaaaaa \cdot \#_1 \cdot aaaaaaaa \cdot \#_1 \cdot aaaaaaaa \in \mathbf{L}_6\ .$$

For every $k \in \mathbb{N}$, we now define a 1SL-NBMCA$_{k-1}(1)$ that recognises exactly the language $\mathbf{L}_k$.

Definition 4.27. Let $M_{\mathbf{L}} := (1, \{a, \#_1, \#_2\}, \delta) \in$ SL-NBMCA(1), where δ is defined by

1. $(¢, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$,
2. $(¢, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$,
3. $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$,
4. $(\#_1, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$,
5. $(\#_2, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{r})$,
6. $(\$, \mathtt{t_1}) \rightarrow_\delta \mathbf{0}$.

For every $k \in \mathbb{N}$, let $M_{\mathbf{L}_k}$ be the above defined automaton $M_{\mathbf{L}}$ interpreted as an 1SL-NBMCA$_{k-1}(1)$.

We now explain how $M_{\mathbf{L}_k}$ recognises $\mathbf{L}_k$ in an informal way. $M_{\mathbf{L}_k}$ uses its counter to count the occurrences of a on the input tape. Whenever an occurrence of $\#_1$ is scanned, the counter must have reached its counter bound, which then implies that the length of the factor over $\{a\}$ correspond to the counter bound. When an occurrence of $\#_2$ is scanned, the counter message must be $\mathtt{t_1}$ as well, and, furthermore, in case that the input is a word from $\mathbf{L}_k$, a new sequence of possibly different factors over $\{a\}$ follows and, thus, the counter is reset in order to guess a new counter bound. As soon as all $k-1$ resets are used, the counter bound does not change anymore; hence, the remaining factors over $\{a\}$ must all have the same length. We note that $k-1$ resets are sufficient as the counter is nondeterministically initialised with a counter bound that can be used for the first factors over $\{a\}$.

Theorem 4.28. *For every $k \in \mathbb{N}$, $\mathbf{L}_k \in \mathcal{L}($1SL-NBMCA$_{k-1}(1))$.*

Proof. We note that it is sufficient to show that $L(M_{\mathbf{L}_k}) = \mathbf{L}_k$. First we prove two claims.

Claim (1). Let ¢w\$, $w \in \{\#_1, \#_2, a\}^*$, be an arbitrary input for $M_{\mathbf{L}_k}$ and let j, j' be arbitrarily chosen with $0 \leq j < j' \leq |w| + 1$, $w[j], w[j'] \in \{\#_2, ¢, \$\}$ and $w[j''] \neq \#_2$, $j < j'' < j'$. Furthermore, let $c_1, c_2, \ldots, c_m$ be the initial part of a computation of $M_{\mathbf{L}_k}$ on w and the counter has been reset at most $k - 2$ times in this initial part of the computation. Then, for all n, n' with $0 \leq n, n' \leq |w|$, the following holds. If $j \neq 0$, then

$$c_m = [j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n', n')] \text{ if and only if } w[j + 1, j' - 1] \in \widetilde{\mathbf{L}}_{n'} \,,$$

and if $j = 0$, then

$$[j, (0, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)] \text{ if and only if } w[j + 1, j' - 1] \in \widetilde{\mathbf{L}}_{n} \,.$$

Proof. (*Claim* (1)) We first show that if $j \neq 0$, then $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n', n')]$ implies $w[j + 1, j' - 1] \in \widetilde{\mathbf{L}}_{n'}$ and if $j = 0$, then $[j, (0, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)]$ implies $w[j + 1, j' - 1] \in \widetilde{\mathbf{L}}_{n}$. To this end, we first assume that $j \neq 0$ and $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n', n')]$ and take a closer look at this computation. Let $u := w[j + 1, j' - 1]$. Initially, the input head scans $w[j] = \#_2$ and the counter configuration is (n, n), thus, the counter message is $\mathtt{t}_1$. This implies that transition 5 applies, i. e., the input head is moved to position $j + 1$ and the new counter configuration is $(0, n')$. If $n' = 0$, the counter message is $\mathtt{t}_1$ and, since $|u|_{\#_2} = 0$, the counter is not reset in the whole process of scanning u and therefore cannot change from $\mathtt{t}_1$ to $\mathtt{t}_0$. Furthermore, since the input head cannot be moved over an occurrence of a with counter message $\mathtt{t}_1$, we can conclude that $|u|_a = 0$, which implies $u \in \{\#_1\}^*$ and, hence, $u \in \widetilde{\mathbf{L}}_{n'} = \widetilde{\mathbf{L}}_0$. If $n' \geq 1$, the first symbol of u must be an a, as the counter message is $\mathtt{t}_0$ when that symbol is scanned. Now, the only applicable transition is transition 3. This transition is successively applied as long as a's are scanned and the counter message is $\mathtt{t}_0$. Furthermore, in each step the counter is incremented. This implies that if the symbol $\#_2$ on position j is followed by less than n' a's the input head reaches $\#_1$ or $\#_2$ with counter message $\mathtt{t}_0$ and if it is followed by more than n' a's, the input head reaches an a with counter message $\mathtt{t}_1$. In both cases no transition is defined, so we conclude that u starts with $a^{n'}$ and the symbol to the right of the n'^{th} symbol of u must be $\#_1$ or $\#_2$. If this symbol is $\#_2$, we conclude $u = a^{n'} \in \widetilde{\mathbf{L}}_{n'}$. If, on the other hand, this symbol is $\#_1$, then transition 4 applies which sets the counter value back to 0 and moves the input head to the right and then, for the same reasons as before, n' a's occur followed by either $\#_1$ or $\#_2$. We conclude that $u = a^{n'} \cdot (\#_1 \cdot a^{n'})^m$ for some $m \geq 1$; thus, $u \in \widetilde{\mathbf{L}}_{n'}$.

Next, we assume that $j = 0$ and $[j, (0, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)]$ holds and, furthermore, that $n = 0$. This means that the input head scans the left endmarker ¢ and the counter message is $\mathtt{t}_1$. Consequently, transition 2 applies which moves the input head a step to the right and leaves the counter unchanged. Again, we note that the counter message cannot change from $\mathtt{t}_1$ to $\mathtt{t}_0$ in the whole process of scanning $u := w[1, j' - 1]$, so we conclude $|u|_a = 0$, which implies $u \in \{\#_1\}^*$ and, hence, $u \in \widetilde{\mathbf{L}}_{n'} = \widetilde{\mathbf{L}}_0$. If we have $n \geq 1$, then, as the counter message is $\mathtt{t}_0$, transition 1 applies first and the input head is moved one step to the right and the counter configuration stays $(0, n)$. Since the counter message is still $\mathtt{t}_0$ we can conclude in the same way as before that $u = a^n \cdot (\#_1 \cdot a^n)^m$ for some $m \geq 1$, thus, $u \in \widetilde{\mathbf{L}}_n$.

Next, we show that if $j \neq 0$, then $w[j+1, j'-1] \in \widetilde{\mathbf{L}}_{n'}$ implies $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n', n')]$ and if $j = 0$, then $w[j + 1, j' - 1] \in \widetilde{\mathbf{L}}_n$ implies $[j, (0, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)]$. To this end, we first observe that $u := w[j + 1, j' - 1] \in \widetilde{\mathbf{L}}_{n'}$ means that there exists an $m \in \mathbb{N}_0$ with $u = a^{n'} \cdot (\#_1 \cdot a^{n'})^m$. Consequently, there are 4 possible cases of how u may look like:

1. If $n' = m = 0$, then $u = \varepsilon$,
2. if $n' = 0$ and $m \geq 1$, then $u = (\#_1)^m$,
3. if $n' \geq 1$ and $m = 0$, then $u = a^{n'}$,
4. if $n' \geq 1$ and $m \geq 1$, then $u = a^{n'} \cdot (\#_1 \cdot a^{n'})^m$.

We can show that for each of the 4 cases mentioned above, $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n', n')]$ holds. So we assume that the input head scans position j and the counter configuration is (n, n). This means that transition 5 is applicable, so the input head is moved to position $j + 1$ and the counter changes into configuration $(0, n')$. In cases 1 and 2 we assume the guessed counter bound to be $n' = 0$; hence, the counter message is $\mathtt{t}_1$ until the next reset is performed, i. e., until the input head reaches position j'. So in case 1 the input head is moved to the next occurrence of $\#_2$ and the counter changes into configuration $(0, 0) = (n', n')$. For case 2 the input head is moved over $u = (\#_1)^m$ by successively applying transition 4. The counter message stays $\mathtt{t}_1$ the whole time until eventually the next occurrence of $\#_2$ is reached with counter configuration $(0, 0) = (n', n')$.

If, on the other hand, u starts with a factor $a^{n'}$, as in cases 3 and 4, then we assume the guessed counter bound to be n', which allows $M_{\mathbf{L}}$ to apply transition 3 n' times until the input head scans the symbol to the right of the last a and the counter configuration is (n', n'). In case 3 this symbol is already the next occurrence of $\#_2$ at position j' and in case 4 this symbol is another $\#_1$ followed by another factor $a^{n'}$. Hence, transition 4 applies and the same procedure starts over again until the next occurrence of $\#_1$ is scanned with counter configuration (n', n'), and so on. Eventually, the next occurrence of $\#_2$ at position j' is scanned with counter configuration (n', n'). Consequently, for all of the 4 cases $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n', n')]$ holds.

We can analogously show that $w[j + 1, j' - 1] \in \widetilde{\mathbf{L}}_n$ implies $[j, (0, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)]$ if $j = 0$. The only difference is that the first transition is not transition 5 anymore, but transition 1 or 2 depending on whether or not $n = 0$. From then on we can apply exactly the same argumentation in order to show for all of the 4 cases above that $[j, (0, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)]$. □ (*Claim* (1))

Claim (2). Let $¢w\$$, $w \in \{\#_1, \#_2, a\}^*$, be an arbitrary input for $M_{\mathbf{L}_k}$, and let j, j' be arbitrarily chosen with $1 \leq j < j' \leq |w| + 1$, $w[j], w[j'] \in \{\#_2, \$\}$ and $w[j''] \neq \#_2$, $j < j'' < j'$. Furthermore, let $c_1, c_2, \ldots, c_m$ be the initial part of a computation of $M_{\mathbf{L}_k}$ on w and the counter has been reset at least $k - 1$ times in this initial part of the computation. Then, for all n with $0 \leq n \leq |w|$, the following holds: If $n = 0$, then

$$[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)] \text{ if and only if } w[j + 1, j' - 1] \in \{\#_1\}^*,$$

and if $n \geq 1$, then

$$[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)] \text{ if and only if } w[j + 1, j' - 1] \in (\{\#_1\} \cdot \widetilde{L}_n).$$

Proof. (*Claim* (2)) First, we show that $[j, (0, 0)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (0, 0)]$ implies $w[j + 1, j' - 1] \in \{\#_1\}^*$ and that, for every $n \in \mathbb{N}$, $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)]$ implies $w[j+1, j'-1] \in (\{\#_1\} \cdot \widetilde{L}_n)$. To this end, we assume that $[j, (0, 0)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (0, 0)]$ and take a closer look at this computation. For the sake of convenience, we define $u := w[j + 1, j' - 1]$. Initially, the input head scans $w[j] = \#_2$ and the counter configuration is $(0, 0)$. Thus, the counter message is $\mathtt{t_1}$ and therefore transition 5 applies. Since the counter has already been reset at least $k - 1$ times, transition 5 is interpreted as $(\#_2, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$, hence, the input head is moved to position $j + 1$ and the counter configuration stays $(0, 0)$. Since the counter message is $\mathtt{t_1}$, it is not possible that u starts with the symbol a. So it must start with symbol $\#_1$. In a next step transition 4 applies which moves the input head a step further to the right and does not change the counter configuration. We can conclude in the same way as before, that the next symbol of u must be $\#_1$ and, thus, $u \in \{\#_1\}^*$.

Next, we assume that $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k}, w} [j', (n, n)]$ for an arbitrary $n \in \mathbb{N}$. Again, the input head scans $w[j] = \#_2$ and the counter configuration is (n, n). Thus, the counter message is $\mathtt{t_1}$ and therefore transition $(\#_2, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$ applies.

Hence, the input head is moved to position $j + 1$ and the counter configuration stays (n, n). Since the counter message is still $\mathtt{t_1}$, symbol a cannot occur next, so the next symbol must be $\#_1$. Now, transition 4 applies which moves the input head another step further to the right and increments the counter to configuration $(0, n)$. Since $n \geq 1$, we can conclude that the counter message is now $\mathtt{t_0}$ so the second symbol of u must be a. In the next step, transition 3 applies which increments the counter and moves the input head a step further to the right. This is repeated as long as the input head scans a and the counter message does not change to $\mathtt{t_1}$. It is neither possible that another a is scanned when the counter message changes to $\mathtt{t_1}$ nor that the counter message is still $\mathtt{t_0}$ when the first symbol different from a is scanned. Consequently, u starts with the prefix $\#_1 \cdot a^n$. The next symbol can be the occurrence of $\#_2$ at position j' or another occurrence of $\#_1$ which implies that transition 4 applies. As before, transition 4 moves the input head a step further to the right and changes the counter configuration into $(0, n)$, so we can conclude, that the next part of u is again a^n; thus, $u \in (\{\#_1\} \cdot \widetilde{L}_n)$.

It remains to show the converse of the two statements, i. e., $w[j + 1, j' - 1] \in \{\#_1\}^*$ implies $[j, (0, 0)] \vdash^*_{M_{\mathbf{L}_k},w} [j', (0, 0)]$ and $w[j + 1, j' - 1] \in (\{\#_1\} \cdot \widetilde{L}_n)$ implies $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k},w} [j', (n, n)]$. We assume that $w[j + 1, j' - 1] \in \{\#_1\}^*$ and consider the computation of $M_{\mathbf{L}_k}$ on w at configuration $[j, (0, 0)]$. By first applying transition $(\#_2, \mathtt{t_1}) \to_\delta (\mathtt{1}, \mathtt{0})$ and then successively applying transition 4, the input head is moved to position j' and the counter configuration is not changed. Consequently, $[j, (0, 0)] \vdash^*_{M_{\mathbf{L}_k},w} [j', (0, 0)]$.

Next, we assume that $w[j+1, j'-1] \in (\{\#_1\} \cdot \widetilde{L}_n)$. More precisely, $w[j+1, j'-1] = \#_1 \cdot a^n \cdot \#_1 \cdot a^n \cdot \#_1 \cdot \cdots \cdot \#_1 \cdot a^n$. We consider the configuration $[j, (n, n)]$. Again, transition $(\#_2, \mathtt{t_1}) \to_\delta (\mathtt{1}, \mathtt{0})$ applies first. Since $\#_1$ is scanned next and the counter configuration is (n, n), transition 4 is applied next and therefore, the counter changes to $(0, n)$ and the input head scans the first occurrence of a. Now, by successively applying transition 3, the input head is moved over the first factor a^n until it scans the second occurrence of $\#_1$ and the counter configuration is (n, n) again. This procedure repeats until configuration $[j', (n, n)]$ is reached and, thus, $[j, (n, n)] \vdash^*_{M_{\mathbf{L}_k},w} [j', (n, n)]$. □ (*Claim* (2))

We now use Claims (1) and (2) to show that $\mathbf{L}_k \subseteq L(M_{\mathbf{L}_k})$. To this end, let $w \in \mathbf{L}_k$ be arbitrarily chosen. We first consider the case $w \in \mathbf{L}_{k,1}$, which implies that $w = u_1 \cdot \#_2 \cdot u_2 \cdot \#_2 \cdot \cdots \cdot \#_2 \cdot u_{k'}$, where $k' \leq k$ and $u_i \in \widetilde{\mathbf{L}}$, $1 \leq i \leq k'$. More precisely, let $u_i \in \widetilde{\mathbf{L}}_{n_i}$, $1 \leq i \leq k'$, and $j_i := |u_1 \cdot \#_2 \cdot u_2 \cdot \#_2 \cdot \cdots \cdot \#_2 \cdot u_i| + 1$, $1 \leq i \leq k'$. Now we can apply Claim 1 and conclude that $[0, (0, n_1)] \vdash^*_{M_{\mathbf{L}_k},w} [j_1, (n_1, n_1)]$ and, for every i, $1 \leq i \leq k' - 1$, $[j_i, (n_i, n_i)] \vdash^*_{M_{\mathbf{L}_k},w} [j_{i+1}, (n_{i+1}, n_{i+1})]$. Consequently, $[0, (0, n_1)] \vdash^*_{M_{\mathbf{L}_k},w} [|w| + 1, (n_k, n_k)]$, i. e., $M_{\mathbf{L}_k}$ guesses $n_2, n_3, \ldots, n_k$

as counter bounds and is initialised with counter bound n_1. Therefore, we conclude $w \in L(M_{\mathbf{L}_k})$ and, thus, $\mathbf{L}_{k,1} \subseteq L(M_{\mathbf{L}_k})$.

If $w \in \mathbf{L}_{k,2}$ we can write $w = u_1 \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_k \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_{k'}$, $u_i \in \widetilde{\mathbf{L}}$, $1 \leq i \leq k$. Furthermore, if $u_k \in \widetilde{\mathbf{L}}_0$, then $u_{i'} \in \widetilde{\mathbf{L}}_0$, $k+1 \leq i' \leq k'$, and if $u_k \in \widetilde{\mathbf{L}}_n$, $n \in \mathbb{N}$, then $u_{i'} \in (\{\#_1\} \cdot \widetilde{\mathbf{L}}_n)$, $k+1 \leq i' \leq k'$. As before, let $u_i \in \widetilde{\mathbf{L}}_{n_i}$, $1 \leq i \leq k$, and $j_i := |u_1 \cdot \#_2 \cdot u_2 \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_i| + 1$, $1 \leq i \leq k'$. In the same way as before, we can apply Claim 1 to show that $[0, (0, n_1)] \vdash_{M_{\mathbf{L}_k}} [j_k, (n_k, n_k)]$. Let us now assume that $u_k \in \widetilde{\mathbf{L}}_0$ which, by definition of $L_{k,2}$, implies $u_{i'} \in \widetilde{\mathbf{L}}_0$, $k+1 \leq i' \leq k'$. Using Claim (2), we can conclude that $[j_i, (n_k, n_k)] \vdash^*_{M_{\mathbf{L}_k},w} [j_{i+1}, (n_k, n_k)]$, $k+1 \leq i \leq k'$. Hence, $[j_{k+1}, (n_k, n_k)] \vdash^*_{M_{\mathbf{L}_k},w} [j_{k'}, (n_k, n_k)]$, which, together with $[0, (0, n_1)] \vdash^*_{M_{\mathbf{L}_k}} [j_k, (n_k, n_k)]$, implies $[0, (0, n_1)] \vdash^*_{M_{\mathbf{L}_k}} [j_{k'}, (n_k, n_k)]$; thus, $w \in M_{\mathbf{L}_k}$.

For the case that $u_k \in \widetilde{\mathbf{L}}_n$, $n \in \mathbb{N}$, we can show that $[j_{k+1}, (n_k, n_k)] \vdash^*_{M_{\mathbf{L}_k},w} [j_{k'}, (n_k, n_k)]$ by applying Claim (2) in the same way as before. So we can conclude that $\mathbf{L}_k \subseteq L(M_{\mathbf{L}_k})$.

It remains to prove the converse statement, i. e., $L(M_{\mathbf{L}_k}) \subseteq \mathbf{L}_k$. To this end, let $w \in L(M_{\mathbf{L}_k})$ be arbitrarily chosen. Obviously, there is a $k' \in \mathbb{N}$ such that $|w|_{\#_2} = k' - 1$ and, thus, $w = u_1 \cdot \#_2 \cdot u_2 \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_{k'}$, where $u_i \in \{a, \#_1\}^*$, $1 \leq i \leq k'$. Furthermore, let $j_i := |u_1 \cdot \#_2 \cdot u_2 \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_i| + 1$, $1 \leq i \leq k'$. We shall first consider the case that $k' \leq k$ and deal with the case $k < k'$ later on. We note that $k' \leq k$ implies that the number of occurrences of $\#_2$ is at most $k-1$. Hence, in an accepting computation of $M_{\mathbf{L}_k}$ on w, each time the input head scans $\#_2$ the counter is reset by applying transition 5. Now let $n_2, n_3, \dots, n_{k'}$ be the counter bounds guessed in an accepting computation of $M_{\mathbf{L}_k}$ on w and, furthermore, let n_1 be the counter bound the counter is initialised with. So we can conclude that $[0, (0, n_1)] \vdash^*_{M_{\mathbf{L}_k},w} [j_1, (n_1, n_1)]$, $[j_i, (n_i, n_i)] \vdash^*_{M_{\mathbf{L}_k},w} [j_{i+1}, (n_{i+1}, n_{i+1})]$, $1 \leq i \leq k' - 1$. Referring to Claim (1) this implies that $u_i \in \widetilde{\mathbf{L}}_{n_i}$, $1 \leq i \leq k'$, and therefore $w \in \mathbf{L}_k$.

In case $k < k'$, we can write w as $w = u_1 \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_k \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_{k'}$, and in the same way as before we can conclude that $u_i \in \widetilde{\mathbf{L}}_{n_i}$, $1 \leq i \leq k$. We know, furthermore, that w is accepted by $M_{\mathbf{L}_k}$ and, thus, $[j_i, (n_k, n_k)] \vdash^*_{M_{\mathbf{L}_k},w} [j_{i+1}, (n_k, n_k)]$, $k \leq i \leq k' - 1$. Now, if $n_k = 0$, Claim (2) implies that $u_i \in \{\#_1\}^*$, $k+1 \leq i \leq k'$, and if $n_k \geq 1$, $u_i \in (\{\#_1\} \cdot \widetilde{\mathbf{L}}_{n_k})$, $k+1 \leq i \leq k'$, is implied. Consequently, in both cases, $w \in \mathbf{L}_k$, which shows $L(M_{\mathbf{L}_k}) \subseteq \mathbf{L}_k$. □

As described above, our next goal is to state that $\mathbf{L}_k$ cannot be accepted by any 1SL-NBMCA$_{k-2}$(1). To this end we first observe that for every $M \in$ 1SL-NBMCA$_k$(1), $k \in \mathbb{N}$, that accept a language $\mathbf{L}_{k'}$, a certain property related to the fact that, by definition, a word $w \in \mathbf{L}_{k'}$ can have $k' - 1$ factors of form $c \cdot a^n \cdot \#_2 \cdot a^{n'} \cdot c'$, $n, n' \in \mathbb{N}$, $c \neq a \neq c'$, must be satisfied. The next lemma states that M must reset its counter at least once in the process of moving the input head over any such factor.

Lemma 4.29. *Let $k, k' \in \mathbb{N}$, $k' \geq 2$, $M \in$ 1SL-NBMCA$_k(1)$ with a transition function δ and $L(M) = \mathbf{L}_{k'}$. Let furthermore $C := (c_1, c_2, \dots, c_m)$ be an arbitrary accepting computation of M on some arbitrarily chosen $w := u_1 \cdot \#_2 \cdot u_2 \cdot \#_2 \cdot \dots \cdot \#_2 \cdot u_{k'}$ with $u_i \in \widetilde{\mathbf{L}}_{n_i}$, $n_i \geq 2k+1$, $|u_i|_{\#_1} \geq 1$, $1 \leq i \leq k'$, $n_i \neq n_{i+1}$, $1 \leq i \leq k'-1$, and let j, j', $1 \leq j < j' \leq |w|$, such that $w[j, j'] = \#_1 \cdot a^{n_i} \cdot \#_2 \cdot a^{n_{i+1}} \cdot \#_1$ with $1 \leq i \leq k' - 1$. If, for some l, l', $1 \leq l < l' \leq m$, and some $p_i, q_i \in \mathbb{N}_0$, $1 \leq i \leq 4$,*

$$c_l, \dots, c_{l'} = [j, (p_1, q_1)], [j+1, (p_2, q_2)], \dots, [j'-1, (p_3, q_3)], [j', (p_4, q_4)]\,,$$

then there exists an i, $l+1 \leq i \leq l'-1$, such that c_i is converted into c_{i+1} by a transition of form $(b, x) \rightarrow_\delta (y, \mathtt{r})$, $b \in \Sigma$, $x \in \{\mathtt{t_0}, \mathtt{t_1}\}$, $y \in \{\mathtt{0}, \mathtt{1}\}$.

Proof. We shall prove the statement of the lemma by first proving two claims establishing certain properties of M, the 1SL-NBMCA$_k(1)$ that recognises $\mathbf{L}_{k'}$. The first claim concerns the way how M scans occurrences of a.

Claim (1). In C, the transitions

T$_1$ $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{0}, \mathtt{1})$,

T$_2$ $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$,

T$_3$ $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$,

T$_4$ $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{0}, \mathtt{1})$,

T$_5$ $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$,

are not applied and the transition $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$ is applied.

Proof. (*Claim* (1)) We shall first show that none of the transitions T$_1$ to T$_5$ is applied in C. To this end, we observe that since $n_i \geq 2k+1$, $1 \leq i \leq k'$, and there are at most k resets possible in C, we can conclude that there are at least two consecutive occurrences of a in w such that only non-reseting transitions are performed while these occurrences are scanned by the input head. We assume that these occurrences are at positions $\widehat{p}$ and $\widehat{p}+1$. Next, we show that it is not possible that any of the transitions T$_1$ to T$_5$ apply when the input head scans position $\widehat{p}$. To this end, we assume to the contrary that one of these transitions is applied in configuration $[\widehat{p}, (p, q)]$, $p, q \in \mathbb{N}$, $p \leq q$, and then show that a word is accepted by M that is not an element of $\mathbf{L}_{k'}$, which is a contradiction.

If transitions T$_2$ or T$_5$ apply in configuration $[\widehat{p}, (p, q)]$, then the input head is moved over the occurrence of a at position $\widehat{p}$ without changing the counter value.

Thus, the counter message does not change. This directly implies that the word $w[1,\widehat{p}] \cdot a \cdot w[\widehat{p}+1,-] \notin \mathbf{L}_{k'}$ is accepted by M as well, which is a contradiction.

If transition T_4 applies in configuration $[\widehat{p},(p,q)]$, then the transition $[\widehat{p},(p+1,q)]$ is reached and, thus, T_4 is repeated until the counter reaches its bound, i. e., M enters configuration $[\widehat{p},(q,q)]$. Since the computation is accepting, a next transition must be defined and this transition must move the head as otherwise no transition is defined that moves the head while an occurrence of a is scanned. Furthermore, by assumption, this transition is non-reseting. Since we have already ruled out transition T_2, the only possible next transition is T_3. This implies that configuration $[\widehat{p}+1,(0,q)]$ is reached. Consequently, the word $w[1,\widehat{p}]\cdot a\cdot w[\widehat{p}+1,-] \notin \mathbf{L}_{k'}$ is accepted as well and, again, a contradiction is obtained.

Next we assume that transition T_1 applies in configuration $[\widehat{p},(p,q)]$. If q, the current counter bound, equals 0, then the counter message cannot change and the automaton is in an infinite loop, which contradicts the fact that the computation C is accepting. So we assume that $q \geq 1$ which implies that configuration $[\widehat{p},(0,q)]$ is reached by applying T_1. Since C is accepting a next transition must be applicable that is non-reseting and moves the input head. We have already ruled out transition T_5. Thus, the only possible next transition is $(a,\mathtt{t}_0) \rightarrow_\delta (\mathtt{1},\mathtt{1})$ and therefore the configuration $[\widehat{p}+1,(1,q)]$ is reached. We observe that this implies that $w[1,\widehat{p}]\cdot a^q\cdot w[\widehat{p}+1,-] \notin \mathbf{L}_{k'}$ is accepted by M as well, which is a contradiction.

It remains to consider the case that T_3 is applied in configuration $[\widehat{p},(p,q)]$. If $q = 0$, then the counter message does not change by applying transition T_3. This implies that the effect of transition T_3 is the same as of transition T_2. Hence, we can show in a similar way as before that $w[1,\widehat{p}] \cdot a \cdot w[\widehat{p}+1,-] \notin \mathbf{L}_{k'}$ is accepted by M. Since this is a contradiction, we conclude that $q \geq 1$ and observe that configuration $[\widehat{p}+1,(0,q)]$ is reached by applying T_3. Again, as C is accepting, a next configuration must be defined. We recall that we assume that no reseting transition is applied while the input head scans positions $\widehat{p}$ and $\widehat{p}+1$. Therefore, the next transition is non-reseting, but does not necessarily move the input head. The only possible transitions of that kind are T_4, T_5 and $(a,\mathtt{t}_0) \rightarrow_\delta (\mathtt{1},\mathtt{1})$. Since $w[\widehat{p}+1] = a$, we can conclude that T_4 and T_5 cannot be applied in configuration $[\widehat{p}+1,(0,q)]$ in exactly the same way as we have already shown above that T_4 and T_5 cannot be applied in configuration $[\widehat{p},(p,q)]$. Therefore the only possible transition left is transition $(a,\mathtt{t}_0) \rightarrow_\delta (\mathtt{1},\mathtt{1})$. Similarly as before, we can conclude that in this case M accepts $w[1,\widehat{p}] \cdot a^{q+1} \cdot w[\widehat{p}+1,-] \notin \mathbf{L}_{k'}$ and, thus, we obtain a contradiction.

We conclude that the transition $(a,\mathtt{t}_0) \rightarrow_\delta (\mathtt{1},\mathtt{1})$ is the only possible transition that can be applied when configuration $[\widehat{p},(p,q)]$ is reached. This proves the claim. □ (*Claim* (1))

The next claim states that M cannot move the input head over an occurrence of $\#_1$ or $\#_2$ by a non-reseting transition if the counter message is $\mathtt{t_0}$.

Claim (2). For every $b \in \{\#_1, \#_2\}$, the transitions

T$_6$ $(b, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$,

T$_7$ $(b, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$

are not defined.

Proof. (*Claim* (2)) We assume to the contrary that, for some $b \in \{\#_1, \#_2\}$, transition T_6 or T_7 is defined and use our accepting computation C of M on w to obtain a contradiction, i. e., we show that M accepts a word that is not an element of $\mathbf{L}_{k'}$.

As we have already shown in Claim (1), there must be a position $\widehat{p}$, $1 \leq \widehat{p} \leq |w|$, such that $c_i := [\widehat{p}, (p, q)]$ is converted into $c_{i+1} := [\widehat{p} + 1, (p + 1, q)]$ by transition $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$. Furthermore, we can conclude that $p < q$.

We now assume that transition T$_6$ is defined and consider the input $w' := w[1, \widehat{p}-1] \cdot b \cdot w[\widehat{p}, -]$, i. e., we insert an occurrence of b to the left of the occurrence of a at position $\widehat{p}$. It is not possible that in configuration c_{i-1} the input head is located at position $\widehat{p}$ as well, as this implies the application of a transition other than $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$ which, by Claim (1), must be reseting. Hence, there is a computation of M on w' that is identical to C up to the first i elements. So configuration $[\widehat{p}, (p, q)]$ is reached and, as $w'[\widehat{p}] = b$ and $p < q$, T$_6$ applies and changes M into configuration $[\widehat{p} + 1, (p, q)]$. Now, as $w'[\widehat{p} + 1, -] = w[\widehat{p}, -]$, it is possible that the computation terminates with the last $m - i$ elements of C, where the first component of each configuration has increased by 1. Hence, w' is accepted by M.

Next, we assume that transition T$_7$ is defined and consider the input $w'' := w[1, \widehat{p} - 1] \cdot b \cdot w[\widehat{p} + 1, -]$, i. e., we substitute the occurrence of a at position $\widehat{p}$ by an occurrence of b. There is a computation of M on w'' that is identical to C up to the first i elements. So configuration $[\widehat{p}, (p, q)]$ is reached and, as $w''[\widehat{p}] = b$ and $p < q$, T$_7$ applies and changes M into configuration $[\widehat{p} + 1, (p + 1, q)]$. Now, as $w''[\widehat{p} + 1, -] = w[\widehat{p} + 1, -]$, it is possible that the computation terminates with the last $m - (i + 1)$ elements of C. Hence, w'' is accepted by M.

In order to conclude the proof, it remains to show that $w' \notin \mathbf{L}_{k'}$ and $w'' \notin \mathbf{L}_{k'}$ for every $b \in \{\#_1, \#_2\}$. We recall that w' is obtained from w by inserting an occurrence of b to the left of an occurrence of a and w'' is obtained from w by substituting an occurrence of a by an occurrence of b. If $b = \#_1$, then there exists a factor $c \cdot a^n \cdot \#_1 \cdot a^{n'} \cdot c'$ in w' (or w''), where $n \neq n'$ and $c \neq a \neq c'$; hence $w' \notin \mathbf{L}_{k'}$ (or $w'' \notin \mathbf{L}_{k'}$, respectively). If $b = \#_2$ and $\widehat{p}$ is such that $w[\widehat{p} - 1] \notin \{¢, \#_2\}$, then there also exists a factor $c \cdot a^n \cdot \#_1 \cdot a^{n'} \cdot c'$ in w' (or w'', respectively), where $n \neq n'$ and $c \neq a \neq c'$. Thus, $w' \notin \mathbf{L}_{k'}$ (or $w'' \notin \mathbf{L}_{k'}$, respectively).

It remains to consider the case where $w[\widehat{p} - 1] \in \{¢, \#_2\}$ and $b = \#_2$. First, we observe that in this case w'' must have a factor $c \cdot \#_2 \cdot a^{n-1} \cdot \#_1 \cdot a^n \cdot c'$, where $c \in \{¢, \#_2\}$ and $c \neq a$. Consequently, $w'' \notin \mathbf{L}_{k'}$. Regarding w', we do not substitute an occurrence of a by an occurrence of $\#_2$, but we insert it to the left of the occurrence of a at position $\widehat{p}$. So if $w[\widehat{p} - 1] \in \{¢, \#_2\}$, then it is possible that $w' \in \mathbf{L}_{k'}$. However, we can show that there must exist a position $\widehat{p}'$, $1 \leq \widehat{p}' \leq |w|$, such that $w[\widehat{p}' - 1] \notin \{¢, \#_2\}$ and in the computation C a configuration $[\widehat{p}', (p, q)]$ is changed into $[\widehat{p}' + 1, (p + 1, q)]$ by transition $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$. To this end, we assume that there exists no $\widehat{p}'$, $1 \leq \widehat{p}' \leq |w|$, such that $w[\widehat{p}' - 1] \neq \{¢, \#_2\}$ and in the computation C a configuration $[\widehat{p}', (p, q)]$ is changed into $[\widehat{p}' + 1, (p + 1, q)]$ by transition $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$. This implies that the input head is moved over all the occurrences of a at a position $\widehat{p}''$ with $w[\widehat{p}'' - 1] = a$ by a transition of form $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, x)$, and by Claim (1) of this lemma we can conclude that $x = \mathtt{r}$. Since there are $n_1 - 1$ such occurrences of a that require an application of the transition $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{r})$ in the prefix a^{n_1} of w and since $n_1 \geq 2k + 1$, we can conclude that all possible k resets are performed in the process of moving the input head over the prefix a^{n_1} of w. Furthermore, after these k applications of $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{r})$, the transition $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{r})$ will be interpreted as $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$ for all further occurrences of a in the prefix a^{n_1}. This implies that in the computation C the transition $(a, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$ is applied which is a contradiction according to Claim (1). This shows that there must exist a position $\widehat{p}'$, $1 \leq \widehat{p}' \leq |w|$, such that $w[\widehat{p}' - 1] \notin \{¢, \#_2\}$ and in the computation C a configuration $[\widehat{p}', (p, q)]$ is changed into $[\widehat{p}' + 1, (p + 1, q)]$ by transition $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$. Consequently, we can construct a w' with respect to that position $\widehat{p}'$ in the way described above and there exists an accepting computation of M on w', but $w' \notin \mathbf{L}_{k'}$. This concludes the proof of Claim (2). □ (*Claim* (2))

We can now prove the statement of the lemma. To this end, we assume that

$$c_l, \ldots, c_{l'} = [j, (p_1, q_1)], [j + 1, (p_2, q_2)], \ldots, [j' - 1, (p_3, q_3)], [j', (p_4, q_4)]\,,$$

such that, for every i, $l + 1 \leq i \leq l' - 1$, c_i is converted into c_{i+1} by a transition of form $(b, x) \rightarrow_\delta (y, z)$, $b \in \Sigma$, $x \in \{\mathtt{t_0}, \mathtt{t_1}\}$, $y, z \in \{\mathtt{0}, \mathtt{1}\}$. We recall that $w[j, j'] = \#_1 \cdot a^{n_i} \cdot \#_2 \cdot a^{n_{i+1}} \cdot \#_1$ for some i, $1 \leq i \leq k' - 1$. Since the input head is moved from position j to position $j + 1$, we know that the transition that converts $[j, (p_1, q_1)]$ into $[j + 1, (p_2, q_2)]$ is of form $(\#_1, x) \rightarrow_\delta (\mathtt{1}, y)$. If $y \neq \mathtt{r}$, then, by Claim (2), $x = \mathtt{t_1}$ is implied and if furthermore $y = 0$, then the occurrence of a at position $j + 1$ is reached with counter message $\mathtt{t_1}$. Now, using Claim (1), we can conclude that the only possible next transition must reset the counter, which contradicts our assumption. Consequently, the transition that converts $[j, (p_1, q_1)]$ into $[j + 1, (p_2, q_2)]$ is either $(\#_1, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$ or a transition of form $(\#_1, x) \rightarrow_\delta (\mathtt{1}, \mathtt{r})$. We note that regardless of which of the possible transitions apply, the input head is moved one step to the right and the counter configuration changes to $(0, q_2)$. We define $v := w[j + 1, j' - 1] = a^{n_i} \cdot \#_2 \cdot a^{n_{i+1}}$. By Claims (1) and (2) and the assumption that the input head is moved over v without counter resets, we conclude that the input head is moved over all the occurrences of a in v by applying transition $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$. So this transition applies until either the input head scans $\#_2$ or the counter message changes to $\mathtt{t_1}$. If the counter message changes to $\mathtt{t_1}$ while still an occurrence of a is scanned by the input head, then, by Claim (1), the next transition would reset the counter, which is not possible; so we can conclude that $q_2 \geq n_i$. If the input head reaches the occurrence of $\#_2$ with a counter message of $\mathtt{t_0}$, then the transition $(\#_2, \mathtt{t_0}) \rightarrow_\delta (\mathtt{0}, \mathtt{1})$ applies, since we assume that the counter is not reset and a non-reseting transition that moves the input head while $\#_2$ is scanned and the counter message is $\mathtt{t_0}$ is not possible, according to Claim (2). However, this implies that the counter is incremented without moving the input head until the counter message changes to $\mathtt{t_1}$. We conclude that the occurrence of a to the left of the occurrence $\#_2$ could be deleted and the computation would still be accepting. This is clearly a contradiction. Therefore the input head reaches $\#_2$ exactly with counter message $\mathtt{t_1}$ and, thus, $q_2 = n_i$. We have now reached the configuration where the input head scans $\#_2$ and the counter message is $\mathtt{t_1}$. If the next transition does not move the input head, then it must increment the counter, as otherwise the transition would be accepting which, by definition, is not possible. This results in the configuration where still $\#_2$ is scanned but with a counter message of $\mathtt{t_0}$. In the same way as before, by applying Claim (2), we can conclude that for such a configuration no non-reseting transition that moves the input head is defined. Hence, the automaton stops, which is a contradiction. Consequently the next transition that applies when $\#_2$ is scanned is transition $(\#_2, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, z)$. Furthermore, $z = \mathtt{1}$, as otherwise the first occurrence of a to the right of $\#_2$ is reached with counter message $\mathtt{t_1}$, which, as already shown above, is not possible. So transition $(\#_2, \mathtt{t_1}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$ applies and then again several times transition $(a, \mathtt{t_0}) \rightarrow_\delta (\mathtt{1}, \mathtt{1})$. For the same reasons as before we can conclude that the counter message must not change to $\mathtt{t_1}$ as long as occurrences of a are scanned and; thus, $q_2 \geq n_{i+1}$. If we reach the occurrence of $\#_1$ at position j' with a counter message of $\mathtt{t_0}$, we have several possibilities. If a transition applies that does not reset the counter, then, by Claim (2), it must be $(\#_1, \mathtt{t_0}) \rightarrow_\delta (\mathtt{0}, \mathtt{1})$. On the other hand, since M is now in configuration $c_{l'}$, it is also possible that a transition of form $(\#_1, \mathtt{t_0}) \rightarrow_\delta (x, \mathtt{r})$ applies. However, for all these cases we observe that if we would delete the occurrence of a to the left of the occurrence of $\#_1$ at position j', then the changed input would still be accepted, which is a contradiction. So we conclude that the input head reaches the occurrence of $\#_1$ exactly with counter message $\mathtt{t_1}$. This implies $q_2 = n_{i+1}$ and, hence, $n_i = n_{i+1}$, which is a contradiction. This concludes the proof Lemma 4.29. $\square$

Now we are able to show that the language $\mathbf{L}_k$, that can be recognised by a 1SL-NBMCA$_{k-1}(1)$ (Theorem 4.28), cannot be recognised by a 1SL-NBMCA$_{k-2}(1)$.

Theorem 4.30. *For every $k \in \mathbb{N}$ with $k \geq 2$, $\mathbf{L}_k \notin \mathcal{L}($1SL-NBMCA$_{k-2}(1))$.*

Proof. We assume to the contrary that there exists an $M \in$ 1SL-NBMCA$_{k-2}(1)$ with $L(M) = \mathbf{L}_k$. Let $w := a^{n_1} \cdot \#_1 \cdot a^{n_1} \cdot \#_2 \cdot a^{n_2} \cdot \#_1 \cdot a^{n_2} \cdot \#_2 \cdot \cdots \cdot \#_2 \cdot a^{n_k} \cdot \#_1 \cdot a^{n_k}$, with $n_i \in \mathbb{N}$, $n_i \geq 2k+1$, $1 \leq i \leq k$, and $n_i \neq n_{i+1}$, $1 \leq i \leq k-1$. Obviously, $w \in \mathbf{L}_k$ and w satisfies the conditions of Lemma 4.29. We observe that in w, there are $k-1$ factors of form $\#_1 \cdot a^{n_i} \cdot \#_2 \cdot a^{n_{i+1}} \cdot \#_1$, but in an accepting computation of M on w, there are at most $k-2$ resets possible. Hence, there must be an i, $1 \leq i \leq k-1$, such that the input head is moved over the factor $a^{n_i} \cdot \#_2 \cdot a^{n_{i+1}}$ without performing a reset. According to Lemma 4.29 this is not possible, so we obtain a contradiction. $\square$

This proves that for every $k \in \mathbb{N}$ there exists a language that can be recognised by a 1SL-NBMCA$_k(1)$, but cannot be recognised by a 1SL-NBMCA$_{k-1}(1)$. Next, we consider the converse question, i. e., whether or not there are languages that can be recognised by a 1SL-NBMCA$_k(1)$, but cannot be recognised by any 1SL-NBMCA$_{k+1}(1)$. It turns out that the existence of such languages can be shown in a non-constructive way by applying Theorems 4.28 and 4.30 and a simple reasoning about the following subsets of the classes 1SL-NBMCA$_k(1)$, $k \in \mathbb{N}$:

Definition 4.31. For every $k \in \mathbb{N}$, let 1SL-NBMCA$_k^\Sigma(1)$ be the class of all automata in 1SL-NBMCA$_k(1)$ that are defined over Σ.

By definition, all $M \in$ 1SL-NBMCA$_k^\Sigma(1)$ have just one counter and are defined over the same alphabet Σ. Hence, for all $k \in \mathbb{N}$, the sets 1SL-NBMCA$_k^\Sigma(1)$, have the same constant cardinality:

Proposition 4.32. *There exists an constant $\widehat{m} \in \mathbb{N}$ such that, for every $k \in \mathbb{N}$, $|$ 1SL-NBMCA$_k^\Sigma(1)| = \widehat{m}$.*

We note that, in general, we cannot assume that $|\mathcal{L}($1SL-NBMCA$_k^\Sigma(1))| =$ $|$ 1SL-NBMCA$_k^\Sigma(1)|$, thus, $|\mathcal{L}($1SL-NBMCA$_k^\Sigma(1))| \leq \widehat{m}$, $k \in \mathbb{N}$. However, it is straightforward to show that there must exist infinitely many $k \in \mathbb{N}$, such that $|\mathcal{L}($1SL-NBMCA$_k^\Sigma(1))| = |\mathcal{L}($1SL-NBMCA$_{k+1}^\Sigma(1))|$ and then Theorems 4.28 and 4.30

imply that these classes are incomparable. This result can be easily extended to the classes $\mathcal{L}(\text{1SL-NBMCA}_k(1))$, $k \in \mathbb{N}$.

Theorem 4.33. *There exist infinitely many* $k \in \mathbb{N}$*, such that* $\mathcal{L}(\text{1SL-NBMCA}_k(1))$ *and* $\mathcal{L}(\text{1SL-NBMCA}_{k+1}(1))$ *are incomparable.*

Proof. We first observe that if, for some $k \in \mathbb{N}$, the classes $\mathcal{L}(\text{1SL-NBMCA}^{\Sigma}_k(1))$ and $\mathcal{L}(\text{1SL-NBMCA}^{\Sigma}_{k+1}(1))$ are incomparable, then also $\mathcal{L}(\text{1SL-NBMCA}_k(1))$ and $\mathcal{L}(\text{1SL-NBMCA}_{k+1}(1))$ are incomparable. This is due to the fact that, for all $k \in \mathbb{N}$, all the languages over Σ in $\mathcal{L}(\text{1SL-NBMCA}_k(1))$ are also contained in $\mathcal{L}(\text{1SL-NBMCA}^{\Sigma}_k(1))$. Hence, we prove the theorem by showing that there exist infinitely many $k \in \mathbb{N}$ such that $\mathcal{L}(\text{1SL-NBMCA}^{\Sigma}_k(1))$ and $\mathcal{L}(\text{1SL-NBMCA}^{\Sigma}_{k+1}(1))$ are incomparable. For the sake of convenience, for every $k \in \mathbb{N}$, we define $\Gamma_k := \mathcal{L}(\text{1SL-NBMCA}^{\Sigma}_k(1))$. We note that it is sufficient to show $|\Gamma_k| \geq |\Gamma_{k+1}|$ in order to conclude that Γ_k and Γ_{k+1} are incomparable. This is due to the fact that by Theorems 4.28 and 4.30 there is a language L with $L \in \Gamma_{k+1}$ and $L \notin \Gamma_k$. Hence, $|\Gamma_k| \geq |\Gamma_{k+1}|$ implies the existence of a language L' with $L' \in \Gamma_k$ and $L' \notin \Gamma_{k+1}$.

Now let $k \in \mathbb{N}$ be arbitrarily chosen. We assume that for each k', $k \leq k' \leq k' + \widehat{m} - 1$, we have $|\Gamma_{k'}| < |\Gamma_{k'+1}|$. Since, for every k' with $k \leq k' \leq k + \widehat{m}$, $|\Gamma_{k'}| \leq \widehat{m}$, this is not possible. Hence, we conclude that there exists a k', $k \leq k' \leq k + \widehat{m} - 1$, such that $|\Gamma_{k'}| \geq |\Gamma_{k'+1}|$, which implies that $\Gamma_{k'}$ and $\Gamma_{k'+1}$ are incomparable. This concludes the proof. □

Theorem 4.33 illustrates that for special subclasses of NBMCA, namely the classes $\text{1SL-NBMCA}_k(1)$, $k \in \mathbb{N}$, the restricted nondeterminism cannot be controlled anymore in the usual way. Intuitively, this is caused by the lack of a finite state control. This result provides some insights on the question of how the existence of a finite state control affects the benefits of nondeterminism.

We shall now conclude this section by a brief summary of our results on stateless NBMCA. We have shown that NBMCA can be simulated by SL-NBMCA and that there exist infinitely many $k \in \mathbb{N}$ such that the classes $\mathcal{L}(\text{1SL-NBMCA}_k(1))$ and $\mathcal{L}(\text{1SL-NBMCA}_{k+1}(1))$ are incomparable. Especially the second result points out that by giving up the finite state control we also lose the possibility to control the nondeterminism, which, in our case, has led to a situation where we cannot prevent automata from using their nondeterminism to the full extent. Hence, by increasing the nondeterminism, the automata are forced to accept words they were not able to recognise before.

However, this result refers to a very restricted class of automata and it might be worthwhile to discuss possibilities to extend it. An obvious generalisation is to increase the number of counters to $m \in \mathbb{N}$ and use languages similar to $\mathbf{L}_k$ defined over an alphabet of $\{a_1, a_2, \ldots, a_m, \#_1, \#_2\}$, i. e., for every i, $1 \leq i \leq m$, all factors delimited by occurrences of $\#_1$ must have the same number of occurrences of a_i. Clearly, each counter i can then be used to count the occurrences of a_i. The difficulty with this approach is that SL-NBMCA(m) can use a part of their counters in order to simulate a finite state control. This suggests that generalising our results in this way might not be straightforward.

We can furthermore note that the result of Theorem 4.33 is non-constructive. The obvious approach in order to prove it constructively would be to show that no 1SL-NBMCA$_k(1)$ can recognise $\mathbf{L}_k$ (we recall that according to Theorem 4.28, $\mathbf{L}_k$ can be recognised by a 1SL-NBMCA$_{k-1}(1)$). The problem is that there is in fact a 1SL-NBMCA$_k(1)$ that recognises $\mathbf{L}_k$, namely the automaton $M_{\mathbf{L}_k}$ of Definition 4.27 with the only difference that the transitions $(¢, \mathtt{t}_0) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$ and $(¢, \mathtt{t}_1) \rightarrow_\delta (\mathtt{1}, \mathtt{0})$ are changed into $(¢, \mathtt{t}_0) \rightarrow_\delta (\mathtt{1}, \mathtt{r})$ and $(¢, \mathtt{t}_1) \rightarrow_\delta (\mathtt{1}, \mathtt{r})$. More precisely, we let the computations of $M_{\mathbf{L}_k}$ start with an reset that is simply not necessary. This suggests that a constructive proof of Theorem 4.33 might be more difficult than it appears at first glance.

4.2 Nondeterministically Initialised Multi-head Automata

In Section 3.1, it is shown that nondeterministic two-way multi-head automata can recognise pattern languages (Proposition 3.1). To this end, a nondeterministic two-way multi-head automaton uses its nondeterminism exclusively in order to initially move some of its input heads to nondeterministically chosen positions without paying attention to the input, and then a completely deterministic computation is performed. In the present section, we introduce and study a variant of two-way multi-head automata that is tailored to investigating this special kind of using nondeterminism. Since the variant to be introduced is an automaton with restricted nondeterminism, we also study a very fundamental aspect of automata theory, i. e., we compare the expressive power of nondeterministic two-way multi-head automata with the expressive power of deterministic two-way multi-head automata. We shall now discuss this aspect in a bit more detail.

Multi-head automata, in their one-way, two-way, deterministic and nondeterministic versions, have been intensely studied over the last decades (for a survey, see Holzer et al. [31]). They were first introduced by Rabin and Scott [62] and Rosenberg [74]. Although many results on multi-head automata have been reported since then, very basic questions still remain unsolved. One of these open problems is to determine whether or not, in the two-way case, nondeterminism is generally more powerful, i. e., whether or not the class of languages defined by two-way nondeterministic multi-head automata (2NFA) is strictly larger than the class of languages defined by two-way deterministic multi-head automata (2DFA). In other words, we ask whether we can remove the nondeterminism from an arbitrary 2NFA – compensated for, as appropriate, by enlarging its set of states and adding several input heads – without a detrimental effect on the computational power of the automaton. It is known that 2DFA and 2NFA characterise the complexity classes of deterministic logarithmic space (DL) and nondeterministic logarithmic space (NL), respectively (see, e. g., Sudborough [84]). Thus, the above described problem is equivalent to the DL-NL-Problem, i. e., the longstanding open question of whether or not DL and NL coincide. This problem has been further narrowed down by Hartmanis [30] and Sudborough [84], such that in fact DL = NL if and only if we can remove the nondeterminism from one-way nondeterministic two-head automata without changing the accepted language.

In order to gain further insights into the role of nondeterminism for a certain computation model, it is common to restrict the amount of nondeterminism (see, e. g., Fischer and Kintala [17] and Kintala [44]). With respect to multi-head automata, we can try to enlarge the set of languages defined by 2DFA by adding *some* amount of nondeterminism to the model of 2DFA and investigate the question whether or not this leads to a strictly more powerful device. If such a new model really is more powerful and, in terms of expressive power, still contained in the set of 2NFA, then the DL-NL-Problem is solved. If, on the other hand, we can show that our modification does not yield any advantages, then we have identified a special kind of nondeterminism that is not responsible for an increase of expressive power regarding 2DFA.

We follow this approach and introduce two-way deterministic multi-head automata, the input heads of which are nondeterministically initialised (IFA). More precisely, in every computation each input head is initially located at some nondeterministically chosen position in the input word; hence, the automaton, for each input head, guesses a position in the input word. Similarly, the first state is nondeterministically chosen from among a given set of possible initial states. After this initialisation, the automaton behaves like a normal 2DFA, i. e., every transition is deterministic. This model clearly is nondeterministic, but its nondeterminism is restricted. Although it is quite easy to see that IFA are not more powerful than classical 2NFA, it is not obvious whether a 2NFA that, for some constant $m \in \mathbb{N}$, performs at most m nondeterministic steps in every accepting computation (2NFA$_m$), can simulate the special nondeterminism of initialising the input heads. This is due to the fact that the natural way to move an input head to a nondeterministically chosen position of the input word is to move it to the right step by step, and, in *each* step, to guess whether it should be moved further on or stopped where it is. This procedure clearly requires a number of nondeterministic steps that depends on the guessed position of the input word and, thus, is not bounded by a constant. The question arises whether or not the model of IFA is more powerful than 2DFA and 2NFA_m. We answer this question in the negative by showing that the nondeterminism of 2NFA_m and IFA can be completely removed, i. e., they can be transformed into 2DFA, without increasing their number of input heads.

4.2.1 Automata With Restricted Nondeterminism

In this section, we define the automata models with restricted nondeterminism that are central for our investigations.

A *Nondeterministically Initialised Multi-head Automaton* (denoted by $\text{IFA}(k)$) is a $\text{DFA}(k)$ M that has a set of possible initial states, denoted by I. An $\text{IFA}(k)$ M accepts a word $w \in \Sigma^*$ if and only if $\widehat{c}_0 \vdash^*_{M,w} \widehat{c}_f$, where $\widehat{c}_f$ is some accepting configuration and $\widehat{c}_0$ is *any* configuration of form $(q, h_1, h_2, \ldots, h_k)$, where $q \in I$ and, for every i, $1 \leq i \leq k$, $0 \leq h_i \leq |w| + 1$.

For every $f : \mathbb{N} \to \mathbb{N}$, an $\text{NFA}(k)$ that makes at most $f(|w|)$ nondeterministic moves in every *accepting computation* on input ¢w\$ is said to have *restricted nondeterminism* and is denoted by $\text{NFA}_{f(n)}(k)$. If $f(n) = m$, for some constant $m \in \mathbb{N}$, then we write $\text{NFA}_m(k)$.

4.2.2 The Expressive Power of IFA(k) and NFA_m(k)

In this section, $\text{NFA}_{f(n)}(k)$, $\text{IFA}(k)$ and $\text{DFA}(k)$ are compared with respect to their expressive power. First, we note that by definition, for every $k \in \mathbb{N}$, $\mathcal{L}(\text{DFA}(k)) \subseteq \mathcal{L}(\text{IFA}(k)) \subseteq \mathcal{L}(\text{NFA}(k))$. This is due to the fact that, since the unrestricted nondeterminism of $\text{NFA}(k)$ can be used to nondeterministically initialise the input heads and to guess an initial state, an arbitrary $\text{IFA}(k)$ can be simulated by an $\text{NFA}(k)$ and, on the other hand, we can easily transform any $\text{DFA}(k)$ M into an equivalent $\text{IFA}(k)$ by aborting every computation that does not start with configuration $(q_0, 0, 0, \ldots, 0)$, i. e., the initial configuration of M.

As already stated at the beginning of Section 4.2, $\bigcup_k \mathcal{L}(\text{DFA}(k))$ coincides with DL, the class of languages that can be accepted by deterministic Turing machines working with $\text{O}(\log(n))$ space, where n is the length of the input. We can show that $\bigcup_k \mathcal{L}(\text{DFA}(k)) = \bigcup_k \mathcal{L}(\text{IFA}(k))$ and $\bigcup_k \mathcal{L}(\text{DFA}(k)) = \bigcup_{k,c} \mathcal{L}(\text{NFA}_{c\log(n)}(k))$ by showing how arbitrary $\text{IFA}(k)$ and $\text{NFA}_{c\log(n)}(k)$ can be simulated by deterministic Turing machines with $\text{O}(\log(n))$ space. We sketch these simulations very briefly. A deterministic Turing machine can simulate an $\text{IFA}(k)$ M_1 by enumerating all possible initial configurations of M_1 and, for each such configuration, it then simulates the *deterministic* computation of M_1 starting in this initial configuration. In order to investigate all possible initial configurations, M needs to keep track of the possible initial states of M_1 as well as of the input head positions. The numbers of input heads and possible initial states are constants, whereas each input head position can be stored within $\log(n)$ space.

In order to simulate an $\mathrm{NFA}_{c\log(n)}(k)$ M_2, the Turing machine M simply enumerates all possible binary strings $\alpha \in \{0,1\}^*$, $|\alpha| = c \times \log(n)$, and, for each such string α, it simulates M_2. If, in this simulation, M_2 performs the i^{th} nondeterministic step in its computation, then M chooses the next transition according to the i^{th} bit in α. This method can only be applied if the maximal nondeterministic branching factor of M_2 is 2, but it is straightforward to change it for the general case.

It follows that an arbitrary $\mathrm{IFA}(k)$ or $\mathrm{NFA}_{f(n)}(k)$ with $f(n) = \mathrm{O}(\log(n))$ can be transformed into a $\mathrm{DFA}(k')$. However, the details of such a transformation are not provided by the above sketched simulations and the question arises whether or not this can be done without increasing the number of input heads. In the following, we shall prove that, in fact, for every $k \in \mathbb{N}$, $\mathcal{L}(\mathrm{DFA}(k)) = \mathcal{L}(\mathrm{IFA}(k)) = \mathcal{L}(\mathrm{NFA}_{f(n)}(k))$, provided that $f(n)$ is a constant. Next, we show that, for every $k, m \in \mathbb{N}$, $\mathrm{NFA}_m(k)$ can be simulated by $\mathrm{IFA}(k)$.

Lemma 4.34. *Let $M \in \mathrm{NFA}_m(k)$, where $k, m \in \mathbb{N}$. There exists an $\mathrm{IFA}(k)$ M' such that $L(M) = L(M')$.*

Proof. There exists an $\widehat{m} \in \mathbb{N}$, such that we can transform M into an $\mathrm{NFA}_{\widehat{m}}(k)$ $\widehat{M} := (k, \widehat{Q}, \Sigma, \widehat{\delta}, q_0, \widehat{F})$ with $L(M) = L(\widehat{M})$ and, for every state $p \in \widehat{Q}$ and for all $b_1, b_2, \ldots, b_k \in \Sigma \cup \{¢, \$\}$, $|\widehat{\delta}(p, b_1, b_2, \ldots, b_k)| \leq 2$. This can be done by substituting a transition $\delta(p, b_1, b_2, \ldots, b_k)$ with $|\delta(p, b_1, b_2, \ldots, b_k)| = l > 2$, where δ is the transition function of M, by $l - 1$ transitions that have exactly two nondeterministic choices. Obviously, this requires $l-1$ new states. In the following we assume some order on the two options of a nondeterministic transition, such that we can write nondeterministic transitions as ordered tuples rather than as sets.

We shall now construct an $\mathrm{IFA}(k)$ M' with $L(M') = L(\widehat{M})$. Let $M' := (k, Q', \Sigma, \delta', I, F')$. Before we formally define M', we informally explain its behaviour. The automaton M' initially chooses one out of $2^{\widehat{m}}$ copies of the initial state q_0 of $\mathrm{NFA}_{\widehat{m}}(k)$ $\widehat{M}$. Each of these $2^{\widehat{m}}$ initial states of M' uniquely corresponds to $\widehat{m}$ nondeterministic binary guesses that may be performed in a computation of $\widehat{M}$. This is done by storing a binary sequence of length $\widehat{m}$ in the initial states of M'. After M' initially guesses one of the initial states, it simulates the computation of $\widehat{M}$. Deterministic steps are performed in exactly the same way and whenever $\widehat{M}$ nondeterministically chooses one out of two possible transitions, then M' chooses the next transition according to the first bit of the binary sequence currently stored in the state and this first bit is then removed. We shall give the formal definitions.

The set of states is defined by $Q' := \{q^{(\alpha)} \mid q \in \widehat{Q}, \alpha \in \{0,1\}^*, |\alpha| \leq \widehat{m}\}$, the set of initial states is defined by $I := \{q_0^{(\alpha)} \mid \alpha \in \{0,1\}^*, |\alpha| = \widehat{m}\}$ and the set of accepting states is defined by $F' := \{q^{(\alpha)} \mid q \in \widehat{F}, \alpha \in \{0,1\}^*, |\alpha| \leq \widehat{m}\}$. For every deterministic transition $\widehat{\delta}(p, b_1, b_2, \ldots, b_k) = \{(q, m_1, m_2, \ldots, m_k)\}$ of $\widehat{M}$ and for every $\alpha \in \{0,1\}^*$, $|\alpha| \leq \widehat{m}$, we define

$$\delta'(p^{(\alpha)}, b_1, b_2, \ldots, b_k) := (q^{(\alpha)}, m_1, m_2, \ldots, m_k)\,.$$

For each nondeterministic transition $\widehat{\delta}(p, b_1, b_2, \ldots, b_k) = ((q_1, m_{1,1}, m_{1,2}, \ldots, m_{1,k}), (q_2, m_{2,1}, m_{2,2}, \ldots, m_{2,k}))$ and for every $\alpha \in \{0,1\}^*$, $|\alpha| \leq \widehat{m} - 1$, we define

$$\begin{aligned}\delta'(p^{(0\cdot\alpha)}, b_1, b_2, \ldots, b_k) &:= (q_1^{(\alpha)}, m_{1,1}, m_{1,2}, \ldots, m_{1,k})\,,\\ \delta'(p^{(1\cdot\alpha)}, b_1, b_2, \ldots, b_k) &:= (q_2^{(\alpha)}, m_{2,1}, m_{2,2}, \ldots, m_{2,k})\,.\end{aligned}$$

This particularly means that if $|\widehat{\delta}(p, b_1, b_2, \ldots, b_k)| \geq 2$, then $\delta'(p^{(\varepsilon)}, b_1, b_2, \ldots, b_k)$ is undefined. Furthermore, in every initial state $q_0^{(\alpha)}$, M' must check whether all input heads scan the left endmarker and reject if this is not the case.

It can be easily verified that, for every input ¢w\$, we have $(q_0, 0, 0, \ldots, 0) \vdash^*_{\widehat{M},w} (q, h_1, h_2, \ldots, h_k)$, $q \in \widehat{F}$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$, by applying binary nondeterministic choices according to the sequence α, if and only if, for some $\alpha' \in \{0,1\}^*$ with $|\alpha \cdot \alpha'| = \widehat{m}$, $(q_0^{(\alpha\cdot\alpha')}, 0, 0, \ldots, 0) \vdash^*_{M',w} (q^{(\alpha')}, h_1, h_2, \ldots, h_k)$. Consequently, $L(\widehat{M}) = L(M')$. □

From Lemma 4.34, we can immediately conclude that, for every $k, m \in \mathbb{N}$, the class of languages described by $\text{NFA}_m(k)$ is included in the class of languages given by $\text{IFA}(k)$:

Theorem 4.35. *For every $k \in \mathbb{N}$ and $m \in \mathbb{N}$, $\mathcal{L}(\text{NFA}_m(k)) \subseteq \mathcal{L}(\text{IFA}(k))$.*

Before we can show our second result, i. e., $\text{IFA}(k)$ can be simulated by $\text{DFA}(k)$, we need to define a few more concepts. First, every $\text{IFA}(k)$ can be transformed into an equivalent one that has exactly one unique accepting configuration and it halts as soon as this configuration is entered:

Definition 4.36. Let $M \in \text{IFA}(k)$, $k \in \mathbb{N}$, and let F be the set of accepting states of M. M is *well-formed* if and only if $F = \{q_f\}$, $(q_f, 0, 0, \ldots, 0)$ is the only possible accepting configuration that can be reached in any computation of M and no transition $\delta(q_f, b_1, b_2, \ldots, b_k)$, $b_i \in \Sigma \cup \{¢, \$\}$, $1 \leq i \leq k$, is defined.

We observe that every IFA(k) can be transformed into an equivalent well-formed one by introducing a new state that serves as the only accepting state.

Proposition 4.37. *Let $M \in$ IFA(k), $k \in \mathbb{N}$. Then there exists a well-formed* IFA(k) *M' with $L(M) = L(M')$.*

The previous proposition shows that, when dealing with IFA(k), we can assume that they are well-formed. Next, we define a special configuration graph for computations of IFA(k), a concept that has already been introduced by Sipser in [81], where it has been applied to space-bounded Turing machines.

Definition 4.38. Let M be a well-formed IFA(k), $k \in \mathbb{N}$, and let $w \in \Sigma^*$. Let $G'_{M,w} := (V'_{M,w}, E'_{M,w})$, where $V'_{M,w} := \{(q, h_1, h_2, \ldots, h_k) \mid q \in Q, 0 \leq h_i \leq |w|+1, 1 \leq i \leq k\}$ and $E'_{M,w} := \{(c_1, c_2) \mid c_2 \vdash_{M,w} c_1\}$. The *backward configuration graph of M on w*, denoted by $G_{M,w}$, is the connected component of $G'_{M,w}$ that contains $(q_f, 0, 0, \ldots, 0)$.

Since the vertex $(q_f, 0, 0, \ldots, 0)$ of the backward configuration graph of a well-formed IFA(k) M cannot have an incoming edge and since all the transitions of M are deterministic, we can conclude that the backward configuration graph is a tree rooted by $(q_f, 0, 0, \ldots, 0)$. Therefore, from now on, we shall use the term *backward configuration tree.* For arbitrary $M \in$ IFA(k) and $w \in \Sigma^*$, the backward configuration tree can also be used to decide on the acceptance of w by M:

Proposition 4.39. *Let M be a well-formed* IFA(k)*, $k \in \mathbb{N}$, and let I be the set of initial states of M. For every $w \in \Sigma^*$, $w \in L(M)$ if and only if there exists a path from $(q_f, 0, 0, \ldots, 0)$ to some vertex $(q_0, h_1, h_2, \ldots, h_k)$, $q_0 \in I$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$, in the backward configuration tree of M on w.*

We can now state our next result, i. e., for every $k \in \mathbb{N}$, every IFA(k) can be transformed into an equivalent DFA(k). We shall prove this statement by applying a technique developed by Sipser in [81] in order to prove that every space-bounded deterministic Turing machine can be transformed into a halting deterministic Turing machine with the same space bound. Furthermore, this technique has also been used by Muscholl et al. [54] in order to show a similar result for deterministic tree-walking automata and by Geffert et al. [25] in order to complement deterministic two-way automata. More precisely, we show for an arbitrary IFA(k) M, how a DFA(k) M' can be constructed that, on any input ¢w\$, searches the backward configuration tree of M on w for a path from $(q_f, 0, 0, \ldots, 0)$ to some $(q_0, h_1, h_2, \ldots, h_k)$, where q_0 is an initial state of M. It is not obvious how M' can do this, since the size of the backward configuration tree of M on w does not only depend on the constant size of M, but also on the size of the current input ¢w\$.

Lemma 4.40. *Let $M \in \text{IFA}(k)$, $k \in \mathbb{N}$. There exists a* $\text{DFA}(k)$ *M', such that $L(M) = L(M')$.*

Proof. Let $\widehat{M} \in \text{IFA}(k)$, $k \in \mathbb{N}$, be arbitrarily chosen. By Proposition 4.37, we can conclude that there exists a well-formed $\text{IFA}(k)$ $M := (k, Q, \Sigma, \delta, I, \{q_f\})$ with $L(M) = L(\widehat{M})$. By Proposition 4.39, for every $w \in \Sigma^*$, we can decide on whether or not $w \in L(M)$ by searching the backward configuration tree of M on w for a path from $(q_f, 0, 0, \ldots, 0)$ to some vertex of form $(q_0, h_1, h_2, \ldots, h_k)$, $q_0 \in I$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$. Consequently, in order to prove the lemma, it is sufficient to show that this task can be carried out by a $\text{DFA}(k)$ M' if ¢w\$ is the input. More precisely, M' needs to perform a Depth-First-Search on the backward configuration tree of M on w starting at the root. Obviously, it is not possible to store the entire tree in the finite state control of M', as this tree grows with the input length. However, we shall see that it is possible for M' to construct the necessary parts of the tree "on-the-fly" without having to store too much information in the states. We shall explain the main idea in more detail.

For an arbitrary $w \in \Sigma^*$, let $(q, h_1, h_2, \ldots, h_k)$ be an arbitrary vertex of $G_{M,w}$. The situation that M' visits this vertex is represented in the following way: The input heads of M' scan the positions h_i, $1 \leq i \leq k$, of the input ¢w\$ and q, the state of M, is stored in the current state of M'. In order to avoid confusion, this state q shall be called the *currently stored state*. Initially, q_f is the the currently stored state, which, according to the above mentioned interpretation of how M' visits vertices of the backward configuration tree, particularly means that the initial configuration of M' corresponds to $(q_f, 0, 0, \ldots, 0)$, i. e., the root of the backward configuration tree. Now, M' has to visit the next vertex of $G_{M,w}$ according to a Depth-First-Search traversal. Let $(p, h'_1, h'_2, \ldots, h'_k)$ be this next vertex, so there is an edge $((q, h_1, h_2, \ldots, h_k), (p, h'_1, h'_2, \ldots, h'_k))$ in the backward configuration tree, which, by definition, implies $(p, h'_1, h'_2, \ldots, h'_k) \vdash_{M,w} (q, h_1, h_2, \ldots, h_k)$. Hence, in order to move from vertex $(q, h_1, h_2, \ldots, h_k)$ to vertex $(p, h'_1, h'_2, \ldots, h'_k)$, M' must simulate a step of M, but in the opposite direction.

The main difficulty with this procedure is that, for any vertex v in $G_{M,w}$, there may be several children to visit and, thus, we have to choose one of them and, furthermore, the next time we visit v we need to know which children have already been visited to decide which one to choose next. To this end we define a *rank* for all *possible* children of a vertex in $G_{M,w}$, and an order of these ranks. To implement the Depth-First-Search, M' then enumerates all *possible* children of the currently visited vertex v with respect to their rank and visits them. Now let u be the first child of v that is visited in this way. As soon as the subtree rooted by u has been completely searched, we move back to v and, in order to pick the next child of v to visit, we need to know the rank of u. Obviously, for every vertex, we cannot directly store the ranks of all its children visited so far, since these informations do not fit in the finite state control. However, by definition of the backward configuration tree, there is exactly one transition of M that changes M on input ¢w\$ from configuration u in configuration v, i. e., from the child to the parent. Therefore, we interpret this transition as the rank of u. This also allows us to restore the rank while moving from the child u back to the parent v without having to store it for the whole Depth-First-Search. Next, we shall formally define the set of ranks and then explain their role for the construction of M' in more detail:

$$\Gamma := \{\langle p, m_1, m_2, \ldots, m_k, q\rangle \mid p, q \in Q, m_i \in \{-1, 0, 1\}, 1 \leq i \leq k\}\,.$$

As mentioned above, a rank $\langle p, m_1, m_2, \ldots, m_k, q\rangle$ corresponds to a transition of M, i. e., the transition that changes M from state p to q and moves the input heads according to $m_1, m_2, \ldots, m_k$. Let $v := (q, h'_1, h'_2, \ldots, h'_k)$ and $u := (p, h_1, h_2, \ldots, h_k)$ be two arbitrarily chosen configurations of M on input ¢w\$. We say that u is an *actual child of v with rank $\langle p, m_1, m_2, \ldots, m_k, q\rangle$* if, for every i, $1 \leq i \leq k$, $m_i = h'_i - h_i$, and $\delta(p, w[h_1], w[h_2], \ldots, w[h_k]) = (q, m_1, m_2, \ldots, m_k)$. If, for every i, $1 \leq i \leq k$, $m_i = h'_i - h_i$, but $\delta(p, w[h_1], w[h_2], \ldots, w[h_k]) \neq (q, m_1, m_2, \ldots, m_k)$, then u is a *ghost child of v with rank $\langle p, m_1, m_2, \ldots, m_k, q\rangle$*. Obviously, u is an actual child of v if and only if u is also a child of v in the backward configuration tree of M on w, whereas ghost children do not exist in the backward configuration tree. However, it shall be very convenient to allow M' to visit ghost children and to interpret the backward configuration tree to contain ghost children as well. We also need an order over the set of ranks, but, as any such order is sufficient for our purpose, we simply assume that an order is given and we define a mapping next : $\Gamma \to \Gamma \cup \{0\}$, such that, for every $r \in \Gamma$ that is not the last rank in the order, next(r) is the successor of r and next$(r) = 0$ if r is the last rank. Now we are ready to formalise the constructions described above.

We assume that M' visits vertex $v := (p, h_1, h_2, \ldots, h_k)$ of the backward configuration tree right now, i. e., p is the currently stored state and the input heads scan positions $h_1, h_2, \ldots, h_k$ of the input ¢w\$. We distinguish two operational modes of M': Either M' just moved to v from its parent (*mode* 1) or it just moved back to v from one of its children (*mode* 2). In order to distinguish and change between these two different modes, M' uses an indicator implemented in the finite state control.

If M' is in mode 1, then it moved from the parent vertex $u := (q, h'_1, h'_2, \ldots, h'_k)$ to v. We assume that when this happens, the rank $r_v := \langle p, m_1, m_2, \ldots, m_k, q\rangle$ of v is already stored in the finite state control. By consulting the transition function δ of M, M' can check whether or not $\delta(p, w[h_1], w[h_2], \dots, w[h_k]) = (q, m_1, m_2, \dots, m_k)$, i. e., it checks whether or not v is an actual child or a ghost child. If v is a ghost child, then M goes back to u by changing the currently stored state back to q, moving the input heads according to $m_1, m_2, \dots, m_k$ and changing into mode 2. This is possible, since all necessary information for this step is provided by the rank r. If, on the other hand, v is an actual child, then M' stores the smallest possible rank $r_{\min}$ in the finite state control and visits the child of v with rank $r_{\min}$ while staying in mode 1.

If M' is in mode 2, then it has just been moved back to v from some child v' and we assume that the rank $r_{v'}$ of v' is stored in the finite state control. Now, if $\text{next}(r_{v'}) = 0$, then all children of v have been visited, thus, M' must go back to the parent vertex of v and stay in mode 2. Furthermore, this has to be done in a way that the rank of v is restored. Again, let $u := (q, h'_1, h'_2, \dots, h'_k)$ be the parent vertex of v. By definition, the rank of v is $r_v := \langle p, m_1, m_2, \dots, m_k, q\rangle$, where, for every i, $1 \leq i \leq k$, $m_i = h'_i - h_i$, and, since v is an actual child, $\delta(p, w[h_1], w[h_2], \dots, w[h_k]) = (q, m_1, m_2, \dots, m_k)$. Hence, all required information to restore the rank of v is provided by the transition function δ and the currently stored state p. So M' stores rank r_v in the finite state control and moves back to vertex v by changing the currently stored state to q and moving the input heads according to m_i, $1 \leq i \leq k$.

If, on the other hand, there exists a child of v that has not yet been visited and the rank of this child is $\text{next}(r_{v'}) = \langle q', m'_1, m'_2, \dots, m'_k, p\rangle$, then $\text{next}(r_{v'})$ is stored in the finite state control and M' visits the child corresponding to rank $\text{next}(r_{v'})$. This is done by changing the currently stored state from p to q' and moving the input heads exactly in the opposite direction as given by $m'_1, m'_2, \dots, m'_k$, i. e., for every i, $1 \leq i \leq k$, the instruction for head i is $(-m'_i)$. Furthermore, M' changes into mode 1.

In the procedure above, it can happen that the next child to visit has a rank that requires input heads to be moved to the left of the left endmarker or to the right of the right endmarker. By definition of an IFA(k), such a child can only be a ghost child, thus, we can simply ignore it and proceed with the next rank. As soon as a vertex of form $(q_0, h_1, h_2, \dots, h_k)$, $q_0 \in I$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$, is visited, M' accepts and if, in mode 2, M moves back to $(q_f, 0, 0, \dots, 0)$ from the child with the highest rank, then M' rejects w. This proves $L(M) = L(M')$. $\square$

From Lemma 4.40, we can immediately conclude the following theorem:

Theorem 4.41. *For every $k \in \mathbb{N}$, $\mathcal{L}(\text{IFA}(k)) \subseteq \mathcal{L}(\text{DFA}(k))$.*

From Theorems 4.35 and 4.41 we can now conclude that, for every $k, m \in \mathbb{N}$, $\mathcal{L}(\text{NFA}_m(k)) \subseteq \mathcal{L}(\text{IFA}(k)) \subseteq \mathcal{L}(\text{DFA}(k))$ and, by combining this result with the fact that, by definition, for every $k, m \in \mathbb{N}$, $\mathcal{L}(\text{DFA}(k)) \subseteq \mathcal{L}(\text{NFA}_m(k))$ trivially holds, we obtain the following corollary:

Corollary 4.42. *For every* $k, m \in \mathbb{N}$, $\mathcal{L}(\text{NFA}_m(k)) = \mathcal{L}(\text{IFA}(k)) = \mathcal{L}(\text{DFA}(k))$.

Thus, with reference to the questions addressed at the beginning of Section 4.2, we conclude that if nondeterminism yields an actual advantage, in terms of the expressive power of two-way multi-head automata, then this nondeterminism must be unrestricted. The proof of this insight is facilitated by the use of $\text{IFA}(k)$, which, in contrast to $\text{NFA}_m(k)$, provide the neat property of initially performing only one nondeterministic step followed by a completely deterministic computation.

4.2.3 Recognising Pattern Languages Deterministically

In this section, we show how pattern languages can be recognised by IFA. To this end, we recall that in the proof of Proposition 3.1, Section 3.1, page 26, we have already seen, for any pattern α, how a $\text{2NFA}(2\,|\operatorname{var}(\alpha)|+1)$ can recognise $L_{\text{Z},\Sigma}(\alpha)$, $\text{Z} \in \{\text{E}, \text{NE}\}$. Intuitively, this has been done by using $2\,|\operatorname{var}(\alpha)|$ input heads in order to implement $|\operatorname{var}(\alpha)|$ counters. These counters are initially incremented to nondeterministically chosen values, which are then interpreted as lengths of factors of the input word. Thus, a factorisation of the input word is guessed and it is then deterministically checked whether or not this factorisation satisfies the pattern α. Since in this procedure the only nondeterministic steps consist of initially incrementing the counters to nondeterministically chosen values, i. e., moving input heads to nondeterministically chosen positions in the input word, we can easily implement this procedure using an $\text{IFA}(2\,|\operatorname{var}(\alpha)| + 1)$. According to Corollary 4.42, this implies the statement of Proposition 3.3, a formal proof of which is omitted in Section 3.1. Furthermore, we can conclude the following corollary:

Corollary 4.43. *For every pattern* $\alpha \in (\Sigma \cup X)^*$ *and* $\text{Z} \in \{\text{E}, \text{NE}\}$, $L_{\text{Z},\Sigma}(\alpha) \in \text{DL}$.

In this regard, the results presented in Section 4.2 can be used in order to gain insights into the *space* complexity of pattern languages. However, we have to keep in mind that Corollary 4.43 is not a result about the space complexity of the membership problem for pattern languages, as defined in Section 2.2.2.1, since it only states that the membership problem for every *fixed* pattern language can be solved deterministically in logarithmic space (with respect to the length of the input word). In fact, it is very unlikely that the problem Z-PATMem, $\text{Z} \in \{\text{E}, \text{NE}\}$, is in NL or DL, since this implies that Z-PATMem is in P (cf. Sipser [82]); thus, the classes P and NP would coincide.

Although it is interesting to know that single pattern languages have deterministic logarithmic space complexity, we believe that complexity results on the problem Z-PATMem are more important. In the next section, we shall turn again towards this question.

Chapter 5

Second Approach: Relational Structures

In Chapter 3, a special kind of automaton model, the Janus automaton, is used in order to show that the variable distance is a crucial parameter with respect to the membership problem for pattern languages, i. e., it is shown that bounding the variable distance yields classes of patterns for which the membership problem can be solved efficiently. In this chapter, we approach the problem of identifying such parameters of patterns in a quite different and more general way. More precisely, in Section 5.1, we encode patterns and words as relational structures and, thus, reduce the membership problem to the homomorphism problem for relational structures. Our main result, a meta-theorem about the complexity of the membership problem, states that any parameter of patterns that is an upper bound for the treewidth of the corresponding relational structures, if restricted to a constant, allows the membership problem to be solved in polynomial time. In this new framework, we can restate the known results about the complexity of the membership problem mentioned in Section 2.2.2.1. Moreover, in Sections 5.2 and 5.3, we apply our meta-theorem in order to identify new and, compared to the old results, rather large classes of patterns with a polynomial time membership problem. Therefore, we provide a convenient way to study the membership problem for pattern languages, which, as shall be pointed out by our results, has still potential for further improvements.

5.1 A Meta-Theorem

The main result of this section can be informally stated in the following way. If a class of patterns can be encoded as special kinds of relational structures (to be defined in this section) in such a way that the treewidth of the corresponding encodings is bounded by a constant, then the membership problem with respect to this class of patterns can be solved in polynomial time.

5.1.1 Patterns and Words as Relational Structures

We now introduce a way of representing patterns and terminal words as relational structures. Our overall goal is to reduce the membership problem for pattern languages to the homomorphism problem for relational structures. For the remainder of this chapter, we define Σ to be some fixed terminal alphabet.

Representing words as relational structures is a common technique when mathematical logic is applied to language theory (see, e. g., Thomas [85] for a survey). However, our representations of patterns and words by structures substantially differ from the standard technique, since our approach is tailored to the homomorphism problem for structures and, furthermore, we want to exploit the treewidth.

In order to encode patterns and terminal words, i. e., an instance of the membership problem for pattern languages, we use the relational vocabulary $\tau_\Sigma := \{E, S, L, R\} \cup \{D_b \mid b \in \Sigma\}$, where E, S are binary relations and L, R, D_b, $b \in \Sigma$, are unary relations. The vocabulary depends on Σ, the alphabet under consideration. In order to represent a pattern α by a τ_Σ-structure, we interpret the set of positions of α as the universe. The roles of S, L, R and D_b, $b \in \Sigma$, are straightforward: S relates adjacent positions, L and R denote the leftmost and rightmost position, respectively, and, for every $b \in \Sigma$, the relation D_b contains the positions in α where the terminal symbol b occurs. For the encoding of the variables, we do not explicitly store their positions in the pattern, which seems impossible, since the number of different variables can be arbitrarily large and we can only use a finite number of relation symbols. Instead, we use the relation E in order to record pairs of positions where the same variable occurs and, furthermore, this is done in a "sparse" way. More precisely, the relation E relates *some* positions with the same variable, i. e., positions i, j with $\alpha[i] = \alpha[j]$, in such a way that the symmetric transitive closure of E contains *all* pairs (i, j) with $\alpha[i] = \alpha[j]$ and $\alpha[i] \in X$. This way of interpreting the relation E is crucial for our results.

We now state the formal definition and shall illustrate it afterwards.

Definition 5.1. Let α be a pattern and let $\mathcal{A}_\alpha$ be a τ_Σ-structure. $\mathcal{A}_\alpha$ is an *α-structure* if it has universe $A_\alpha := \{1, 2, \ldots, |\alpha|\}$ and $S^{\mathcal{A}_\alpha} := \{(i, i+1) \mid 1 \leq i \leq |\alpha| - 1\}$, $L^{\mathcal{A}_\alpha} := \{1\}$, $R^{\mathcal{A}_\alpha} := \{|\alpha|\}$, for every $b \in \Sigma$, $D_b^{\mathcal{A}_\alpha} := \{i \mid \alpha[i] = b\}$, and $E^{\mathcal{A}_\alpha}$ is such that, for all $i, j \in A_\alpha$,

- $(i, j) \in E^{\mathcal{A}_\alpha}$ implies $\alpha[i] = \alpha[j]$ and $i \neq j$,
- $\alpha[i] = \alpha[j]$ implies that (i, j) is in the symmetric transitive closure of $E^{\mathcal{A}_\alpha}$.

Since τ_Σ contains only unary and binary relation symbols, it is straightforward to derive the Gaifman graph from an α-structure, which is simply a graph with two different kinds of edges due to $S^{\mathcal{A}_\alpha}$ and $E^{\mathcal{A}_\alpha}$. Hence, in the following, we shall switch between these two models at our convenience without explicitly mentioning it. In the previous definition, the universe as well as the interpretations for the relation symbols S, L, R and D_b, $b \in \Sigma$, are uniquely defined for a fixed pattern α, while there are several possibilities of defining an interpretation of E. Intuitively, a valid interpretation of E is created by connecting different occurrences of the same variable by edges in such a way that all the occurrences of some variable describe a connected component. The simplest way of doing this is to add an edge between *any two* occurrences of the same variable, i. e., $E^{\mathcal{A}_\alpha} := \{(i,j) \mid \alpha[i] = \alpha[j]\}$. However, we shall see that for our results the interpretation of E is crucial and using the one just mentioned is not advisable. Another example of a valid interpretation of E is the following one. For every $x \in \operatorname{var}(\alpha)$, let l_x be the leftmost occurrence of x in α. Defining $E^{\mathcal{A}_\alpha} := \bigcup_{x \in \operatorname{var}(\alpha)} \{(l_x, i) \mid l_x < i \leq |\alpha|, \alpha[i] = x\}$ yields another possible α-structure.

Next, we define a canonical α-structure, i. e., the interpretation of E is such that every occurrence of a variable x at position i is connected to the next occurrence of x to the right of position i.

Definition 5.2. Let α be a pattern. The *standard α-structure (or $\mathcal{A}^s_\alpha$ for short)* is the α-structure where $E^{\mathcal{A}^s_\alpha} := \{(i,j) \mid 1 \leq i < j \leq |\alpha|, \exists\, x \in X \text{ such that } x = \alpha[i] = \alpha[j] \text{ and } \alpha[k] \neq x, i < k < j\}$.

As an example, we consider the standard α-structure $\mathcal{A}^s_\alpha$ for the pattern $\alpha := x_1\, \mathtt{a}\, \mathtt{b}\, x_1\, \mathtt{b}\, x_2\, \mathtt{a}\, x_1\, x_2\, x_1$. The universe of $\mathcal{A}^s_\alpha$ is $A_\alpha = \{1, 2, \ldots, 10\}$ and the relations are interpreted in the following way. $S^{\mathcal{A}^s_\alpha} = \{(1,2), (2,3), \ldots, (9,10)\}$, $L^{\mathcal{A}^s_\alpha} = \{1\}$, $R^{\mathcal{A}^s_\alpha} = \{10\}$, $D_{\mathtt{a}}^{\mathcal{A}^s_\alpha} = \{2,7\}$, $D_{\mathtt{b}}^{\mathcal{A}^s_\alpha} = \{3,5\}$ and, finally, $E^{\mathcal{A}^s_\alpha} = \{(1,4), (4,8), (6,9), (8,10)\}$.

We now introduce our representation of words over the terminal alphabet Σ as τ_Σ-structures. We recall that it is our goal to represent the membership problem for pattern languages as homomorphism problem for relational structures. Hence, the way we represent terminal words by τ_Σ-structures must cater for this purpose. Furthermore, we have to distinguish between the E case and the NE case. We first introduce the NE case and shall afterwards point out how to extend the constructions for the E case. We choose the universe to be the set of all possible factors of w, where these factors are represented by their unique start and end positions in w; thus, two factors that are equal but occur at different positions in w are different elements of the universe. The interpretation of L contains all prefixes and the interpretation of R contains all suffixes of w. The interpretation of S, which for patterns contains pairs of adjacent variables, contains now pairs of adjacent (non-overlapping) factors of w. The relation E is interpreted such that it contains *all* pairs of factors that are equal and non-overlapping. Finally, for every $b \in \Sigma$, D_b contains all factors of length one that equal b. This is necessary for the possible terminal symbols in the pattern.

For the E case, the empty factors of w need to be represented as well. To this end, for every i, $0 \leq i \leq |w|$, we add an element i_ε to the universe denoting the empty factor between positions i and $i+1$ in w. The interpretations of S and R are extended to also contain the empty prefix and the empty suffix, respectively, and relation S is extended to relate non-empty factors to adjacent empty factors and, in addition, each empty factor is also related to itself by S. Next, we formally define this construction for the NE case and its extension to the E case.

Definition 5.3. Let $w \in \Sigma^*$ be a terminal word. The NE-*w-structure ($\mathcal{A}_w$)* with universe A_w is defined by

- $A_w := \{(i,j) \mid 1 \leq i \leq j \leq |w|\}$,
- $E^{\mathcal{A}_w} := \{((i,j),(i',j')) \mid j < i' \text{ or } j' < i, w[i,j] = w[i',j']\}$,
- $S^{\mathcal{A}_w} := \{((i,j),(j+1,j')) \mid 1 \leq i \leq j, j+1 \leq j' \leq |w|\}$,
- $L^{\mathcal{A}_w} := \{(1,j) \mid 1 \leq j \leq |w|\}$,
- $R^{\mathcal{A}_w} := \{(i,|w|)| \mid 1 \leq i \leq |w|\}$ and,
- for every $b \in \Sigma$, $D_b^{\mathcal{A}_w} := \{(i,i) \mid w[i] = b\}$.

Let $\mathcal{A}_w$ be the NE-w-structure with universe A_w. We define the *E-w-structure ($\mathcal{A}_w^\varepsilon$)* with universe A_w^ε as follows:

- $A_w^\varepsilon := A_w \cup \{i_\varepsilon \mid 0 \leq i \leq |w|\}$,
- $E^{\mathcal{A}_w^\varepsilon} := E^{\mathcal{A}_w} \cup \{(i_\varepsilon, j_\varepsilon) \mid 0 \leq i \leq |w|, 0 \leq j \leq |w|\}$,
- $S^{\mathcal{A}_w^\varepsilon} := S^{\mathcal{A}_w} \cup \{(i_\varepsilon, i_\varepsilon) \mid 0 \leq i \leq |w|\} \cup$
  $\{((i,j), j_\varepsilon)) \mid 1 \leq i \leq j \leq |w|\} \cup \{(i_\varepsilon, (i+1,j)) \mid 0 \leq i \leq j \leq |w|\}$,
- $L^{\mathcal{A}_w^\varepsilon} := L^{\mathcal{A}_w} \cup \{0_\varepsilon\}$,
- $R^{\mathcal{A}_w^\varepsilon} := R^{\mathcal{A}_w} \cup \{|w|_\varepsilon\}$ and,
- for every $b \in \Sigma$, $D_b^{\mathcal{A}_w^\varepsilon} := D_b^{\mathcal{A}_w}$.

We illustrate the above definition with a brief example. To this end, let $w := \mathtt{abab}$. According to Definition 5.3, the universe of the NE-w-structure $\mathcal{A}_w$ is the set of all factors of w, given by their start and end positions in w, i. e.,

$$A_w = \{(1,1),(1,2),(1,3),(1,4),(2,2),(2,3),(2,4),(3,3),(3,4),(4,4)\}\,.$$

For every i, j, k, $1 \leq i \leq j < k \leq 4$, the elements (i, j) and $(j + 1, k)$ are in S relation under $\mathcal{A}_w$. Thus,

$$\begin{aligned} S^{\mathcal{A}_w} = \{&((1,1),(2,2)),((1,1),(2,3)),((1,1),(2,4)),((1,2),(3,3)),\\ &((1,2),(3,4)),((1,3),(4,4)),((2,2),(3,3)),((2,2),(3,4)),\\ &((2,3),(4,4)),((3,3),(4,4))\}\,. \end{aligned}$$

Every prefix of w is in L relation and every suffix of w is in R relation under $\mathcal{A}_w$. Hence, $L^{\mathcal{A}_w} = \{(1,1),(1,2),(1,3),(1,4)\}$ and $R^{\mathcal{A}_w} = \{(1,4),(2,4),(3,4),(4,4)\}$. Furthermore, $D_{\mathtt{a}}^{\mathcal{A}_w}$ and $D_{\mathtt{b}}^{\mathcal{A}_w}$ contain all factors that correspond to a single occurrence of $\mathtt{a}$ and $\mathtt{b}$, respectively, which implies $D_{\mathtt{a}}^{\mathcal{A}_w} = \{(1,1),(3,3)\}$, $D_{\mathtt{b}}^{\mathcal{A}_w} = \{(2,2),(4,4)\}$. Finally, two elements (i, j) and (i', j') are in E relation under $\mathcal{A}_w$ if and only if $w[i, j] = w[i', j']$; thus,

$$\begin{aligned} E^{\mathcal{A}_w} = \{&((1,1),(3,3)),((2,2),(4,4)),((1,2),(3,4)),\\ &((3,3),(1,1)),((4,4),(2,2)),((3,4),(1,2))\}\,. \end{aligned}$$

5.1.2 Reduction to the Homomorphism Problem for Relational Structures

In the following, we state that the membership problem for pattern languages can be reduced to the homomorphism problem for relational structures. We shall informally explain this for the case of NE-pattern languages given by patterns that do not contain any terminal symbols. Let α be a pattern without terminal symbols and let w be a terminal word, let $\mathcal{A}_\alpha$ be an α-structure and let $\mathcal{A}_w$ be the NE-w-structure. If there exists a substitution h that maps α to w, then we can construct a homomorphism g from $\mathcal{A}_\alpha$ to $\mathcal{A}_w$ by mapping the positions of α to the factors of w according to the substitution h. If two positions in α are adjacent, then so are their images under h in w and the same holds for equal variables in α; hence, g is a valid homomorphism. If, on the other hand, there exists a homomorphism g from $\mathcal{A}_\alpha$ to $\mathcal{A}_w$, then the elements of the universe of $\mathcal{A}_\alpha$, i. e., positions of α, are mapped to factors of w such that a factorisation of w is described. This is enforced by the relations S, L and R. Furthermore, this mapping from α to w induced by g is a substitution, since the symmetric transitive closure of $E^{\mathcal{A}_\alpha}$ contains all pairs (i, j) with $\alpha[i] = \alpha[j]$ and $\alpha[i] \in X$. For general patterns with terminal symbols and for the E case the idea is the same, but the situation is technically more complex.

Lemma 5.4. *Let α be a pattern, $w \in \Sigma^*$ and let $\mathcal{A}_\alpha$ be an α-structure. Then $w \in L_{\mathrm{NE},\Sigma}(\alpha)$ (or $w \in L_{\mathrm{E},\Sigma}(\alpha)$) if and only if there exists a homomorphism from $\mathcal{A}_\alpha$ to $\mathcal{A}_w$ (or from $\mathcal{A}_\alpha$ to $\mathcal{A}^\varepsilon_w$, respectively).*

Proof. We only prove the E-case, i. e., $w \in L_{\mathrm{E},\Sigma}(\alpha)$ if and only if there exists a homomorphism from $\mathcal{A}_\alpha$ to $\mathcal{A}^\varepsilon_w$. The proof for the NE-case is easier and can be done analogously. We start with the *if* direction. To this end, we assume that there exists a homomorphism $g : A_\alpha \to A^\varepsilon_w$ from $\mathcal{A}_\alpha$ to $\mathcal{A}^\varepsilon_w$, i. e., for every $p, q \in A_\alpha$,

- if $(p, q) \in E^{\mathcal{A}_\alpha}$, then $(g(p), g(q)) \in E^{\mathcal{A}^\varepsilon_w}$,
- if $(p, q) \in S^{\mathcal{A}_\alpha}$, then $(g(p), g(q)) \in S^{\mathcal{A}^\varepsilon_w}$,
- for every $b \in \Sigma$, if $p \in D_b^{\mathcal{A}_\alpha}$, then $g(p) \in D_b^{\mathcal{A}^\varepsilon_w}$
- $g(1) \in L^{\mathcal{A}^\varepsilon_w}$ and
- $g(|\alpha|) \in R^{\mathcal{A}^\varepsilon_w}$.

For the sake of convenience, we partition the universe of $\mathcal{A}^\varepsilon_w$ into $A^\varepsilon_w = A^{\neg\varepsilon}_w \cup A^\varepsilon_w$, where $A^{\neg\varepsilon}_w := \{(i, j) \mid 1 \leq i \leq j \leq |w|\}$ and $A^\varepsilon_w := \{i_\varepsilon \mid 0 \leq i \leq |w|\}$. For every $p \in A_\alpha$, if $g(p) = (s, t) \in A^{\neg\varepsilon}_w$, then we define $h(\alpha[p]) := w[s, t]$ and if, on the other hand, $g(p) \in A^\varepsilon_w$, then we define $h(\alpha[p]) := \varepsilon$. We can observe that if $\alpha[p]$ is a terminal $b \in \Sigma$, then $p \in D_b^{\mathcal{A}_\alpha}$. Thus, $g(p) \in D_b^{\mathcal{A}^\varepsilon_w}$, which implies $g(p) = (s, s)$ with $w[s] = b$ and, therefore, $h(b) = b$. For every $p, q \in A_\alpha$ with $\alpha[p] = \alpha[q]$ and $\alpha[p] \in X$, (p, q) is in the symmetric transitive closure of $E^{\mathcal{A}_\alpha}$. We note that, by definition, $E^{\mathcal{A}^\varepsilon_w}$ equals its symmetric transitive closure. Hence, we can conclude that $(g(p), g(q)) \in E^{\mathcal{A}^\varepsilon_w}$, which implies that $w[s, t] = w[s', t']$, where $(s, t) := g(p)$ and $(s', t') := g(q)$. Since $h(\alpha[p]) = w[s, t]$ and $h(\alpha[q]) = w[s', t']$, we may conclude $h(\alpha[p]) = h(\alpha[q])$. Consequently, h is a valid substitution and it remains to show $h(\alpha) = w$.

For every $p \in A_\alpha$, $p < |\alpha|$, $(p, p + 1) \in S^{\mathcal{A}_\alpha}$ and, thus, $(g(p), g(p + 1)) \in S^{\mathcal{A}^\varepsilon_w}$. By definition of $S^{\mathcal{A}^\varepsilon_w}$, this implies that either

1. $g(p) = (s, t) \in A^{\neg\varepsilon}_w$ and $g(p + 1) = (t + 1, t') \in A^{\neg\varepsilon}_w$,
2. $g(p) = s_\varepsilon \in A^\varepsilon_w$ and $g(p + 1) = (s + 1, t') \in A^{\neg\varepsilon}_w$,

3. $g(p) = (s, t) \in A_w^{\neg\varepsilon}$ and $g(p+1) = t_\varepsilon \in A_w^\varepsilon$ or

4. $g(p) = s_\varepsilon \in A_w^\varepsilon$ and $g(p+1) = s_\varepsilon \in A_w^\varepsilon$.

By the definition of h above, we can conclude that, for every $p, q \in A_\alpha$, $p < q$, with $g(p) = (s, t) \in A_w^{\neg\varepsilon}$ and $g(q) = (s', t') \in A_w^{\neg\varepsilon}$, $h(\alpha[p, q]) = w[s, t']$. Now let $l, r \in A_\alpha$ such that $g(l), g(r) \in A_w^{\neg\varepsilon}$ and, for every i with $1 \leq i < l$ and $r < i \leq |w|$, $g(i) \in A_w^\varepsilon$. This particularly means that $g(l) = (1, t)$ and $g(r) = (s', |w|)$. Since $1 \in L^{\mathcal{A}_\alpha}$ and $|\alpha| \in R^{\mathcal{A}_\alpha}$, we can conclude that $g(i) = 0_\varepsilon$, $1 \leq i < l$, and $g(i) = |w|_\varepsilon$, $r < i \leq |\alpha|$. Consequently, $h(\alpha[1, l-1]) = \varepsilon$, $h(\alpha[l, r]) = w$ and $h(\alpha[r+1, |\alpha|]) = \varepsilon$; hence, $h(\alpha) = w$.

For the *only if* direction, we assume that there exists a substitution h with $h(\alpha) = w$. We define a mapping $g : A_\alpha \to A_w^\varepsilon$ in the following way. For every $p \in A_\alpha$, if $h(\alpha[p]) \neq \varepsilon$, then we define $g(p) := (|h(\alpha[1, p-1])|+1, |h(\alpha[1, p])|) \in A_w^{\neg\varepsilon}$ and, if $h(\alpha[p]) = \varepsilon$, then we define $g(p) := |h(\alpha[1, p-1])|_\varepsilon \in A_w^\varepsilon$. It remains to show that g is a homomorphism from $\mathcal{A}_\alpha$ to $\mathcal{A}_w^\varepsilon$. For every $b \in \Sigma$, if $p \in D_b^{\mathcal{A}_\alpha}$, then $\alpha[p] = h(\alpha[p]) = b$; thus, $g(p) = (s, s)$, where $s := |h(\alpha[1, p])|$, and, since $h(\alpha) = w$, $w[s] = b$, which implies $g(p) \in D_b^{\mathcal{A}_w^\varepsilon}$. Obviously, either $g(1) = (1, |h(\alpha[1])|)$ or $g(1) = 0_\varepsilon$, and therefore $g(1) \in L^{\mathcal{A}_w^\varepsilon}$. Similarly, either $g(|\alpha|) = (|h(\alpha[1, |\alpha|-1])| + 1, |h(\alpha[1, |\alpha|])|)$ or $g(|\alpha|) = |\alpha|_\varepsilon$, which implies $g(|\alpha|) \in R^{\mathcal{A}_w^\varepsilon}$. For every $p, q \in A_\alpha$, if $(p, q) \in E^{\mathcal{A}_\alpha}$, then $\alpha[p] = \alpha[q]$ and, since $h(\alpha[p]) = h(\alpha[q])$, either $g(p) = (s, t)$ and $g(q) = (s', t')$ with $w[s, t] = w[s', t']$ or $g(p) = s_\varepsilon$ and $g(q) = s'_\varepsilon$. In both cases we can conclude that $(g(p), g(q)) \in E^{\mathcal{A}_w^\varepsilon}$. Let $p \in A_\alpha$, $p < |\alpha|$. We recall that $(p, p+1) \in S^{\mathcal{A}_\alpha}$ and observe four possible cases:

- If $g(p), g(p+1) \in A_w^{\neg\varepsilon}$, then $g(p) = (s, t)$ and $g(p+1) = (t+1, t')$.
- If $g(p) \in A_w^\varepsilon$ and $g(p+1) \in A_w^{\neg\varepsilon}$, then $g(p) = s_\varepsilon$ and $g(p+1) = (s+1, t')$.
- If $g(p) \in A_w^{\neg\varepsilon}$ and $g(p+1) \in A_w^\varepsilon$, then $g(p) = (s, t)$ and $g(p+1) = t_\varepsilon$.
- If $g(p), g(p+1) \in A_w^{\neg\varepsilon}$, then $g(p) = s_\varepsilon$ and $g(p+1) = s_\varepsilon$.

For all of these cases, $(g(p), g(p+1)) \in S^{\mathcal{A}_w^\varepsilon}$ is implied. This shows that g is an homomorphism from $\mathcal{A}_\alpha$ to $\mathcal{A}_w^\varepsilon$, which concludes the proof of the lemma. □

The above lemma shows that the membership problem for pattern languages is reducible to the homomorphism problem for relational structures and, thus, it can be solved by first transforming the pattern and the word into an α-structure and the NE-w-structure or E-w-structure and then deciding the homomorphism problem for these structures.

In the following, we say that a set of patterns P has bounded treewidth if and only if there exists a polynomial time computable mapping g that maps every

$\alpha \in P$ to an α-structure, such that $\{g(\alpha) \mid \alpha \in P\}$ has bounded treewidth. From Theorem 2.14 of Section 2.3.2 and Lemma 5.4 we can conclude the following result.

Corollary 5.5. *Let $P \subseteq (X \cup \Sigma)^+$ be a set of patterns with bounded treewidth. Then* NE-PATMem(P) *and* NE-PATMem(P) *are decidable in polynomial time.*

Proof. We assume that P has a bounded treewidth of $k \in \mathbb{N}$. Let $\alpha \in P$ and let $w \in \Sigma^*$. Obviously, w can be converted into the E-w-structure $\mathcal{A}^{\varepsilon}_w$ or into the NE-w-structure $\mathcal{A}_w$ in time $\mathrm{O}(|w|^4)$. Furthermore, by assumption, an α-structure $\mathcal{A}_\alpha$ that satisfies $\mathrm{tw}(\mathcal{A}_\alpha) \leq k$ can be computed in polynomial time. From Theorem 2.14, it follows that we can check whether or not there exists a homomorphism from $\mathcal{A}_\alpha$ to $\mathcal{A}^{\varepsilon}_w$ (or from $\mathcal{A}_\alpha$ to $\mathcal{A}_w$, respectively) in polynomial time. Now with Lemma 5.4, we can conclude the statement of the corollary. □

Due to Corollary 5.5, the task of identifying classes of patterns for which the membership problem is decidable in polynomial time can now be seen from a different angle, i. e., as the problem of finding classes of patterns with bounded treewidth. The fact that we can easily rephrase known results about the complexity of the membership problem for pattern languages (see Section 2.2.2.1) in terms of standard α-structures with a bounded treewidth, stated by the following proposition, indicates that this point of view is natural and fits with our current knowledge of the membership problem for pattern languages.

Proposition 5.6. *For every $k \in \mathbb{N}$, the sets of patterns $\{\alpha \mid \alpha \text{ is regular}\}$, $\{\alpha \mid \alpha \text{ is non-cross}\}$, $\{\alpha \mid |\mathrm{var}(\alpha)| \leq k\}$ and $\{\alpha \mid \mathrm{vd}(\alpha) \leq k\}$ have all bounded treewidth.*

Proof. If α is regular, then $\mathcal{A}^s_\alpha$ is a path and, thus, $\mathrm{tw}(\mathcal{A}^s_\alpha) = 1$. If α is non-cross, then it is straightforward to construct a path decomposition of $\mathcal{A}^s_\alpha$ with a width of at most 2. We can note that, By Lemma 2.5, $\mathrm{scd}(\alpha) \leq \mathrm{vd}(\alpha) + 1$ and, obviously, $\mathrm{vd}(\alpha) + 1 \leq |\mathrm{var}(\alpha)|$. In Section 5.2, Lemma 5.9, it shall be shown that $\mathrm{tw}(\mathcal{A}^s_\alpha) \leq \mathrm{scd}(\alpha) + 1$, which implies that $\{\alpha \mid |\mathrm{var}(\alpha)| \leq k\}$ and $\{\alpha \mid \mathrm{vd}(\alpha) \leq k\}$ have bounded treewidth. □

We conclude that our encodings of patterns and words as relational structures provide a convenient way to approach the membership problem for pattern languages and the hardness of the membership problem seems to be covered by the treewidth of the α-structures.

In the next section, we show that the numerical parameter of the scope coincidence degree of a pattern α is an upper bound for the treewidth of the standard α-structure; thus, restricting it yields classes of patterns with a polynomial time solvable membership problem. At the end of Section 5.2, we compare this result with the main result of Chapter 3, i. e., the membership problem for patterns with a bounded variable distance can be solved in polynomial time. Moreover, in Section 5.3, we identify a large class of patterns with a bounded treewidth of 2, but an unbounded scope coincidence degree.

5.2 Application I: The Scope Coincidence Degree

In order to show that, for every $k \in \mathbb{N}$, the set $\{\alpha \mid \mathrm{scd}(\alpha) \leq k\}$ has bounded treewidth we define, for any pattern α, a path decomposition of its standard α-structure.

Definition 5.7. Let α be a pattern and let $V := \{v_1, v_2, \ldots, v_{|\alpha|}\}$ be the set of vertices of the Gaifman graph of its standard α-structure, where, for every i, $1 \leq i \leq |\alpha|$, v_i corresponds to $\alpha[i]$. We inductively construct a sequence P_α of subsets of V in the following way.

1. Add $\{v_1\}$ to P_α, add $\{v_1, v_2\}$ to P_α, define $B := \{v_1, v_2\}$ and $i := 3$.
2. Define $B := B \cup \{v_i\}$ and, if $\alpha[i-2]$ is a terminal symbol or the rightmost occurrence of a variable in α, then define $B := B \setminus \{v_{i-2}\}$.
3. Add B to P_α.
4. If $\alpha[i] = x \in X$, but $\alpha[i]$ is not the leftmost occurrence of x, then define $B := B \setminus \{v_j\}$, where $j < i$, $\alpha[j] = x$ and, for every j', $j < j' < i$, $\alpha[j'] \neq x$.
5. Define $i := i + 1$ and if $i \leq |\alpha|$, then go to step 2.

Intuitively, the sequence $P_\alpha := (B_1, B_2, \ldots, B_k)$ is constructed in the following way. The first two sets are $\{v_1\}$ and $\{v_1, v_2\}$, respectively, and every following set is obtained from the previous one by adding the next vertex v_i and removing v_{i-2} if it corresponds to a terminal or the rightmost occurrence of a variable. Furthermore, if v_i corresponds to a variable that is not the leftmost occurrence of that variable, then the previous occurrence of this variable is still in our set and can now be removed. This ensures that for every edge $\{v_i, v_j\}$ of the Gaifman graph of the standard α-structure, there exists an l, $1 \leq l \leq k$, such that $\{v_i, v_j\} \subseteq B_l$. Furthermore, it can be easily verified that, for every vertex v of the Gaifman graph of the standard α-structure, there exist i, j, $1 \leq i < j \leq k$, such that $v \in \bigcap_{l=i}^{j} B_l$ and $v \notin ((\bigcup_{l=1}^{i-1} B_l) \cup (\bigcup_{l=j+1}^{k} B_l))$. Since, for every i, $1 \leq i \leq k$, exactly one element B_i is added to P_α in the construction of Definition 5.7, we can conclude that $k = |\alpha|$. We can further note that, for every i, $2 \leq i \leq k$, B_i contains exactly one new vertex that is not already contained in B_{i-1}, i. e., $|B_i \setminus B_{i-1}| = 1$. Next, we shall illustrate Definition 5.7 by a short example. Let $\beta := x_1 \mathtt{a} x_2 x_1 \mathtt{ab} x_2 x_3 x_3$. Then $P_\alpha = (\{v_1\}, \{v_1, v_2\}, \{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}, \{v_3, v_4, v_5\}, \{v_3, v_5, v_6\}, \{v_3, v_6, v_7\}, \{v_7, v_8\}, \{v_8, v_9\})$.

The above considerations imply the following:

Proposition 5.8. *Let α be a pattern. Then $P_\alpha := (B_1, B_2, \dots, B_k)$ is a path decomposition of the Gaifman graph of its standard α-structure. Moreover, $k = |\alpha|$ and, for every i, $2 \leq i \leq |\alpha|$, $|B_i \setminus B_{i-1}| = 1$.*

We call P_α the *standard path decomposition of* α and we shall now show that the width of the standard path decomposition is bounded by the scope coincidence degree of the corresponding pattern.

Lemma 5.9. *Let α be a pattern. Then the standard path decomposition of α has width at most* $\operatorname{scd}(\alpha) + 1$.

Proof. Let $P_\alpha := (B_1, B_2, \dots, B_{|\alpha|})$ be the standard path decomposition of α. We assume to the contrary that P_α has a width of at least $\operatorname{scd}(\alpha) + 2$, which implies that there exists a q, $1 \leq q \leq |\alpha|$, such that $|B_q| = m \geq \operatorname{scd}(\alpha) + 3$. Let $B_q := \{v_{i_1}, v_{i_2}, \dots, v_{i_m}\}$, where the vertices of B_q are in ascending order with respect to their indices. By definition of the standard path decomposition of α, for every j, $1 \leq j \leq m-2$, v_{i_j} corresponds to an occurrence of a distinct variable y_j in α. Furthermore, for every j, $1 \leq j \leq m-2$, there must exist an occurrence of y_j to the left and to the right of position q in α. This is due to the fact that if there is no occurrence of y_j to the left of q, then no vertex that corresponds to an occurrence of variable y_j is contained in B_q, and if there is no occurrence of y_j to the right of q, then vertex v_{i_j} would have been removed in step 2 of the procedure described in Definition 5.7. This directly implies that the scopes of variables $y_1, y_2, \dots, y_{m-2}$ coincide and, since $m \geq \operatorname{scd}(\alpha) + 3$, there are at least $\operatorname{scd}(\alpha) + 1$ variables in α, the scopes of which coincide, which is a contradiction. $\square$

By the previous lemma, we can conclude that, for every pattern α, the tree-width of the standard α-structure is bounded by the scope coincidence degree of α. Hence, for every $k \in \mathbb{N}$, the class of patterns $\{\alpha \mid \operatorname{scd}(\alpha) \leq k\}$ has bounded treewidth, and with Corollary 5.5 we can conclude the following:

Theorem 5.10. *Let* $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$ *and let* $k \in \mathbb{N}$. *The problem* Z-PATMem*(*$\{\alpha \mid \operatorname{scd}(\alpha) \leq k\}$*) is solvable in polynomial time.*

However, we are interested in a more detailed analysis of the time complexity of the membership problem for patterns with a bounded scope coincidence degree.

To this end, we give an algorithm that solves the homomorphism problem for the standard α-structure and a w-structure by using the standard path decomposition of α and analyse its time complexity. This algorithm follows the obvious way of using tree decompositions, which has already been briefly outlined at the end of Section 2.3.2. Hence, the main effort is to determine its runtime.

Theorem 5.11. *Let $k \in \mathbb{N}$ and $Z \in \{\mathrm{E}, \mathrm{NE}\}$. The problem* Z-PATMem*($\{\alpha \mid \mathrm{scd}(\alpha) \leq k\}$) is solvable in time* $\mathrm{O}(|\alpha| \times |w|^{2(k+3)} \times (k+2)^2)$.

Proof. We only show that NE-PATMem($\{\alpha \mid \mathrm{scd}(\alpha) \leq k\}$) is solvable in time $\mathrm{O}(|\alpha| \times |w|^{2(k+3)} \times (k+2)^2)$, since the E case can be dealt with analogously. Let (α, w) be an instance of NE-PATMem($\{\alpha \mid \mathrm{scd}(\alpha) \leq k\}$). We decide on whether or not $w \in L_{\mathrm{NE},\Sigma}(\alpha)$ by reduction to the homomorphism problem for relational structures. To this end, we first need to construct $\mathcal{A}^s_\alpha$ and $\mathcal{A}_w$, which can be done in time $\mathrm{O}(|w|^4 + |\alpha|)$. Let A_α and A_w be the universes of $\mathcal{A}^s_\alpha$ and $\mathcal{A}_w$, respectively, and let $P_\alpha := (B_1, B_2, \ldots, B_{|\alpha|})$ be the standard path decomposition of α. Before we give an algorithm deciding on whether or not there exists a homomorphism from $\mathcal{A}^s_\alpha$ to $\mathcal{A}_w$, we introduce some helpful notations.

Let h be a partial mapping from A_α to A_w. We say that h satisfies condition $(*)$ if and only if, for every $R \in \tau_\Sigma$ and for all $a_1, a_2, \ldots, a_{\mathrm{ar}(R)} \in A_\alpha$ for which h is defined, $(a_1, a_2, \ldots, a_{\mathrm{ar}(R)}) \in R^{\mathcal{A}^s_\alpha}$ implies $(h(a_1), h(a_2), \ldots, h(a_{\mathrm{ar}(R)})) \in R^{\mathcal{A}_w}$. Let $A := (a_1, a_2, \ldots, a_k)$ and $B := (b_1, b_2, \ldots, b_k)$ be arbitrary tuples of equal length. Then $A \mapsto B$ denotes the mapping that, for every $1 \leq i \leq k$, maps a_i to b_i. For any $C \subseteq A_\alpha$, $\mathrm{ord}(C)$ is a tuple containing the elements from C in increasing order (recall that $A_\alpha = \{1, 2, \ldots, |\alpha|\}$). Two partial mappings g and h from A_α to A_w are called *compatible* if and only if, for every $a \in A_\alpha$ for which both h and g are defined, $g(a) = h(a)$ is satisfied.

In the following, we shall describe an algorithm that decides on whether or not there exists a homomorphism from $\mathcal{A}^s_\alpha$ to $\mathcal{A}_w$. First, we compute a set H_1 of all tuples C of size $|B_1|$ containing elements from A_w such that the mapping $\mathrm{ord}(B_1) \mapsto C$ satisfies condition $(*)$. After that, for every i, $2 \leq i \leq |\alpha|$, we inductively compute a set H_i that is defined in the following way. For every tuple C of size $|B_i|$ containing elements from A_w, if the mapping $\mathrm{ord}(B_i) \mapsto C$ satisfies condition $(*)$ and the set H_{i-1} contains a tuple C' such that the mappings $\mathrm{ord}(B_i) \mapsto C$ and $B_{i-1} \mapsto C'$ are compatible, then we add C to H_i.

We now claim that there exists a homomorphism from $\mathcal{A}^s_\alpha$ to $\mathcal{A}_w$ if and only if $H_{|\alpha|}$ is nonempty. In order to prove this claim, we first assume that there exists a homomorphism from $\mathcal{A}^s_\alpha$ to $\mathcal{A}_w$. Now, for every i, $1 \leq i \leq |\alpha|$, let C_i be the tuple of elements from A_w, such that the mappings $\mathrm{ord}(B_i) \mapsto C_i$, $1 \leq i \leq |\alpha|$, if combined, form h. We note that this particularly implies that each two of the mappings $\text{ord}(B_i) \mapsto C_i$, $1 \leq i \leq |\alpha|$, are compatible. Since h is a homomorphism from $\mathcal{A}^s_\alpha$ to $\mathcal{A}_w$, for every i, $1 \leq i \leq |\alpha|$, the mapping $\text{ord}(B_i) \mapsto C_i$ satisfies condition $(*)$. This implies that $C_1 \in H_1$ holds and if, for some i, $1 \leq i \leq |\alpha| - 1$, $C_i \in H_i$ is satisfied, then, since the mappings $\text{ord}(B_i) \mapsto C_i$ and $\text{ord}(B_{i+1}) \mapsto C_{i+1}$ are compatible, $C_{i+1} \in H_{i+1}$ follows. By induction, this implies that $H_{|\alpha|}$ contains $C_{|\alpha|}$ and, thus, is nonempty.

Next, we assume that $H_{|\alpha|}$ is nonempty; thus, it contains some $C_{|\alpha|}$. By definition, this directly implies that, for every i, $1 \leq i \leq |\alpha| - 1$, H_i contains some element C_i and, without loss of generality, we can also conclude that, for every i, $1 \leq i \leq |\alpha| - 1$, the mappings $\text{ord}(B_i) \mapsto C_i$ and $\text{ord}(B_{i+1}) \mapsto C_{i+1}$ are compatible. Furthermore, since, for every $a \in A_\alpha$, there must exist at least one i, $1 \leq i \leq |\alpha|$, with $a \in B_i$ and, for all j, j', $1 \leq j < j' \leq |\alpha|$, $a \in (B_j \cap B_{j'})$ implies $a \in B_{j''}$, $j \leq j'' \leq j'$, we can conclude that each two of the mappings $\text{ord}(B_i) \mapsto C_i$, $1 \leq i \leq |\alpha|$, are compatible and for every $a \in A_\alpha$ at least one of the mappings $\text{ord}(B_i) \mapsto C_i$, $1 \leq i \leq |\alpha|$, is defined. This particularly implies that we can construct a total mapping h from A_α to A_w by combining all the mappings $\text{ord}(B_i) \mapsto C_i$, $1 \leq i \leq |\alpha|$. Now let $a_1, a_2, \ldots, a_{\text{ar}(R)}$ be arbitrary elements from A_α such that, for some $R \in \tau_\Sigma$, $(a_1, a_2, \ldots, a_{\text{ar}(R)}) \in R^{\mathcal{A}^s_\alpha}$. Since there must exist an i, $1 \leq i \leq |\alpha|$, with $a_1, a_2, \ldots, a_{\text{ar}(R)} \in B_i$ and since $C_i \in H_i$, i. e., $\text{ord}(B_i) \mapsto C_i$ satisfies condition $(*)$, we can conclude that $(h(a_1), h(a_2), \ldots, h(a_{\text{ar}(R)})) \in R^{\mathcal{A}_w}$, which implies that h is a homomorphism from $\mathcal{A}^s_\alpha$ to $\mathcal{A}_w$.

It remains to determine the runtime of the above algorithm. A central element of that algorithm is to check whether or not, for some i, $1 \leq i \leq |\alpha|$, and some tuple C of size $|B_i|$ containing elements from A_w, the mapping $\text{ord}(B_i) \mapsto C$ satisfies condition $(*)$. Since the arity of any relation symbol in τ_Σ is at most 2, this can be done in time $\text{O}(|B_i|^2)$. The set H_1 can be computed by simply considering every tuple C of elements from A_w of size $|B_1|$ and checking whether $\text{ord}(B_1) \mapsto C$ satisfies condition $(*)$. Thus, time $\text{O}(|B_1|^2 \times |A_w|^{|B_1|})$ is sufficient for computing H_1 and it remains to compute H_i, for every i, $2 \leq i \leq |\alpha|$. We recall that in order to compute such an H_i, we need to collect all tuples C of size $|B_i|$ containing elements from A_w such that the mapping $\text{ord}(B_i) \mapsto C$ satisfies condition $(*)$ and the set H_{i-1} contains a tuple C' such that the mappings $\text{ord}(B_i) \mapsto C$ and $\text{ord}(B_{i-1}) \mapsto C'$ are compatible. However, this can be done without having to enumerate all possible tuples C of size $|B_i|$ and then check for each such tuple whether or not H_{i-1} contains a tuple C' such that the mappings $\text{ord}(B_i) \mapsto C$ and $\text{ord}(B_{i-1}) \mapsto C'$ are compatible. This is due to the fact that, by Proposition 5.8, $|B_i \setminus B_{i-1}| = 1$, thus, all elements but one of the tuple C are already determined by the condition that there needs to be a $C' \in H_{i-1}$ such that the mappings $\text{ord}(B_i) \mapsto C$ and $\text{ord}(B_{i-1}) \mapsto C'$ are compatible. Consequently, there are at most $|A_w| \times |H_{i-1}|$ tuples that need to be checked for whether or not they satisfy condition ($*$). We conclude that the set H_i can be computed in time $\mathrm{O}(|A_w| \times |A_w|^{|B_{i-1}|} \times |B_i|^2) = \mathrm{O}(|A_w|^{|B_{i-1}|+1} \times |B_i|^2)$. Since, by Lemma 5.9, the width of the standard path decomposition is at most $k+1$, which implies $|B_i| \leq k+2$, for every i, $1 \leq i \leq |\alpha|$, we can conclude that the total runtime of the algorithm is $\mathrm{O}(|\alpha| \times |A_w|^{k+3} \times (k+2)^2) = \mathrm{O}(|\alpha| \times |w|^{2(k+3)} \times (k+2)^2)$. $\square$

The above result is similar to, but much stronger than the result that every class of patterns with a bounded variable distance has a polynomial time membership problem (see Chapter 3). This is due to the fact that if the variable distance is bounded by a constant, then this constitutes a much stronger restriction on the structure of a pattern than if the scope coincidence degree is restricted. Intuitively, this can be illustrated by the following scenario. For an arbitrary pattern $\alpha := \alpha_1 \cdot \alpha_2$, we insert a pattern β with $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta) = \emptyset$ into α, i. e., $\alpha' := \alpha_1 \cdot \beta \cdot \alpha_2$. Now, if $\mathrm{var}(\alpha_1) \cap \mathrm{var}(\alpha_2) \neq \emptyset$, then the variable distance of α' increases at least by $|\mathrm{var}(\beta)| - \mathrm{vd}(\alpha)$ compared to α regardless of the structure of β. This implies that it is rather difficult to enlarge a pattern by inserting new variables without increasing its variable distance. On the other hand, the scope coincidence degree of α' increases at least by $\mathrm{scd}(\beta) - \mathrm{scd}(\alpha)$ compared to α. This implies that the scope coincidence degree of α' depends on the structure of β or, more precisely, on the scope coincidence degree of β.

5.3 Application II: Mildly Entwined Patterns

In this section, we shall identify another structural property of patterns that allows the membership problem to be solved in polynomial time and that is incomparable to the variable distance and the scope coincidence degree. Next, we define this property.

Let α be a pattern. We say that two variables $x, y \in \mathrm{var}(\alpha)$ are *entwined* (in α) if and only if there exists a factorisation $\alpha = \beta \cdot x \cdot \gamma_1 \cdot y \cdot \gamma_2 \cdot x \cdot \gamma_3 \cdot y \cdot \delta$ or $\alpha = \beta \cdot y \cdot \gamma_1 \cdot x \cdot \gamma_2 \cdot y \cdot \gamma_3 \cdot x \cdot \delta$, where $\beta, \gamma_1, \gamma_2, \gamma_3, \delta \in (X \cup \Sigma)^*$. If no two variables in α are entwined, then α is a *nested* pattern. Intuitively, in a nested pattern, if a variable x occurs between two occurrences of another variable y, then *all* occurrences of x occur between these two occurrences of y. For example, $x_1\, x_3\, x_3\, x_4\, x_4\, x_1\, x_5\, x_5\, x_1\, x_2\, x_6\, x_7\, x_7\, x_6\, x_2$ is a nested pattern.

Next, we define a class of patterns that comprises entwined variables, but in a very restricted form.

Definition 5.12. Let α be a pattern. Two variables $x, y \in \mathrm{var}(\alpha)$, $x \neq y$, are *closely entwined* if they are entwined and, for every factorisation $\alpha = \beta \cdot x \cdot \gamma_1 \cdot y \cdot$

$\gamma_2 \cdot x \cdot \gamma_3 \cdot y \cdot \delta$ or $\alpha = \beta \cdot y \cdot \gamma_1 \cdot x \cdot \gamma_2 \cdot y \cdot \gamma_3 \cdot x \cdot \delta$, with $\beta, \gamma_1, \gamma_2, \gamma_3, \delta \in (X \cup \Sigma)^*$ and $|\gamma_2|_x = |\gamma_2|_y = 0$, $\gamma_2 = \varepsilon$ is implied. A pattern α is *closely entwined* if and only if all variables that are entwined are closely entwined.

In a closely entwined pattern, we allow variables to be entwined, but in the closest possible way, i. e., we require γ_2 to be empty. The following is an example for a closely entwined pattern: $\beta := x_1\, x_4\, x_1\, x_4\, x_5\, x_5\, x_4\, x_2\, x_1\, x_3\, x_2\, x_3\, x_2$. In β the variables x_1 and x_4, the variables x_1 and x_2 and the variables x_2 and x_3 are all pairs of variables that are entwined and, furthermore, they are all closely entwined. Obviously, the set of nested patterns is a proper subset of the class of closely entwined patterns. Next, we define a class of patterns that properly lies between the classes of nested patterns and closely entwined patterns.

Definition 5.13. A pattern α is *mildly entwined* if and only if it is closely entwined and, for every $x \in \text{var}(\alpha)$, if $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$ with $\beta, \gamma, \delta \in (X \cup \Sigma)^*$ and $|\gamma|_x = 0$, then γ is nested.

Intuitively, a mildly entwined pattern is by definition a closely entwined pattern with the additional condition that every factor that lies between two consecutive occurrences of a variable is a nested pattern. Obviously, there exist closely entwined patterns that are not mildly entwined (e. g., $x_1\, x_2\, x_3\, x_2\, x_3\, x_1$) and mildly entwined patterns that are not nested (e. g., $x_1\, x_2\, x_1\, x_2$). The following constitutes a more involved example for a mildly entwined pattern:

$$\gamma := x_1\, x_3\, x_4\, x_4\, x_3\, x_3\, x_1\, x_2\, x_3\, x_5\, x_5\, x_2\, x_5\, x_6\, x_6\, x_2\,.$$

First, we can note that the variables x_1 and x_3 are closely entwined, x_2 and x_3 are closely entwined, x_2 and x_5 are closely entwined and these are the only pairs of variables that are entwined. Furthermore, every factor between two consecutive occurrences of the same variable is nested. We emphasise that a factor γ between two consecutive occurrences of the same variable can still contain occurrences of a variable that is entwined with other variables in γ, as long as γ, considered individually, is nested. For example, the factor $x_3 x_4 x_4 x_3 x_3$ in between the first two occurrences of x_1 in γ contains variable x_3, which is entwined with variables x_1 and x_2.

Since we can decide in polynomial time on whether or not a given pattern is nested or closely entwined, we can also decide on whether or not a given pattern is mildly entwined in polynomial time.

We shall now show that the membership problem with respect to the class of mildly entwined patterns can be decided in polynomial time. To this end, we need to introduce a special class of graphs:

Definition 5.14. A graph is called *outerplanar* if and only if it can be drawn on the plane in such a way that no two edges cross each other and no vertex is entirely surrounded by edges (or, equivalently, all vertices lie on the exterior face).

For example, a cycle with 4 vertices is outerplanar, but the complete graph with 4 vertices, although planar, is not outerplanar.

Next, we show that a pattern is mildly entwined if and only if its standard α-structure is outerplanar.

Lemma 5.15. *Let α be a pattern. The Gaifman graph of the standard α-structure is outerplanar if and only if α is mildly entwined.*

Proof. Let $\mathcal{G}$ be the Gaifman graph of the standard α-structure and let $V := \{v_1, v_2, \ldots, v_{|\alpha|}\}$ be its set of vertices, where, for every i, $1 \leq i \leq |\alpha|$, v_i corresponds to $\alpha[i]$. We first show the *only if* direction by contraposition. To this end, we assume that α is not mildly entwined, which implies that α is either not closely entwined or there exists an $x \in \text{var}(\alpha)$ such that $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$ with $|\gamma|_x = 0$ and γ is not nested. If α is not closely entwined, then there are $x, y \in \text{var}(\alpha)$ such that $\alpha = \beta \cdot x \cdot \gamma_1 \cdot y \cdot \gamma_2 \cdot x \cdot \gamma_3 \cdot y \cdot \delta$ with $|\gamma_2|_x = |\gamma_2|_y = 0$ and $\gamma_2 \neq \varepsilon$. Furthermore, without loss of generality, we can assume that $|\gamma_1|_x = |\gamma_3|_y = 0$. Now let p_x, q_x, p_y, q_y be the positions of the occurrences of x and y shown by the above factorisation of α, i. e., $p_x = |\beta| + 1$, $p_y = p_x + |\gamma_1| + 1$, $q_x = p_y + |\gamma_2| + 1$ and $q_y = q_x + |\gamma_3| + 1$. We note that, since $|\gamma_1 \cdot \gamma_2|_x = 0$ and $|\gamma_2 \cdot \gamma_3|_y = 0$, there are edges $\{v_{p_x}, v_{q_x}\}$ and $\{v_{p_y}, v_{q_y}\}$ in $\mathcal{G}$ and, furthermore, there exists paths $(v_{p_x}, v_{p_x+1}, \ldots, v_{p_y})$ and $(v_{q_x}, v_{q_x+1}, \ldots, v_{q_y})$. This directly implies that, for every i, $p_y < i < q_x$, the vertex v_i is necessarily entirely surrounded by edges. Since $\alpha[p_y + 1, q_x - 1] = \gamma_2 \neq \varepsilon$, there exists at least one such vertex and, thus, $\mathcal{G}$ is not outerplanar.

If, on the other hand, there exists an $x \in \text{var}(\alpha)$ such that $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$ with $|\gamma|_x = 0$ and γ is not nested, then we can conclude that, for some $y, z \in \text{var}(\alpha)$, $\alpha = \beta \cdot x \cdot \gamma_1 \cdot y \cdot \gamma_2 \cdot z \cdot \gamma_3 \cdot y \cdot \gamma_4 \cdot z \cdot \gamma_5 \cdot x \cdot \delta$ and, without loss of generality, $|\gamma_2 \cdot \gamma_3|_y = |\gamma_3 \cdot \gamma_4|_z = 0$. Now let p_x, q_x, p_y, q_y, p_z and q_z be the positions of the occurrences of variables x, y and z, respectively, as highlighted by the above factorisation. We note that in $\mathcal{G}$ there are edges $\{v_{p_x}, v_{q_x}\}$, $\{v_{p_y}, v_{q_y}\}$ and $\{v_{p_z}, v_{q_z}\}$ and, in a similar way as above, this implies that vertex v_{p_z} or v_{q_y} is necessarily entirely surrounded by edges.

It remains to show that if α is mildly entwined, then $\mathcal{G}$ is outerplanar. To this end, we assume that α is mildly entwined and show how to draw a diagram of $\mathcal{G}$ on the plane that satisfies the following condition referred to as $(*)$: no two edges cross each other and no vertex is entirely surrounded by edges. First, we draw the path $(v_1, v_2, \ldots, v_{|\alpha|})$ in a straight line and note that the diagram of this path satisfies condition $(*)$. We shall now step by step add the remaining edges, which we call E-edges, since they are induced by the relation symbol E, and then show that in every step condition $(*)$ is maintained. In the following procedure, each of the E-edges will be drawn either above or below the path and we call a vertex v_i *covered above* or *covered below* (by an edge) if and only if we have already drawn an E-edge $\{v_j, v_{j'}\}$ with $j < i < j'$ above (or below, respectively) the path. We note that a vertex in the diagram is entirely surrounded by edges if and only if it is covered below and above at the same time. Next, we pass through the path from left to right, vertex by vertex. If for the current vertex v_p there does not exist an E-edge $\{v_p, v_q\}$ with $p < q$ (e. g., if v_p corresponds to a terminal symbol or to the rightmost occurrence of a variable), then we simply ignore this vertex and move on to the next one. If, one the other hand, such an E-edge exists, then we carry out one of the following steps.

1. If v_p is not covered above or below, then we draw the edge $\{v_p, v_q\}$ above the path.

2. If v_p is covered above or below by some edge and v_q is covered by the same edge, then we draw $\{v_p, v_q\}$ above the path (or below the path, respectively).

3. If v_p is covered above or below by some edge and v_q is not covered by this edge, then we draw $\{v_p, v_q\}$ below the path (or above the path, respectively).

It remains to show that each of the three steps above maintain condition $(*)$. If step 1 applies, then, since v_p is not covered by an edge, the subgraph with vertices $v_p, v_{p+1}, \ldots, v_{|\alpha|}$ is still a path and, thus, drawing $\{v_p, v_q\}$ above that path does not violate condition $(*)$. Now let us assume that step 2 applies and v_p is covered above by some edge $\{v_{p'}, v_{q'}\}$ with $p' < p < q < q'$. This implies that none of the vertices v_i, $p' < i < q'$, can be covered below by some edge, as otherwise they would be entirely surrounded by edges. So we can draw the edge $\{v_p, v_q\}$ above the path and still no vertex is entirely surrounded by edges. However, we have to show that we do not cross another edge by drawing $\{v_p, v_q\}$ in this way. To this end, we assume that there exists another edge $\{v_{\widehat{p}}, v_{\widehat{q}}\}$ that has already be drawn and that now crosses $\{v_p, v_q\}$ and we shall show that this assumption contradicts with the fact that α is mildly entwined. First, we can note that $\{v_{\widehat{p}}, v_{\widehat{q}}\}$ must be an E-edge that has been drawn above with either $p < \widehat{p} < q < \widehat{q}$ or $\widehat{p} < p < \widehat{q} < q$. We shall only consider the first of these two cases, since the second one can be handled analogously. Now, if $\widehat{q} < q'$, then $\alpha[p' + 1, q' - 1]$ is not nested, but, for some $x \in \text{var}(\alpha)$, $\alpha[p'] = \alpha[q'] = x$ and $|\alpha[p' + 1, q' - 1]|_x = 0$. This is a contradiction to the fact that α is mildly entwined. If, on the other hand, $q' < \widehat{q}$, then we can observe the following. Let $\alpha = \beta \cdot x \cdot \gamma_1 \cdot y \cdot \gamma_2 \cdot x \cdot \gamma_3 \cdot y \cdot \delta$ with $p' = |\beta| + 1$,

$\widehat{p} = |\beta \cdot x \cdot \gamma_1| + 1$, $q' = |\beta \cdot x \cdot \gamma_1 \cdot y \cdot \gamma_2| + 1$ and $\widehat{q} = |\beta \cdot x \cdot \gamma_1 \cdot y \cdot \gamma_2 \cdot x \cdot \gamma_3| + 1$. Since $\{v_{p'}, v_{q'}\}$ and $\{v_{\widehat{p}}, v_{\widehat{q}}\}$ are E-edges, we can conclude that $|\gamma_2|_x = |\gamma_2|_y = 0$, but, since $\widehat{p} < q < q'$, $\gamma_2 \neq \varepsilon$. This is a contradiction to the fact that α is closely entwined. Therefore, we can conclude that in fact $\{v_p, v_q\}$ does not cross an already existing edge and, thus, the diagram still satisfies condition $(*)$. If in step 2 vertex v_p is covered below instead of above, then an analogous argumentation can be used.

Finally, we assume that step 3 applies and v_p is covered above by some edge $\{v_{p'}, v_{q'}\}$ with $p' < p < q' < q$. We recall that $\alpha[p'] = \alpha[q']$ and $\alpha[p] = \alpha[q]$ and, since α is closely entwined, this implies that $p + 1 = q'$. Now we assume that no edge other than $\{v_{p'}, v_{q'}\}$ covers v_p, which particularly means that $v_{q'}$ is not covered by any edge. We conclude that we can draw the edge $\{v_p, v_q\}$ below the path without crossing an existing edge and since $p + 1 = q'$, i. e., there are no vertices between v_p and $v_{q'}$, no vertex is entirely surrounded by edges. It remains to show that there is in fact no other edge $\{v_{\widehat{p}}, v_{\widehat{q}}\}$ that covers v_p. To this end, we assume that there exists such an edge and note that this implies that one of the following 4 cases holds (recall that $p + 1 = q'$):

1. $\widehat{p} < p' < p < q' < q < \widehat{q}$,
2. $\widehat{p} < p' < p < q' < \widehat{q} < q$,
3. $p' < \widehat{p} < p < q' < q < \widehat{q}$,
4. $p' < \widehat{p} < p < q' < \widehat{q} < q$.

We can now show in a similar way as above, that cases 2 to 4 imply that α is not closely entwined and case 1 implies that there exists a variable $x \in \mathrm{var}(\alpha)$ such that $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$ with $|\gamma|_x = 0$ and γ is not nested. This contradicts our assumption that α is mildly entwined and, thus, we can conclude that in fact no edge other than $\{v_{p'}, v_{q'}\}$ covers v_p. If in step 3 vertex v_p is covered below instead of above, then an analogous argumentation can be used. This shows that the diagram drawn by the above procedure satisfies condition $(*)$, which proves that $\mathcal{G}$ is outerplanar. □

It is a well known fact that the class of outerplanar graphs has a bounded treewidth:

Theorem 5.16 (Bodlaender [8]). *If $\mathcal{G}$ is an outerplanar graph, then* $\mathrm{tw}(\mathcal{G}) \leq 2$.

Consequently, by Lemma 5.15 and Theorem 5.16, the class of mildly entwined patterns has bounded treewidth. Using Corollary 5.5, we can conclude that the membership problem with respect to mildly entwined patterns is decidable in polynomial time.

Theorem 5.17. *Let* $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$ *and let* P *be the class of mildly entwined patterns. The problem* Z-PATMem(P) *is solvable in polynomial time.*

Theorem 5.10 and the above Theorem 5.17, which both are applications of Corollary 5.5, constitute the two main results of this chapter. According to the definition of properties and parameters of patterns as given in Section 2.2.1, Theorem 5.10 shows that the membership problem can be solved efficiently if the parameter of the scope coincidence degree is bounded, and the statement of Theorem 5.17 is similar, but with respect to patterns that satisfy the property of being mildly entwined.

We shall now compare patterns with bounded scope coincidence degree and mildly entwined patterns. If a pattern has a scope coincidence degree of 1, then it is a non-cross pattern and, thus, it is also mildly entwined. The converse of this statement is not true, i. e., there are mildly entwined patterns with an arbitrarily large scope coincidence degree. This is illustrated by the pattern $\alpha := x_1 \cdot x_2 \cdot \dots \cdot x_k \cdot x_k \cdot x_{k-1} \cdot \dots \cdot x_1$, $k \in \mathbb{N}$. It can be easily verified that α is nested and, thus, also mildly entwined and, furthermore, $\mathrm{scd}(\alpha) = k$. Consequently, for every $k \geq 2$, the class of patterns with a scope coincidence degree of at most k and the class of mildly entwined patterns are incomparable, which shows that by our general approach, we have identified a parameter and a property of patterns that both contribute to the complexity of the membership problem, but in completely different ways.

We conclude this section by mentioning that the concept of outerplanarity of graphs can be generalised to k-outerplanarity in the following way. The 1-outerplanar graphs are exactly the outerplanar graphs and, for every $k \geq 2$, a graph is k-outerplanar if and only if it can be drawn on the plane in such a way that no two edges cross each other and, furthermore, if we remove all vertices on the exterior face and all their adjacent edges, then all remaining components are $(k-1)$-outerplanar. It can be shown that if a graph $\mathcal{G}$ is k-outerplanar, then $\mathrm{tw}(\mathcal{G}) \leq 3^k - 1$ (see Bodlaender [8] for further details on k-outerplanarity). Consequently, the property of being mildly entwined can be generalised to a parameter of patterns that corresponds to the k-outerplanarity of their standard α-structures. However, it is not straightforward to identify such a parameter of patterns, and therefore it is left to future research.

5.4 Ideas for Further Applications

In this chapter, we define a way of encoding patterns as relational structures, and we show that any parameter of patterns that is an upper bound for the treewidth of these encodings, if restricted, allows the membership problem for pattern languages to be solved in polynomial time. We then apply this meta-result in order to prove that all classes of patterns with a bounded scope coincidence degree and the class of mildly entwined patterns have a polynomial time membership problem.

In the definition of an α-structure (Definition 5.1), there are several different ways of how the relation symbol E can be interpreted. Thus, for a single pattern α, there are many possible α-structures that all permit an application of Theorem 5.5. However, the standard way of encoding patterns (Definition 5.2) turns out to be sufficient for all results in the present paper. It would be interesting to know whether or not, for some pattern α, there exists an α-structure $\mathcal{A}_\alpha$ that is better than the standard one, i. e., $\mathrm{tw}(\mathcal{A}_\alpha) < \mathrm{tw}(\mathcal{A}^s_\alpha)$. We conjecture that this question can be answered in the negative.

Section 5.3 constitutes an application of a more general technique that can be described in the following way. We consider an arbitrary class A of graphs with bounded treewidth and then we identify a class of patterns P and a polynomial time computable function g that maps the patterns of P to α-structures such that $\widehat{P} \subseteq A$, where $\widehat{P} := \{\mathcal{G}_\alpha \mid \alpha \in P, \mathcal{G}_\alpha$ is the Gaifman graph of $g(\alpha)\}$. Ideally, the class P can be characterised in terms of a parameter or a property of patterns that can be computed in polynomial time.

This indicates that, by applying the above described general technique, other classes of patterns with a polynomial time membership problem can be found, for example, by using the class of k-outerplanar graphs as the class of graphs with bounded treewidth, as outlined at the end of Section 5.3.

Chapter 6

Pattern Languages and the Chomsky Hierarchy

It is one of the many beneficial properties of regular and also context-free languages that their membership problem is comparatively easy to solve. Pattern languages, on the other hand, are context-sensitive languages and they are usually not regular or context-free; thus, their membership problem is hard to solve. This context-sensitivity of pattern languages is indicated by the fact that they can be interpreted as generalisations of the well known *copy language* $\{xx \mid x \in \Sigma^*\}$, which for $|\Sigma| \geq 2$ is a standard textbook example of a context-sensitive and non-context-free language. An exception, as mentioned in Chapter 2, are regular patterns, which do not contain variables with multiple occurrences, and therefore they describe regular languages. This particularly implies that their membership problem can be solved efficiently (cf. Shinohara [80]). While it is not difficult to show that regular patterns necessarily describe regular languages, we can observe that with respect to alphabets of size 2 and 3, pattern languages can be regular or context-free in an unexpected way, i. e., there are non-regular patterns, the pattern languages of which are nevertheless regular or context-free. For instance, the NE-pattern language of $\alpha := x_1 \, x_2 \, x_2 \, x_3$ is regular for $|\Sigma| = 2$, since squares are unavoidable for binary alphabets, which means that the language is co-finite. Surprisingly, for terminal alphabets of size 2 and 3, there are even certain E- and NE-pattern languages that are context-free but not regular. This recent insight is due to Jain et al. [40] and solves a longstanding open problem.

We wish to further investigate this existence of pattern languages that appear to be variants of the copy language, but are nevertheless regular or context-free. More precisely, we seek to identify criteria for patterns where the seemingly high complexity of a pattern does not translate into a high complexity of its pattern language (and, as an immediate result, the membership problem is also less complex than expected). Since, as demonstrated by Jain et al., this phenomenon does not occur for E-pattern languages if the pattern does not contain any terminal symbols or if the size of the terminal alphabet is at least 4, our investigations focus on patterns with terminal symbols and on small alphabets of sizes 2 or 3.

6.1 Definitions and Known Results

For the regularity of E-pattern languages, so-called block-regular patterns play an important role. Hence, we recapitulate these block-regular patterns as defined by Jain et al. [40]. Every factor of variables of α that is delimited by terminal symbols is called a variable block. More precisely, for every i, j, $1 \leq i \leq j \leq |\alpha|$, $\alpha[i, j]$ is a *variable block* if and only if $\alpha[k] \in X$, $i \leq k \leq j$, $\alpha[i-1] \in \Sigma$ or $i = 1$ and $\alpha[j+1] \in \Sigma$ or $j = |\alpha|$. A pattern α is *block-regular* if in every variable block of α there occurs at least one variable x with $|\alpha|_x = 1$. Let $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. The class of Z-pattern languages defined by regular patterns and block-regular patterns are denoted by Z-$\mathrm{PAT}_{\Sigma,\mathrm{reg}}$ and Z-$\mathrm{PAT}_{\Sigma,\mathrm{b\text{-}reg}}$, respectively. To avoid any confusion, we explicitly mention that the term regular pattern always refers to a pattern with the syntactical property of being a regular pattern and a regular E- or NE-pattern language is a pattern language that is regular, but that is not necessarily given by a regular pattern.

Known Characterisations

It can be easily shown that every E- or NE-pattern language over a unary alphabet is a regular language (cf. Reidenbach [63] for further details). Hence, the classes of regular and context-free pattern languages over a unary alphabet are trivially characterised. In Jain et al. [40] it has been shown that for any alphabet of cardinality at least 4, the regular and context-free E-pattern languages are characterised by the class of regular patterns.

Theorem 6.1 (Jain et al. [40]). *Let Σ be an alphabet with $|\Sigma| \geq 4$. Then* $(\text{E-PAT}_\Sigma \cap \mathrm{REG}) = (\text{E-PAT}_\Sigma \cap \mathrm{CF}) = \text{E-PAT}_{\Sigma,\mathrm{reg}}$.

Unfortunately, the above mentioned cases are the only complete characterisations of regular or context-free pattern languages that are known to date. In particular, characterisations of the regular and context-free E-pattern languages with respect to alphabets with cardinality 2 and 3, and characterisations of the regular and context-free NE-pattern languages with respect to alphabets with cardinality at least 2 are still missing. In the following, we shall briefly summarise the known results in this regard, and the reader is referred to Jain et al. [40] and Reidenbach [63] for further details. Jain et al. [40] present the example patterns

- $\alpha_1 := x_1\, x_2\, x_3\, \mathtt{a}\, x_2\, x_4\, x_4\, x_5\, \mathtt{a}\, x_6\, x_5\, x_7$ and
- $\alpha_2 := x_1\, x_2\, x_3\, \mathtt{a}\, x_2\, x_4\, x_4\, x_5\, \mathtt{b}\, x_6\, x_5\, x_7$,

and they show that $L_{\mathrm{E},\{\mathtt{a},\mathtt{b}\}}(\alpha_1)$ and $L_{\mathrm{E},\{\mathtt{a},\mathtt{b},\mathtt{c}\}}(\alpha_2)$ are regular languages that cannot be described by regular patterns. Moreover, in [40] it is shown that the patterns

- $\beta_1 := \mathtt{a}\, x_1\, \mathtt{a}\, x_2\, \mathtt{a}\, x_1\, \mathtt{a}\, x_3$ and
- $\beta_2 := x_1\, \mathtt{a}\, x_2\, \mathtt{b}\, x_3\, \mathtt{a}\, x_2\, \mathtt{b}\, x_4$

describe non-regular context-free E-pattern languages with respect to alphabet size 2 and 3, respectively. More precisely, $L_{\mathrm{E},\{\mathtt{a},\mathtt{b}\}}(\beta_1) \in \mathrm{CF} \setminus \mathrm{REG}$ and $L_{\mathrm{E},\{\mathtt{a},\mathtt{b},\mathtt{c}\}}(\beta_2) \in \mathrm{CF} \setminus \mathrm{REG}$. Regarding NE-pattern languages, it is shown that, for every alphabet Σ with cardinality at least 2, the class ($\text{NE-PAT}_\Sigma \cap \mathrm{REG}$) is not characterised by regular patterns, and with respect to alphabet sizes 2 and 3 it is not characterised by block-regular patterns either. Furthermore, for alphabet sizes 2 and 3, it is shown that the patterns β_1 and β_2 from above are also examples for non-regular context-free NE-pattern languages, i. e., $L_{\mathrm{NE},\{\mathtt{a},\mathtt{b}\}}(\beta_1) \in \mathrm{CF} \setminus \mathrm{REG}$ and $L_{\mathrm{NE},\{\mathtt{a},\mathtt{b},\mathtt{c}\}}(\beta_2) \in \mathrm{CF} \setminus \mathrm{REG}$. For alphabets with cardinality of at least 4 the existence of such patterns is still open.

6.2 Regularity and Context-Freeness of Pattern Languages: Sufficient Conditions and Necessary Conditions

Since their introduction by Shinohara [80], it has been known that, for both the E and NE case and for any terminal alphabet, regular patterns can only describe regular languages. This is an immediate consequence of the fact that regular patterns do not use the essential mechanism of patterns, i. e., repeating variables in order to define sets of words that contain repeated occurrences of variable factors. In Jain et al. [40], the concept of regular patterns is extended to block-regular patterns, defined in Section 6.1. By definition, every regular pattern is a block-regular pattern. Furthermore, in the E case, every block-regular pattern α is equivalent to the regular pattern obtained from α by substituting every variable block by a single occurrence of a variable.

Proposition 6.2. *Let Σ be some terminal alphabet and let $\alpha \in (\Sigma \cup X)^*$ be a pattern. If α is regular, then $L_{\mathrm{NE},\Sigma}(\alpha) \in \mathrm{REG}$. If α is block-regular, then $L_{\mathrm{E},\Sigma}(\alpha) \in \mathrm{REG}$.*

As mentioned in Section 6.1, for alphabets of size at least 4, both the class of regular patterns and the class of block-regular patterns characterise the set of regular and context-free E-pattern languages. However, in the NE case as well as in the E case with respect to alphabets of size 2 or 3, Jain et al. [40] demonstrate that block-regular patterns do not characterise the set of regular or context-free pattern languages.

Obviously, the regularity of languages given by regular patterns or block-regular patterns follows from the fact that there are variables that occur only once in the pattern. Hence, it is the next logical step to ask whether or not the existence of variables with only one occurrence is also necessary for the regularity or the context-freeness of a pattern language. Jain et al. [40] answer that question with respect to terminal-free patterns or, more precisely, an answer directly follows from the results provided in [40]. So the following Theorem 6.3 is due to Jain et al., but we feel that it is appropriate to point out how exactly it can be concluded from the results provided in [40], which is done in a separate proof for Theorem 6.3.

Theorem 6.3 (Jain et al. [40])**.** *Let Σ be a terminal alphabet with $|\Sigma| \geq 2$ and let α be a terminal-free pattern with $|\alpha|_x \geq 2$, for every $x \in \mathrm{var}(\alpha)$. Then $L_{\mathrm{E},\Sigma}(\alpha) \notin \mathrm{CF}$ and $L_{\mathrm{NE},\Sigma}(\alpha) \notin \mathrm{REG}$.*

Proof. Let Σ' be an alphabet with $|\Sigma'| = 2$ and $\Sigma' \subseteq \Sigma$. By Lemma 11 of [40], it follows that $L_{\mathrm{E},\Sigma'}(\alpha) \notin \mathrm{CF}$. Since $L_{\mathrm{E},\Sigma}(\alpha) \cap \Sigma'^*$ equals $L_{\mathrm{E},\Sigma'}(\alpha)$ and since the class of context-free languages is closed under intersection with regular sets, we can conclude that $L_{\mathrm{E},\Sigma}(\alpha) \notin \mathrm{CF}$.

In order to show $L_{\mathrm{NE},\Sigma}(\alpha) \notin \mathrm{REG}$, we can apply the proof of Theorem 6.*a* of [40], which states that for any terminal alphabet Σ' with $|\Sigma'| \geq 4$ and for any pattern β that is not block-regular, $L_{\mathrm{NE},\Sigma'}(\beta)$ is not a regular language. However, for terminal-free patterns in which every variable occurs at least twice this proof also works for an alphabet of size 2 and 3, since we do not need the two terminal symbols to both sides of the variable block (cf. [40] for details). □

We can note that Proposition 6.2 and Theorem 6.3 characterise the regular and context-free E-pattern languages given by terminal-free patterns with respect to alphabets of size at least 2. More precisely, for every alphabet Σ with $|\Sigma| \geq 2$ and for every terminal-free pattern α, if α is block-regular, then $L_{\mathrm{E},\Sigma}(\alpha)$ is regular (and, thus, also context-free) and if α is not block-regular, then every variable of α occurs at least twice, which implies that $L_{\mathrm{E},\Sigma}(\alpha)$ is neither regular nor context-free.

However, for the NE case, we cannot hope for such a simple characterisation. This is due to the close relationship between the regularity of NE-pattern languages and the combinatorial phenomenon of unavoidable patterns, as already mentioned at the beginning of this chapter.

In the following, we concentrate on E-pattern languages over alphabets of size 2 and 3 (since for all other alphabet sizes complete characterisations are known) that are given by patterns that are *not* terminal-free (since, as described above, the characterisation of regular and context-free E-pattern languages given by terminal-free patterns has been settled). Nevertheless, some of our results also hold for NE-pattern languages and we shall always explicitly mention if this is the case.

The next two results present a sufficient condition for the non-regularity and a sufficient condition for the non-context-freeness of pattern languages over small alphabets. More precisely, we generalise Theorem 6.3 to patterns that are not necessarily terminal-free. The first result states that for a pattern α (that may contain terminal symbols), if every variable in α occurs at least twice, then both the E- and NE-pattern language of α, with respect to alphabets of size at least two, is not regular.

Theorem 6.4. *Let Σ be an alphabet with $|\Sigma| \geq 2$, let $\alpha \in (\Sigma \cup X)^*$ be a pattern, and let $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. If, for every $x \in \mathrm{var}(\alpha)$, $|\alpha|_x \geq 2$, then $L_{\mathrm{Z},\Sigma}(\alpha) \notin \mathrm{REG}$.*

Proof. We only prove that $L_{\mathrm{NE},\Sigma}(\alpha) \notin \mathrm{REG}$ since $L_{\mathrm{E},\Sigma}(\alpha) \notin \mathrm{REG}$ can be shown in exactly the same way. To this end, we assume to the contrary that $L_{\mathrm{NE},\Sigma}(\alpha) \in \mathrm{REG}$ and we let n be the constant from Lemma 2.2 (see page 9) with respect to $L_{\mathrm{NE},\Sigma}(\alpha)$. Furthermore, we assume that $\alpha := u_0 \cdot y_1 \cdot u_1 \cdot y_2 \cdot u_2 \cdot \cdots \cdot u_{k-1} \cdot y_k \cdot u_k$, where $y_i \in X$, $1 \leq i \leq k$, and $u_i \in \Sigma^*$, $0 \leq i \leq k$. Now, we let w be the word obtained from α by substituting every variable by the word $\mathtt{ba}^n\mathtt{b}^n\mathtt{a}$, i. e.,

$$w = u_0 \cdot \mathtt{ba}^n\mathtt{b}^n\mathtt{a} \cdot u_1 \cdot \mathtt{ba}^n\mathtt{b}^n\mathtt{a} \cdot u_2 \cdot \cdots \cdot u_{k-1} \cdot \mathtt{ba}^n\mathtt{b}^n\mathtt{a} \cdot u_k\,.$$

By first applying Lemma 2.2 on the factor $\mathtt{ba}^n\mathtt{b}^n\mathtt{a}$ that results from y_1, then on the factor $\mathtt{ba}^n\mathtt{b}^n\mathtt{a}$ that results from y_2 and so on, we can obtain the word

$$w' := u_0 \cdot \mathtt{ba}^{n_1}\mathtt{b}^{n_2}\mathtt{a} \cdot u_1 \cdot \mathtt{ba}^{n_3}\mathtt{b}^{n_4}\mathtt{a} \cdot u_2 \cdot \cdots \cdot u_{k-1} \cdot \mathtt{ba}^{n_{2k-1}}\mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k\,,$$

where $n \times |\alpha| < n_1$, and, for every i, $1 \leq i \leq 2k-1$, $n_i \times |\alpha| < n_{i+1}$. We shall now show that $w' \notin L_{\mathrm{NE},\Sigma}(\alpha)$. To this end, we assume to the contrary that there exists a substitution h with $h(\alpha) = w'$. Let p, $1 \leq p \leq |\alpha|$, be such that $\alpha[p,-]$ is the shortest suffix of α such that $\mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$ is a suffix of $h(\alpha[p,-])$. If $h(\alpha[p,-]) = v \cdot \mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$, $v \neq \varepsilon$, then $\alpha[p]$ must be a variable, since otherwise $\mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$ is also a suffix of $h(\alpha[p+1,-])$ which implies that $\alpha[p,-]$ is not the shortest suffix of α such that $\mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$ is a suffix of $h(\alpha[p,-])$. Moreover, for similar reasons, we can conclude that $h(\alpha[p]) = v \cdot v'$, where v' is a non-empty prefix of $\mathtt{b}^{n_{2k}}$. Now if $h(\alpha[p])$ contains the whole factor $\mathtt{a}^{n_{2k-1}}$, then, since $\alpha[p]$ is a repeated variable in α, there are two non-overlapping occurrences of factor $\mathtt{a}^{n_{2k-1}}$ in $h(\alpha)$, which is a contradiction, since there are no two non-overlapping occurrences of factor $\mathtt{a}^{n_{2k-1}}$ in w'. So we can conclude that either $h(\alpha[p,-]) = \mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$ or $h(\alpha[p,-]) = \mathtt{a}^m \cdot \mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$ and $\alpha[p]$ is a variable with $h(\alpha[p]) = \mathtt{a}^m \cdot \mathtt{b}^l$, $1 \leq m < n_{2k-1}$, $l \neq 0$.

There must exist at least one variable $x \in \text{var}(\alpha)$ with $|h(x)| > n_{2k-1}$, since otherwise $|h(\alpha)| \leq |\alpha| \times n_{2k-1} < n_{2k} < |w'|$, which is a contradiction. Now let $z \in \text{var}(\alpha)$ be such a variable, i. e., $|h(z)| > n_{2k-1}$. We recall that $h(\alpha[1, p-1]) := u_0 \cdot \mathtt{b}\mathtt{a}^{n_1}\mathtt{b}^{n_2}\mathtt{a} \cdot u_1 \cdot \cdots \cdot u_{k-1} \cdot \mathtt{b}\mathtt{a}^{n_{2k-1}-m}$. If $z \in \text{var}(\alpha[1, p-1])$, then there are to cases to consider. If, for some i, $1 \leq i \leq k-1$, $h(z)$ contains a factor $\mathtt{a}\mathtt{b}^{n_{2i}}\mathtt{a}$ or a factor $\mathtt{b}\mathtt{a}^{n_{2i-1}}\mathtt{b}$, then we obtain a contradiction, since in w' there is exactly one occurrence of such a factor, but there are at least two occurrences of variable z in α. If, on the other hand, $h(z)$ contains no such factor, then $h(z)$ is a factor of the suffix $\mathtt{b}^{n_{2k-2}}\mathtt{a} \cdot u_{k-1} \cdot \mathtt{b}\mathtt{a}^{n_{2k-1}-m}$ of $h(\alpha[1, p-1])$. Since $|h(z)| > n_{2k-1}$, this implies that $h(z)$ must have a suffix $\mathtt{a}^q$, where $q > n_{2k-1} - (n_{2k-2} + |u_{k-1}| + 2)$. We observe that

$$\begin{aligned} &n_{2k-1} - (n_{2k-2} + |u_{k-1}| + 2) > n_{2k-1} - (3 \times n_{2k-2}) > \\ &|\alpha| \times n_{2k-2} - (3 \times n_{2k-2}) = (|\alpha| - 3) \times n_{2k-2}\,. \end{aligned}$$

Now, we can conclude that since $(|\alpha|-3) \times n_{2k-2} > (|\alpha|-3) \times |\alpha| \times n_{2k-3} > n_{2k-3}$, $q > n_{2k-3}$. This directly implies that in $h(\alpha[1, p-1])$ there does not exist another occurrence of factor $\mathtt{a}^q$ and, thus, there is exactly one occurrence of variable z in $\alpha[1, p-1]$, which implies that there must be another occurrence of variable z in $\alpha[p,-]$. This particularly means that there is an occurrence of $h(z)$ in $h(\alpha[p,-]) = \mathtt{a}^m \cdot \mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$. We recall that $h(z)$ contains $\mathtt{a}^q$ as a suffix, which implies that in $h(\alpha[p,-])$, $h(z)$ cannot end in $\mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$, since this means that the whole suffix $\mathtt{a}^q$ is contained in $\mathtt{b}^{n_{2k}}\mathtt{a} \cdot u_k$. So $h(z)$ must entirely be contained in $\mathtt{a}^m$, which is a contradiction, since $|h(z)| > n_{2k-1}$ and $m < n_{2k-1}$.

This proves that the word w' is not in $L_{\text{NE},\Sigma}(\alpha)$, which, by Lemma 2.2, implies $L_{\text{NE},\Sigma}(\alpha) \notin \text{REG}$. □

For alphabets of size at least 3 Theorem 6.4 can be strengthened, i. e., if every variable in a pattern α occurs at least twice, then the E- and the NE-pattern language of α are not context-free. This result is due to Reidenbach [73].

Theorem 6.5 (Reidenbach [73]). *Let Σ be an alphabet with $|\Sigma| \geq 3$, let $\alpha \in (\Sigma \cup X)^+$ be a pattern, and let $Z \in \{\text{E}, \text{NE}\}$. If, for every $x \in \text{var}(\alpha)$, $|\alpha|_x \geq 2$,* *then* $L_{Z,\Sigma}(\alpha) \notin \mathrm{CF}$.

At this point, we recall that patterns, provided that they contain repeated variables, describe languages that are generalisations of the copy language, which strongly suggests that these languages are context-sensitive, but not context-free or regular. However, as stated at the beginning of this chapter, for small alphabets this is not necessarily the case and the above results provide a strong indication of where to find this phenomenon of regular and context-free copy languages. More precisely, by Theorems 6.4 and 6.5, the existence of variables with only one occurrence is crucial. Furthermore, since, in the terminal-free case, regular and context-free E-pattern languages are characterised in a compact and simple manner, we should also focus on patterns containing terminal symbols.

Consequently, we concentrate on the question of how the occurrences of terminal symbols in conjunction with non-repeated variables can cause E-pattern languages to become regular. To this end, we shall now consider some simply structured examples of such patterns for which we can formally prove whether or not they describe a regular language with respect to terminal alphabets $\Sigma_2 := \{\mathtt{a}, \mathtt{b}\}$ and $\Sigma_{\geq 3}$, where $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \subseteq \Sigma_{\geq 3}$. Most parts of the following propositions require individual proofs, some of which, in contrast to the simplicity of the example patterns, are surprisingly involved. If, for some pattern α and $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$, $L_{\mathrm{Z},\Sigma_2}(\alpha) \notin \mathrm{REG}$, then $L_{\mathrm{Z},\Sigma_{\geq 3}}(\alpha) \notin \mathrm{REG}$. This follows directly from the fact that regular languages are closed under intersection. Hence, in the following examples, we consider $L_{\mathrm{Z},\Sigma_{\geq 3}}(\alpha)$ only if $L_{\mathrm{Z},\Sigma_2}(\alpha)$ is regular.

Firstly, we consider the pattern $x_1 \cdot d \cdot x_2 x_2 \cdot d' \cdot x_3$, which, for all choices of $d, d' \in \{\mathtt{a}, \mathtt{b}\}$, describes a regular E-pattern language with respect to Σ_2, but a non-regular E-pattern language with respect to $\Sigma_{\geq 3}$.

Proposition 6.6.

$$\begin{aligned}
L_{\mathrm{E},\Sigma_2}(x_1 \ \mathtt{a} \ x_2 \ x_2 \ \mathtt{a} \ x_3) &\in \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_{\geq 3}}(x_1 \ \mathtt{a} \ x_2 \ x_2 \ \mathtt{a} \ x_3) &\notin \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_2}(x_1 \ \mathtt{a} \ x_2 \ x_2 \ \mathtt{b} \ x_3) &\in \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_{\geq 3}}(x_1 \ \mathtt{a} \ x_2 \ x_2 \ \mathtt{b} \ x_3) &\notin \mathrm{REG}\ .
\end{aligned}$$

Proof. Let $\alpha_1 := x_1 \mathtt{a} x_2 x_2 \mathtt{a} x_3$ and let $\alpha_2 := x_1 \mathtt{a} x_2 x_2 \mathtt{b} x_3$. It follows from Lemmas 6.13 and 6.11, respectively, that $L_{\mathrm{E},\Sigma_2}(\alpha_1)$ and $L_{\mathrm{E},\Sigma_2}(\alpha_2)$ are regular languages. Hence, it only remains to prove that $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_1) \notin \mathrm{REG}$ and $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2) \notin \mathrm{REG}$.

We assume that $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_1) \in \mathrm{REG}$ and we shall show that this assumption leads to a contradiction. Let $w := \mathtt{a} \cdot \mathtt{c}^n \mathtt{b} \cdot \mathtt{c}^n \mathtt{b} \cdot \mathtt{a} \in L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_1)$, where n is the constant of Lemma 2.2 (see page 9) with respect to $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_1)$. By Pumping Lemma 2, there exists a word $w' := \mathtt{a} \cdot \mathtt{c}^n\mathtt{b} \cdot \mathtt{c}^{n'}\mathtt{b} \cdot \mathtt{a}$, $n < n'$, with $w' \in L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_1)$, which is obviously not the case.

Similarly, we can show that the assumption $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2) \in \mathrm{REG}$ leads to a contradiction. Let $v := \mathtt{a}\cdot\mathtt{c}^n\mathtt{b}\cdot\mathtt{c}^n\mathtt{b}\cdot\mathtt{b} \in L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2)$, where n is odd and n is greater than the constant of Lemma 2.2 with respect to $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2)$. By Lemma 2.2, there exists a word $v' := \mathtt{a} \cdot \mathtt{c}^n\mathtt{b} \cdot \mathtt{c}^{n'}\mathtt{b} \cdot \mathtt{b}$, $n < n'$, with $v' \in L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2)$, which is not the case, since for every factor $\mathtt{a} \cdot u \cdot \mathtt{b}$ in v', u is not a square. □

Next, we insert another occurrence of a terminal symbol between the two occurrences of x_2, i. e., we consider $\beta := x_1{\cdot}d{\cdot}x_2{\cdot}d'{\cdot}x_2{\cdot}d''{\cdot}x_3$, where $d, d', d'' \in \{\mathtt{a}, \mathtt{b}\}$. Here, we find that $L_{\mathrm{Z},\Sigma}(\beta) \in \mathrm{REG}$ if and only if $\mathrm{Z} = \mathrm{E}$, $\Sigma = \Sigma_2$ and $d = d''$, $d \neq d' \neq d''$.

Proposition 6.7. *For every $Z \in \{\mathrm{E}, \mathrm{NE}\}$,*

$$\begin{aligned}
L_{Z,\Sigma_2}(x_1\ \mathtt{a}\ x_2\ \mathtt{a}\ x_2\ \mathtt{a}\ x_3) &\notin \mathrm{REG}\,,\\
L_{Z,\Sigma_2}(x_1\ \mathtt{a}\ x_2\ \mathtt{a}\ x_2\ \mathtt{b}\ x_3) &\notin \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_2}(x_1\ \mathtt{a}\ x_2\ \mathtt{b}\ x_2\ \mathtt{a}\ x_3) &\in \mathrm{REG}\,,\\
L_{\mathrm{NE},\Sigma_2}(x_1\ \mathtt{a}\ x_2\ \mathtt{b}\ x_2\ \mathtt{a}\ x_3) &\notin \mathrm{REG}\,,\\
L_{Z,\Sigma_{\geq 3}}(x_1\ \mathtt{a}\ x_2\ \mathtt{b}\ x_2\ \mathtt{a}\ x_3) &\notin \mathrm{REG}\ .
\end{aligned}$$

Proof. Let $\alpha_1 := x_1\mathtt{a}x_2\mathtt{a}x_2\mathtt{a}x_3$, $\alpha_2 := x_1\mathtt{a}x_2\mathtt{a}x_2\mathtt{b}x_3$ and $\alpha_3 := x_1\mathtt{a}x_2\mathtt{b}x_2\mathtt{a}x_3$. It follows from Proposition 6.10 that $L_{Z,\Sigma_2}(\alpha_1) \notin \mathrm{REG}$, $L_{Z,\Sigma_2}(\alpha_2) \notin \mathrm{REG}$ and $L_{Z,\Sigma_{\geq 3}}(\alpha_3) \notin \mathrm{REG}$. It remains to prove that $L_{\mathrm{E},\Sigma_2}(\alpha_3) \in \mathrm{REG}$ and $L_{\mathrm{NE},\Sigma_2}(\alpha_3) \notin \mathrm{REG}$. We shall first prove $L_{\mathrm{E},\Sigma_2}(\alpha_3) \in \mathrm{REG}$. To this end, we claim that $L_{\mathrm{E},\Sigma_2}(\alpha_3) = L(r)$, where $r := \Sigma_2^* \cdot \mathtt{a} \cdot (\mathtt{bb})^*\mathtt{b} \cdot \mathtt{a} \cdot \Sigma_2^*$. It can be easily verified that $L(r) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha_3)$. In order to prove the converse, we let h be an arbitrary substitution for α_3. If $h(x_2) \in L(\mathtt{b}^*)$, then $h(\alpha_3) \in L(r)$. Thus, we assume that $h(x_2) = \mathtt{b}^n \cdot \widehat{u} \cdot \mathtt{b}^{n'}$, where $n, n' \in \mathbb{N}_0$, $\widehat{u} \in \Sigma_2^*$ and $\widehat{u}$ starts and ends with an occurrence of $\mathtt{a}$ (note that this includes the case $\widehat{u} = \mathtt{a}$). We note that $h(\alpha_3) = u \cdot \mathtt{a} \cdot \mathtt{b}^n \cdot \widehat{u} \cdot \mathtt{b}^{n+n'+1} \cdot \widehat{u} \cdot \mathtt{b}^{n'} \cdot \mathtt{a} \cdot v$, where $u := h(x_1)$ and $v := h(x_3)$. In order to prove that $h(\alpha_3) \in L(r)$ it is sufficient to identify a factor of form $\mathtt{ab}^k\mathtt{a}$ in $h(\alpha_3)$, where k is odd. If n is odd, then $\mathtt{a} \cdot \mathtt{b}^n \cdot \widehat{u}[1]$ is such a factor and if n' is odd, then $\widehat{u}[-] \cdot \mathtt{b}^{n'} \cdot \mathtt{a}$ is such a factor. If both n and n' are even, then $\widehat{u}[-] \cdot \mathtt{b}^{n+n'+1} \cdot \widehat{u}[1]$ is a factor of form $\mathtt{ab}^k\mathtt{a}$, k odd, since $n + n' + 1$ is odd. Hence, $h(\alpha_3) \in L(r)$ and $L_{\mathrm{E},\Sigma_2}(\alpha_3) \subseteq L(r)$ is implied, which concludes the proof.

Next, in order to prove $L_{\mathrm{NE},\Sigma_2}(\alpha_3) \notin \mathrm{REG}$, we assume to the contrary that $L_{\mathrm{NE},\Sigma_2}(\alpha_3) \in \mathrm{REG}$ and we define $w := \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{ab}^n\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{ab}^n\mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b} \in L_{\mathrm{NE},\Sigma_2}(\alpha_3)$, where n is greater than the constant of Lemma 2.2 (see page 9) with respect to $L_{\mathrm{NE},\Sigma_2}(\alpha_3)$ and n is even. By applying Lemma 2.2, we can obtain the word $w' := \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{ab}^n\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{ab}^{n'}\mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b}$, where $n < n'$ and n' is even. It can be verified that for every factor of form $\mathtt{a} \cdot u \cdot \mathtt{b} \cdot v \cdot \mathtt{a}$, $u, v \in \Sigma_2^+$, in $\mathtt{a} \cdot \mathtt{ab}^n\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{ab}^{n'}\mathtt{a} \cdot \mathtt{a}$, $u \neq v$, which implies that $w' \notin L_{\mathrm{NE},\Sigma_2}(\alpha_3)$. Consequently, with Lemma 2.2, we can conclude that $L_{\mathrm{NE},\Sigma_2}(\alpha_3) \notin \mathrm{REG}$. □

The next type of pattern that we investigate is similar to the first one, but it contains two factors of form xx instead of only one, i. e., $\beta' := x_1 \cdot d \cdot x_2x_2 \cdot d' \cdot x_3x_3 \cdot d'' \cdot x_4$, where $d, d', d'' \in \{\mathtt{a}, \mathtt{b}\}$. Surprisingly, $L_{\mathrm{E},\Sigma_2}(\beta')$ is not regular if $d = d' = d''$, but regular in all other cases. However, if we consider the NE case or alphabet $\Sigma_{\geq 3}$, then β' describes a non-regular language with respect to all choices of $d, d', d'' \in \{\mathtt{a}, \mathtt{b}\}$.

Proposition 6.8. *For every $Z \in \{\mathrm{E}, \mathrm{NE}\}$,*

$$\begin{aligned}
L_{Z,\Sigma_2}(x_1\ \mathtt{a}\ x_2\ x_2\ \mathtt{a}\ x_3\ x_3\ \mathtt{a}\ x_4) &\notin \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_2}(x_1\ \mathtt{a}\ x_2\ x_2\ \mathtt{b}\ x_3\ x_3\ \mathtt{a}\ x_4) &\in \mathrm{REG}\,,\\
L_{\mathrm{NE},\Sigma_2}(x_1\ \mathtt{a}\ x_2\ x_2\ \mathtt{b}\ x_3\ x_3\ \mathtt{a}\ x_4) &\notin \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_{\geq 3}}(x_1\ \mathtt{a}\ x_2\ x_2\ \mathtt{b}\ x_3\ x_3\ \mathtt{a}\ x_4) &\notin \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_2}(x_1\ \mathtt{a}\ x_2\ x_2\ \mathtt{a}\ x_3\ x_3\ \mathtt{b}\ x_4) &\in \mathrm{REG}\,,\\
L_{\mathrm{NE},\Sigma_2}(x_1\ \mathtt{a}\ x_2\ x_2\ \mathtt{a}\ x_3\ x_3\ \mathtt{b}\ x_4) &\notin \mathrm{REG}\,,\\
L_{\mathrm{E},\Sigma_{\geq 3}}(x_1\ \mathtt{a}\ x_2\ x_2\ \mathtt{a}\ x_3\ x_3\ \mathtt{b}\ x_4) &\notin \mathrm{REG}\ .
\end{aligned}$$

Proof. We define $\alpha_1 := x_1\mathtt{a}x_2x_2\mathtt{a}x_3x_3\mathtt{a}x_4$, $\alpha_2 := x_1\mathtt{a}x_2x_2\mathtt{b}x_3x_3\mathtt{a}x_4$ and $\alpha_3 := x_1\mathtt{a}x_2x_2\mathtt{a}x_3x_3\mathtt{b}x_4$. We shall now prove the lemma by proving each of the 7 statements as individual claims.

Claim (1). $L_{\mathrm{Z},\Sigma_2}(\alpha_1) \notin \mathrm{REG}$, $Z \in \{\mathrm{E}, \mathrm{NE}\}$.

Proof. (*Claim* (1)) We first prove that $L_{\mathrm{NE},\Sigma_2}(\alpha_1) \notin \mathrm{REG}$. To this end, we assume to the contrary that $L_{\mathrm{NE},\Sigma_2}(\alpha_1)$ is a regular language and let $k \in \mathbb{N}$ be the constant from Lemma 2.2 (see page 9) with respect to $L_{\mathrm{NE},\Sigma_2}(\alpha_1)$. Furthermore, let h be the substitution defined by $h(x_1) = h(x_4) = \mathtt{b}$, $h(x_2) := \mathtt{b}^n\mathtt{ab}$ and $h(x_3) := \mathtt{b}^m\mathtt{ab}$, where, $k < n$, $6n < m < 12n$ and both n and m are odd. We note that $h(\alpha_1) = \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{b}^n\mathtt{ab} \cdot \mathtt{b}^n\mathtt{ab} \cdot \mathtt{a} \cdot \mathtt{b}^m\mathtt{ab} \cdot \mathtt{b}^m\mathtt{ab} \cdot \mathtt{a} \cdot \mathtt{b}$. By applying Lemma 2.2 first on the second occurrence of factor $\mathtt{b}^n$ and then on the second occurrence of factor $\mathtt{b}^m$, we can obtain the word

$$w := \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{b}^n\mathtt{ab} \cdot \mathtt{b}^{n'}\mathtt{ab} \cdot \mathtt{a} \cdot \mathtt{b}^m\mathtt{ab} \cdot \mathtt{b}^{m'}\mathtt{ab} \cdot \mathtt{a} \cdot \mathtt{b}\,,$$

such that $2n < n' < 4n$ and $12n < m'$. Since we assume that $L_{\mathrm{NE},\Sigma_2}(\alpha_1) \in \mathrm{REG}$, we can conclude from Lemma 2.2 that $w \in L_{\mathrm{NE},\Sigma_2}(\alpha_1)$. Let $p_1, p_2, \ldots, p_7$ be exactly the positions in w where there is an occurrence of $\mathtt{a}$. We shall now show that, for all r, s, t, $1 \leq r < s < t \leq 7$, the factor $w[p_r + 1, p_s - 1]$ is not a non-empty square or the factor $w[p_s + 1, p_t - 1]$ is not a non-empty square. This directly implies that there does not exist a substitution g with $g(\alpha_1) = w$ and, thus, $w \notin L_{\mathrm{NE},\Sigma_2}(\alpha_1)$, which is a contradiction.

We can note that, for all r, s, with $1 \leq r < s \leq 7$, if $s - r$ is even, then $w[p_r + 1, p_s - 1]$ has an odd number of $\mathtt{a}$'s and, thus, it is not a square. Furthermore, since n and m are odd numbers, $w[p_1 + 1, p_2 - 1]$ and $w[p_4 + 1, p_5 - 1]$ cannot be squares and since $w[p_3 + 1, p_4 - 1] = w[p_6 + 1, p_7 - 1] = \mathtt{b}$, these cannot be squares either. The factor $w[p_1 + 1, p_4 - 1] = \mathtt{b}^n\mathtt{ab} \cdot \mathtt{b}^{n'}\mathtt{ab}$ is not a square since $n \neq n'$ and, since $m \neq m'$, the same holds for $w[p_4 + 1, p_7 - 1]$. The factor $w[p_1 + 1, p_6 - 1] = \mathtt{b}^n\mathtt{ab} \cdot \mathtt{b}^{n'}\mathtt{ab} \cdot \mathtt{a} \cdot \mathtt{b}^m\mathtt{ab} \cdot \mathtt{b}^{m'}$ cannot be a square, since $2n < n' < 4n$, $6n < m < 12n$ and $12n < m'$ implies that $n + n' + 2 < m + m' + 1$, and with similar argumentations, we can conclude that factors $w[p_2 + 1, p_7 - 1]$, $w[p_2 + 1, p_5 - 1]$ and $w[p_3 + 1, p_6 - 1]$ are no squares as well. We conclude that the only factors that can possibly be squares are $w[p_2 + 1, p_3 - 1]$ and $w[p_5 + 1, p_6 - 1]$. However, for all r, s, t, $1 \leq r < s < t \leq 7$, it is impossible that $(r, s) = (2, 3)$ and $(s, t) = (5, 6)$. Hence, we obtain a contradiction as described above and, thus, we can conclude that $L_{\mathrm{NE},\Sigma_2}(\alpha_1) \notin \mathrm{REG}$. Moreover, in exactly the same way, we can also prove that $L_{\mathrm{E},\Sigma_2}(\alpha_1) \notin \mathrm{REG}$. This is due to the fact that in the word w there are no two occurrences of symbol $\mathtt{a}$ without occurrences of symbol $\mathtt{b}$ in between them, i. e., we do not need to consider the empty squares. So by exactly the same argumentation, we can show that w is not in $L_{\mathrm{E},\Sigma_2}(\alpha_1) \notin \mathrm{REG}$, which, since $h(\alpha_1)$ clearly is in $L_{\mathrm{E},\Sigma_2}(\alpha_1)$, leads to a contradiction in the same way. □ (*Claim* (1))

Claim (2). $L_{\mathrm{E},\Sigma_2}(\alpha_2) \in \mathrm{REG}$.

Proof. (*Claim* (2)) We claim that $L_{\mathrm{E},\Sigma_2}(\alpha_2) = L(r)$, where $r := \Sigma_2^* \cdot \mathtt{a} \cdot (\mathtt{bb})^* \cdot \mathtt{b} \cdot \mathtt{a} \cdot \Sigma_2^*$. First, we can note that $L(r) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha_2)$ trivially holds. Now let h be an arbitrary substitution. In order to prove that $h(\alpha_2) \in L(r)$, it is sufficient to show that in $h(\alpha_2)$ there occurs a factor of form $\mathtt{a} \cdot \mathtt{b}^{2n-1} \cdot \mathtt{a}$, $n \in \mathbb{N}$.

We first consider the case that $h(x_2) = b^n \cdot u \cdot b^{n'}$, $n, n' \in \mathbb{N}_0$, where u starts and ends with the symbol $\mathtt{a}$. We note that if n is odd, then in $h(\alpha_2)$ there occurs the factor $\mathtt{a} \cdot \mathtt{b}^n \cdot \mathtt{a}$. If, on the other hand, n is even and n' is odd, then $n + n'$ is odd and in $h(\alpha_2)$ there occurs the factor $\mathtt{a} \cdot \mathtt{b}^{n+n'} \cdot \mathtt{a}$. Furthermore, if n' and n are even, then we cannot directly conclude that there exists a factor $\mathtt{a} \cdot \mathtt{b}^{2n-1} \cdot \mathtt{a}$, $n \in \mathbb{N}$, and we have to take a closer look at $h(x_3)$. If $h(x_3) \in L(\mathtt{b}^*)$, then we have the factor $\mathtt{a} \cdot \mathtt{b}^{n'} \cdot \mathtt{b} \cdot h(x_3) \cdot h(x_3) \cdot \mathtt{a}$ that necessarily is of form an $\mathtt{a} \cdot \mathtt{b}^{2n-1} \cdot \mathtt{a}$, $n \in \mathbb{N}$. If, on the other hand, $h(x_3) = \mathtt{b}^m \cdot v \cdot \mathtt{b}^{m'}$, $m, m' \in \mathbb{N}_0$, where v starts and ends with an $\mathtt{a}$, then we have to consider several cases depending on whether m and m' is odd or even. If m is even, then the factor $\mathtt{a} \cdot \mathtt{b}^{n'} \cdot \mathtt{b} \cdot \mathtt{b}^{m} \cdot \mathtt{a}$ occurs in $h(\alpha_2)$, where $n' + m + 1$ is odd. If, on the other hand, m is odd and m' is even, then the factor $\mathtt{a} \cdot \mathtt{b}^{m'+m} \cdot \mathtt{a}$ occurs in $h(\alpha_2)$, where $m' + m$ is odd. Finally, if m' and m are odd, then the factor $\mathtt{a} \cdot \mathtt{b}^{m'} \cdot \mathtt{a}$ occurs in α_2. So we can conclude that if $h(x_2) = b^n \cdot u \cdot b^{n'}$, then there necessarily occurs a factor of form $\mathtt{a} \cdot \mathtt{b}^{2n-1} \cdot \mathtt{a}$, $n \in \mathbb{N}$ in $h(\alpha_2)$.

It remains to consider the case where $h(x_2) \in L(\mathtt{b}^*)$. We first note that if also $h(x_3) \in L(\mathtt{b}^*)$, then the factor $\mathtt{a} \cdot h(x_2) \cdot h(x_2) \cdot \mathtt{b} \cdot h(x_3) \cdot h(x_3) \cdot \mathtt{a}$ occurs in $h(\alpha_2)$, that is of form $\mathtt{a} \cdot \mathtt{b}^{2n-1} \cdot \mathtt{a}$, $n \in \mathbb{N}$. So we need to consider the case that $h(x_3) = \mathtt{b}^m \cdot v \cdot \mathtt{b}^{m'}$, $m, m' \in \mathbb{N}_0$, where v starts and ends with $\mathtt{a}$. If m is even, then the factor $\mathtt{a} \cdot h(x_2) \cdot h(x_2) \cdot \mathtt{b} \cdot \mathtt{b}^m \cdot \mathtt{a}$ occurs in $h(\alpha_2)$, that is of form $\mathtt{a} \cdot \mathtt{b}^{2n-1} \cdot \mathtt{a}$, $n \in \mathbb{N}$. If m' is odd, then the factor $\mathtt{a} \cdot \mathtt{b}^{m'} \cdot \mathtt{a}$ occurs and, finally, if m is odd and m' is even, then factor $\mathtt{a} \cdot \mathtt{b}^{m} \cdot \mathtt{b}^{m'} \cdot \mathtt{a}$ occurs in $h(\alpha_2)$. Consequently, $h(\alpha_2)$ necessarily contains a factor of form $\mathtt{a} \cdot \mathtt{b}^{2n-1} \cdot \mathtt{a}$, $n \in \mathbb{N}$. Thus, $h(\alpha_2) \in L(r)$, which shows that $L(r) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha_2)$ holds. □ (*Claim* (2))

Claim (3). $L_{\mathrm{NE},\Sigma_2}(\alpha_2) \notin \mathrm{REG}$

Proof. (*Claim* (3)) We assume that $L_{\mathrm{NE},\Sigma_2}(\alpha_2)$ is a regular language and we define $w := \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{a}\mathtt{b}^n\mathtt{a} \cdot \mathtt{a}\mathtt{b}^n\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{b} \in L_{\mathrm{NE},\Sigma_2}(\alpha_2)$, where n is greater than the constant of Lemma 2.2 (see page 9) with respect to $L_{\mathrm{NE},\Sigma_2}(\alpha_2)$ and n is even. By pumping, we can produce a word $w' := \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{a}\mathtt{b}^n\mathtt{a} \cdot \mathtt{a}\mathtt{b}^{n'}\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{b}$, where $n < n'$ and n' is even. Now we can note that in w', for every factor of form $\mathtt{a} \cdot u \cdot \mathtt{b} \cdot v \cdot \mathtt{a}$, $u, v \in \Sigma'^+$, in $\mathtt{a} \cdot \mathtt{a}\mathtt{b}^n\mathtt{a} \cdot \mathtt{a}\mathtt{b}^{n'}\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a}$, u is not a square or v is not a square. This implies that $w' \notin L_{\mathrm{NE},\Sigma_2}(\alpha_2)$, which is a contradiction to Lemma 2.2. □ (*Claim* (3))

Claim (4). $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2) \notin \mathrm{REG}$.

Proof. (*Claim* (4)) We assume that $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2) \in \mathrm{REG}$ and we define $w := \mathtt{a} \cdot \mathtt{c}^n\mathtt{b} \cdot \mathtt{c}^n\mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \in L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2)$, where n is greater than the constant of Lemma 2.2 (see page 9) with respect to $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2)$ and n is odd. By pumping, we can produce a word $w' := \mathtt{a} \cdot \mathtt{c}^n\mathtt{b} \cdot \mathtt{c}^{n'}\mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a}$, where $n < n'$. Since in w' there is no factor of form $\mathtt{a} \cdot vv \cdot \mathtt{b}$, $v \in \Sigma_{\geq 3}^*$, we can conclude that $w' \notin L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2)$, which contradicts with Lemma 2.2. □ (*Claim* (4))

Claim (5). $L_{\mathrm{E},\Sigma_2}(\alpha_3) \in \mathrm{REG}$.

Proof. (*Claim* (5)) We claim that $L_{\mathrm{E},\Sigma_2}(\alpha_3) = L(r)$, where $r := \Sigma_2^* \cdot \mathtt{a} \cdot (\mathtt{b}\mathtt{b})^* \cdot \mathtt{a} \cdot \mathtt{b} \cdot \Sigma_2^*$. First, we can note that $L(r) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha_3)$ trivially holds. Now let h be an arbitrary substitution. We shall show that $h(\alpha_3) \in L(r)$, which implies that $L_{\mathrm{E},\Sigma_2}(\alpha_3) \subseteq L(r)$. If $h(x_2)$ starts with the symbol $\mathtt{a}$, $h(x_2)$ ends with the symbol $\mathtt{a}$ or $h(x_3)$

starts with the symbol $\mathtt{a}$, then the factor $\mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b}$ occurs in $h(\alpha_3)$, which implies that $h(\alpha_3) \in L(r)$. Hence, we only need to consider the following case: if $h(x_2)$ is non-empty, then it starts and ends with the symbol $\mathtt{b}$ and if $h(x_3)$ is non-empty, then it starts with the symbol $\mathtt{b}$. Next, we can note that if $h(x_2)$ is empty or $h(x_2) = \mathtt{b}^n$, $n \in \mathbb{N}$, then, since $h(x_3)$ is either empty or it starts with $\mathtt{b}$, the factor $\mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b}$ occurs in $h(\alpha_3)$ or the factor $\mathtt{a} \cdot \mathtt{b}^{2n} \cdot \mathtt{a} \cdot \mathtt{b}$ occurs in $h(\alpha_3)$, respectively, which implies that $h(\alpha_3) \in L(r)$. Therefore, we need to take a closer look at the case that $h(x_2) = \mathtt{b}^n \cdot u \cdot \mathtt{b}^{n'}$, $n, n' \in \mathbb{N}$, where u starts and ends with the symbol $\mathtt{a}$. If u contains the factor $\mathtt{a} \cdot \mathtt{a}$, then the factor $\mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b}$ is contained in $h(\alpha_3)$, thus, $h(\alpha_3) \in L(r)$. If, on the other hand, u does not contain the factor $\mathtt{a} \cdot \mathtt{a}$, i. e., every $\mathtt{a}$ in u is followed by a $\mathtt{b}$, then we need to use a different argumentation. We note that in $h(\alpha_3)$ the factors $\mathtt{a} \cdot \mathtt{b}^n \cdot \mathtt{a} \cdot \mathtt{b}$ and $\mathtt{a} \cdot \mathtt{b}^{n'} \cdot \mathtt{b}^n \cdot \mathtt{a} \cdot \mathtt{b}$ occur. Furthermore, since $h(x_3)$ is either empty or it starts with $\mathtt{b}$, we can also conclude that the factor $\mathtt{a} \cdot \mathtt{b}^{n'} \cdot \mathtt{a} \cdot \mathtt{b}$ occurs in $h(\alpha_3)$. We can now observe that if n is even or n' is even, then $h(\alpha_3) \in L(r)$. Furthermore, if n is odd and n' is odd, then $n + n'$ is even and, thus, $h(\alpha_3) \in L(r)$. Consequently, for all possible cases, $h(\alpha_3) \in L(r)$, which implies that $L_{\mathrm{E},\Sigma_2}(\alpha_3) \subseteq L(r)$. □ (*Claim* (5))

Claim (6). $L_{\mathrm{NE},\Sigma_2}(\alpha_3) \notin \mathrm{REG}$.

Proof. (*Claim* (6)) We assume that $L_{\mathrm{NE},\Sigma_2}(\alpha_3)$ is a regular language and we define $w := \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{b}^n \mathtt{a} \cdot \mathtt{b}^n \mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \in L_{\mathrm{NE},\Sigma_2}(\alpha_3)$, where n is greater than the constant of Lemma 2.2 (see page 9) with respect to $L_{\mathrm{NE},\Sigma_2}(\alpha_3)$ and n is odd. By pumping, we can produce a word $w' := \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{b}^n \mathtt{a} \cdot \mathtt{b}^{n'} \mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a}$, where $n < n'$ and n' is odd. Now we can note that in w' there is no factor of form $\mathtt{a} \cdot vv \cdot \mathtt{a}$, $v \in \Sigma_2^+$. Thus, $w' \notin L_{\mathrm{NE},\Sigma_2}(\alpha_3)$, which contradicts Lemma 2.2. □ (*Claim* (6))

Claim (7). $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_3) \notin \mathrm{REG}$

Proof. (*Claim* (7)) This claim can be proved analogously to the claim $L_{\mathrm{E},\Sigma_{\geq 3}}(\alpha_2) \notin \mathrm{REG}$. □ (*Claim* (7))

This concludes the proof of the proposition.

□

We call two patterns $\alpha, \beta \in (\Sigma_2 \cup X)^*$ *almost identical* if and only if $|\alpha| = |\beta|$ and, for every i, $1 \leq i \leq |\alpha|$, $\alpha[i] \neq \beta[i]$ implies $\alpha[i], \beta[i] \in \Sigma_2$. The above examples show that even for almost identical patterns α and β, we can have the situation that α describes a regular and β a non-regular language. Even if α and β are almost identical and further satisfy $|\alpha|_{\mathtt{a}} = |\beta|_{\mathtt{a}}$ and $|\alpha|_{\mathtt{b}} = |\beta|_{\mathtt{b}}$, then it is still possible that α describes a regular and β a non-regular language (cf. Proposition 6.7 above). This implies that the regular E-pattern languages over an alphabet with size 2 require a characterisation that caters for the exact order of terminal symbols in the patterns.

The examples considered in Propositions 6.6 and 6.8 mainly consist of factors of form $d \cdot xx \cdot d'$, $d, d' \in \Sigma_2$, where x does not have any other occurrence in the pattern. Hence, it might be worthwhile to investigate the question of whether or not patterns can also describe regular languages if we allow them to contain factors of form $d \cdot x^k \cdot d'$, where $k \geq 3$ and there is no other occurrence of x in the pattern. In the next result, we state that if a pattern α contains a factor $d \cdot x^k \cdot d'$ with $d = d'$, $k \geq 3$ and $|\alpha|_x = k$, then, for every $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$, its Z-pattern language with respect to any alphabet of size at least 2 is not regular and, furthermore, for alphabets of size at least 3, we can show that this also holds for $d \neq d'$.

Theorem 6.9. *Let Σ and Σ' be terminal alphabets with $\{\mathtt{a}, \mathtt{b}\} \subseteq \Sigma$ and $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \subseteq \Sigma'$. Let $\alpha := \alpha_1 \cdot \mathtt{a} \cdot z^l \cdot \mathtt{a} \cdot \alpha_2$, let $\beta := \beta_1 \cdot \mathtt{a} \cdot z^l \cdot \mathtt{c} \cdot \beta_2$, where $z \in X$, $\alpha_1, \alpha_2 \in ((\Sigma \cup X) \setminus \{z\})^*$, $\beta_1, \beta_2 \in ((\Sigma' \cup X) \setminus \{z\})^*$ and $l \geq 3$. Then, for every $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$, $L_{\mathrm{Z},\Sigma}(\alpha) \notin \mathrm{REG}$ and $L_{\mathrm{Z},\Sigma'}(\beta) \notin \mathrm{REG}$.*

Proof. We first prove that $L_{\mathrm{NE},\Sigma}(\alpha) \notin \mathrm{REG}$. Let k be the constant of Lemma 2.1 (see page 8) with respect to $L_{\mathrm{NE},\Sigma}(\alpha)$ and let h be the substitution defined by $h(z) := \mathtt{b}^{k'} \cdot \mathtt{a} \cdot \mathtt{b}$, where $k' \geq k$, $k' \mod l = 1$, and $h(x) := \mathtt{b}$, $x \in \mathrm{var}(\alpha) \setminus \{z\}$. We can note that $w := h(\alpha) = u \cdot \mathtt{a} \cdot (\mathtt{b}^{k'} \cdot \mathtt{a} \cdot \mathtt{b})^l \cdot \mathtt{a} \cdot v$, where u and v equal $h(\alpha_1)$ and $h(\alpha_2)$, respectively. Obviously, $|w| \geq k$ and $w \in L_{\mathrm{NE},\Sigma}(\alpha)$. We shall now show that for every factorisation $w = v_1 \cdot v_2 \cdot v_3$ with $|v_1 v_2| \leq k$ and $v_2 \neq \varepsilon$, there exists a $t \in \mathbb{N}_0$ such that $v_1 \cdot v_2^t \cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$, which, by Lemma 2.1, proves that $L_{\mathrm{NE},\Sigma}(\alpha)$ is not regular. We first note that $|v_1 v_2| \leq k$ and $v_2 \neq \varepsilon$ implies that

- $v_2 = u'$, where u' is a factor of u with $1 \leq |u'| \leq k$ or
- $v_2 = u' \cdot \mathtt{a} \cdot \mathtt{b}^i$, where u' is a suffix of u and $0 \leq i \leq k - (|u'| + 1)$ or
- $v_2 = \mathtt{b}^i$, where $1 \leq i \leq k - (|u| + 1)$.

We first consider the case that $v_2 = \mathtt{b}^i$, $1 \leq i \leq k - (|u| + 1)$, and, furthermore, we assume that i is a multiple of l, which implies that $k' - i$ is not a multiple of l, since k' is not a multiple of l. Next, we consider the word $v_1 \cdot v_2^0 \cdot v_3 = u \cdot \mathtt{a} \cdot \mathtt{b}^{k'-i} \cdot \mathtt{a} \cdot \mathtt{b} \cdot (\mathtt{b}^{k'} \cdot \mathtt{a} \cdot \mathtt{b})^{l-1} \cdot \mathtt{a} \cdot v$. We want to show that $v_1 \cdot v_2^0 \cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$. To this end, we first note that if there exists a substitution g with $g(\alpha) = v_1 \cdot v_2^0 \cdot v_3$, then, since u and v are obtained by substituting all variables of α_1 and α_2 by a word of length 1, u must be a prefix of $g(\alpha_1)$ and v must be a suffix of $g(\alpha_2)$. This implies that, in order to conclude $v_1 \cdot v_2^0 \cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$, it is sufficient to show that every factor of form $\mathtt{a} \cdot w \cdot \mathtt{a}$, $w \in \Sigma^*$, of $\mathtt{a} \cdot \mathtt{b}^{k'-i} \cdot \mathtt{a} \cdot \mathtt{b} \cdot (\mathtt{b}^{k'} \cdot \mathtt{a} \cdot \mathtt{b})^{l-1} \cdot \mathtt{a}$ is not of form $\mathtt{a} \cdot (w')^l \cdot \mathtt{a}$, $w' \in \Sigma^*$. We first note that the factor $\mathtt{a} \cdot \mathtt{b}^{k'-i} \cdot \mathtt{a} \cdot \mathtt{b} \cdot (\mathtt{b}^{k'} \cdot \mathtt{a} \cdot \mathtt{b})^{l-1} \cdot \mathtt{a}$ is obviously not of this form. For all other factors $\mathtt{a}\cdot w\cdot\mathtt{a}$ of $\mathtt{a}\cdot\mathtt{b}^{k'-i}\cdot\mathtt{a}\cdot\mathtt{b}\cdot(\mathtt{b}^{k'}\cdot\mathtt{a}\cdot\mathtt{b})^{l-1}\cdot\mathtt{a}$, where $|w|_{\mathtt{a}} \geq 1$, we have $|w|_{\mathtt{a}} \leq l-1$, thus, they cannot be of form $\mathtt{a}\cdot(w')^l\cdot\mathtt{a}$, $w' \in \Sigma^*$, either. Consequently, it remains to take a closer look at the factors $\mathtt{a}\cdot w\cdot\mathtt{a}$, where $|w|_{\mathtt{a}} = 0$. We can observe that for these factors the length of w is either $k'+1$, $k'-i$ or 1, and, since $l \geq 3$, neither $k'+1$, $k'-i$ nor 1 is a multiple of l. This implies that these factors are also not of form $\mathtt{a}\cdot(w')^l\cdot\mathtt{a}$, $w' \in \Sigma^*$, which proves that $v_1\cdot v_2^0\cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$.

Next, we consider the case that $v_2 = \mathtt{b}^i$, where i is not a multiple of l. Now if $k'-i$ is not a multiple of l, then we can show in exactly the same way as before that $v_1\cdot v_2^0\cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$. If, on the other hand, $k'-i$ is a multiple of l, then, since $k' \mod l = 1$, we can conclude that $i \mod l = 1$ and, thus, $k'+i \mod l = 2$. We now consider the word $v_1\cdot v_2^2\cdot v_3 = u\cdot\mathtt{a}\cdot\mathtt{b}^{k'+i}\cdot\mathtt{a}\cdot\mathtt{b}\cdot(\mathtt{b}^{k'}\cdot\mathtt{a}\cdot\mathtt{b})^{l-1}\cdot\mathtt{a}\cdot v$. As demonstrated above, $k'+i$ is not a multiple of l and, thus, we can apply the same argumentation as before in order to show that $v_1\cdot v_2^2\cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$.

In order to conclude the proof, we have to consider the case that $v_2 = u'$, where u' is a factor of u with $1 \leq |u'| \leq k$ and the case that $v_2 = u'\cdot\mathtt{a}\cdot\mathtt{b}^i$, where u' is a suffix of u and $0 \leq i \leq k-(|u'|+1)$. We first assume that $v_2 = u'$ with $u = q_1\cdot u'\cdot q_2$, $1 \leq |u'| \leq k$, and consider the word $v_1\cdot v_2^0\cdot v_3 := q_1\cdot q_2\cdot\mathtt{a}\cdot(\mathtt{b}^{k'}\cdot\mathtt{a}\cdot\mathtt{b})^l\cdot\mathtt{a}\cdot v$. If there exists a substitution g with $g(\alpha) = v_1\cdot v_2^0\cdot v_3$, then, since $|q_1\cdot q_2| < |u|$, we can conclude that $q_1\cdot q_2\cdot\mathtt{a}$ is a prefix of $g(\alpha_1)$, which implies that, in order to conclude $v_1\cdot v_2^0\cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$, it is sufficient to show that every factor $\mathtt{a}\cdot w\cdot\mathtt{a}$, $w \in \Sigma^*$ of $(\mathtt{b}^{k'}\cdot\mathtt{a}\cdot\mathtt{b})^l\cdot\mathtt{a}$ is not of form $\mathtt{a}\cdot(w')^l\cdot\mathtt{a}$, $v \in \Sigma^*$. This can be easily seen, since $|(\mathtt{b}^{k'}\cdot\mathtt{a}\cdot\mathtt{b})^l\cdot\mathtt{a}|_{\mathtt{a}} \leq l+1$ and, for every factor of form $\mathtt{a}\cdot w\cdot\mathtt{a}$, where $|w|_{\mathtt{a}} = 0$, we can observe that $|w|$ equals either $k'+1$ or 1, and, since $l \geq 3$, neither of these is a multiple of l. If $v_2 = u'\cdot\mathtt{a}\cdot\mathtt{b}^i$, where u' is a suffix of u and $0 \leq i \leq k-(|u'|+1)$, then we can argue analogously. This proves that for every factorisation $w = v_1\cdot v_2\cdot v_3$ with $|v_1v_2| \leq k$ and $v_2 \neq \varepsilon$, there exists a $t \in \mathbb{N}_0$ such that $v_1\cdot v_2^t\cdot v_3 \notin L_{\mathrm{NE},\Sigma}(\alpha)$, which, by Lemma 2.1, implies that $L_{\mathrm{NE},\Sigma}(\alpha)$ is not regular.

It can be shown analogously that $L_{\mathrm{E},\Sigma}(\alpha) \notin \mathrm{REG}$. The only difference of the prove is that the substitution h erases all variables of α_1 and α_2 instead of substituting them by $\mathtt{b}$. This is necessary to be able to assume that for any other substitution g, $h(\alpha_1)$ must be a prefix of $g(\alpha_1)$ and $h(\alpha_2)$ must be a suffix of $g(\alpha_2)$.

It remains to show that $L_{\mathrm{NE},\Sigma'}(\beta) \notin \mathrm{REG}$ and $L_{\mathrm{E},\Sigma'}(\beta) \notin \mathrm{REG}$. We shall first show that $L_{\mathrm{NE},\Sigma'}(\beta) \notin \mathrm{REG}$. Let k be the constant of Lemma 2.2 (see page 9) with respect to $L_{\mathrm{NE},\Sigma'}(\beta)$ and let h be the substitution defined by $h(z) := \mathtt{b}^k\cdot\mathtt{a}$ and $h(x) := \mathtt{b}$, $x \in \mathrm{var}(\beta)\backslash\{z\}$. We can note that $w := h(\beta) = u\cdot\mathtt{a}\cdot(\mathtt{b}^k\cdot\mathtt{a})^l\cdot\mathtt{c}\cdot v$, where u and v equal $h(\beta_1)$ and $h(\beta_2)$, respectively. Obviously, $|w| \geq k$ and $w \in L_{\mathrm{NE},\Sigma'}(\beta)$. By applying Lemma 2.2, we can obtain a word $w' := u\cdot\mathtt{a}\cdot\mathtt{b}^{k'}\cdot\mathtt{a}\cdot(\mathtt{b}^k\cdot\mathtt{a})^{l-1}\cdot\mathtt{c}\cdot v$ with

$k < k'$. We shall now show that $w' \notin L_{\mathrm{NE},\Sigma'}(\beta)$. To this end, we first note that if there exists a substitution g with $g(\beta) = w'$, then, since u and v are obtained by substituting all variables of β_1 and β_2 by a word of length 1, u must be a prefix of $g(\beta_1)$ and v must be a suffix of $g(\beta_2)$. This implies that, in order to conclude $w' \notin L_{\mathrm{NE},\Sigma'}(\beta)$, it is sufficient to show that every factor of form $\mathtt{a} \cdot w \cdot \mathtt{c}$, $w \in \Sigma'^{+}$, in $\mathtt{a} \cdot \mathtt{b}^{k'} \cdot \mathtt{a} \cdot (\mathtt{b}^{k} \cdot \mathtt{a})^{l-1} \cdot \mathtt{c}$ is not of form $\mathtt{a} \cdot (w')^{l} \cdot \mathtt{c}$, $w' \in \Sigma'^{+}$. It is easy to see that $\mathtt{a} \cdot \mathtt{b}^{k'} \cdot \mathtt{a} \cdot (\mathtt{b}^{k} \cdot \mathtt{a})^{l-1} \cdot \mathtt{c}$ is not of this form and for all other factors of form $\mathtt{a} \cdot w \cdot \mathtt{c}$, $w \in \Sigma'^{+}$, we have $|w|_{\mathtt{a}} \leq l - 1$, which implies that w cannot be of form $(w')^{l}$, $w' \in \Sigma'^{+}$. This implies that $w' \notin L_{\mathrm{NE},\Sigma'}(\beta)$ and, thus, $L_{\mathrm{NE},\Sigma'}(\beta) \notin \mathrm{REG}$.

It can be shown analogously that $L_{\mathrm{E},\Sigma'}(\beta) \notin \mathrm{REG}$. The only difference of the prove is that the substitution h erases all variables of β_1 and β_2 instead of substituting them by $\mathtt{b}$. □

In the examples of Propositions 6.6, 6.7 and 6.8 as well as in the above theorem, we do not consider the situation that two occurrences of the same variable are separated by a terminal symbol. In the next result, we state that, in certain cases, this implies non-regularity of pattern languages.

Proposition 6.10. *Let Σ and Σ' be terminal alphabets with $|\Sigma| \geq 2$ and $|\Sigma'| \geq 3$ and let $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. Furthermore, let $\alpha_1 \in (\Sigma \cup X)^*$ and $\alpha_2 \in (\Sigma' \cup X)^*$ be patterns.*

1. *If there exists a $\gamma \in (\Sigma \cup X)^*$ with $|\operatorname{var}(\gamma)| \geq 1$ such that, for some $d \in \Sigma$,*
   - *$\alpha_1 = \gamma \cdot d \cdot \delta$ and $\operatorname{var}(\gamma) \subseteq \operatorname{var}(\delta)$,*
   - *$\alpha_1 = \gamma \cdot d \cdot \delta$ and $\operatorname{var}(\delta) \subseteq \operatorname{var}(\gamma)$ or*
   - *$\alpha_1 = \beta \cdot d \cdot \gamma \cdot d \cdot \delta$ and $\operatorname{var}(\gamma) \subseteq (\operatorname{var}(\beta) \cup \operatorname{var}(\delta))$,*

   then $L_{\mathrm{Z},\Sigma}(\alpha_1) \notin \mathrm{REG}$.
2. *If in α_2 there exists a non-empty variable block, all the variables of which also occur outside this block, then $L_{\mathrm{Z},\Sigma'}(\alpha_2) \notin \mathrm{REG}$.*

Proof. We first prove point 1 of the proposition. To this end, we assume that $L_{\mathrm{NE},\Sigma}(\alpha)$ is a regular language. Furthermore, we assume that for α one of the three cases described in point 1 is satisfied with $d = \mathtt{b}$. Let w be the word obtained from α by substituting all variables in $\operatorname{var}(\gamma)$ by $\mathtt{a}^n$, where n is the constant of Pumping Lemma 2 with respect to $L_{\mathrm{NE},\Sigma}(\alpha)$, and all other variables by $\mathtt{a}$. By applying Lemma 2.2 (see page 9), we can obtain a word w' from w by pumping the part that results from γ without pumping the other parts of the word. Since every variable of γ occurs in the other parts as well, and since we only substituted the variables that do not occur in γ by $\mathtt{a}$, we can conclude that w' is not in

$L_{\mathrm{NE},\Sigma}(\alpha)$, which proves that $L_{\mathrm{NE},\Sigma}(\alpha) \notin \mathrm{REG}$. Furthermore, the above proof can be applied in exactly the same way in order to show that $L_{\mathrm{E},\Sigma}(\alpha_1) \notin \mathrm{REG}$.

Point 2 of the proposition can be proved analogously. If in α_2 there exists a variable block, all the variables of which also occur outside this block, then we can substitute all variables in this block by $\mathtt{a}^n$, where n is the constant of Lemma 2.2 with respect to $L_{\mathrm{NE},\Sigma'}(\alpha)$ and, since $|\Sigma'| \geq 3$, we can assume that the variable block is not delimited by $\mathtt{a}$ to either side. Furthermore, we substitute all variables that do not occur in the variable block by $\mathtt{a}$. Now we can show in exactly the same way as before that the thus obtained word is not in $L_{\mathrm{NE},\Sigma'}(\alpha)$, which proves $L_{\mathrm{NE},\Sigma'}(\alpha) \notin \mathrm{REG}$ and $L_{\mathrm{E},\Sigma'}(\alpha) \notin \mathrm{REG}$ can be shown in exactly the same way. □

We conclude this section by referring to the examples presented in Propositions 6.6, 6.7 and 6.8, which, as described above, suggest that complete characterisations of the regular E-pattern languages over small alphabets might be extremely complex. In the next section, we wish to find out about the fundamental mechanisms of the above example patterns that are responsible for the regularity of their pattern languages. Intuitively speaking, some of these example patterns describe regular languages, because they contain a factor that is less complex than it seems to be, e. g., for the pattern $\beta := x_1 \cdot \mathtt{a} \cdot x_2 x_2 \cdot \mathtt{a} \cdot x_3 x_3 \cdot \mathtt{b} \cdot x_4$ it can be shown that the factor $\mathtt{a} \cdot x_2 x_2 \cdot \mathtt{a} \cdot x_3 x_3 \cdot \mathtt{b}$ could be replaced by $\mathtt{a} \cdot x_{(\mathtt{bb})^*} \cdot \mathtt{a} \cdot \mathtt{b}$ (where $x_{(\mathtt{bb})^*}$ is a special variable that can only be substituted by a unary string over $\mathtt{b}$ of even length) without changing its E-pattern language with respect to Σ_2. This directly implies that $L_{\mathrm{E},\Sigma_2}(\beta) = L(\Sigma_2^* \cdot \mathtt{a}(\mathtt{bb})^*\mathtt{ab} \cdot \Sigma_2^*)$, which shows that $L_{\mathrm{E},\Sigma_2}(\beta) \in \mathrm{REG}$.

In the next section, by generalising the above observation, we develop a method that allows us to transform complicated patterns into shorter and equivalent ones that can be easily seen to describe a regular language.

6.3 Regularity of E-Pattern Languages: A Sufficient Condition Taking Terminal Symbols into Account

In this section we investigate the phenomenon that a whole factor in a pattern can be substituted by a less complex one, without changing the corresponding pattern language. This technique can be used in order to show that a complicated pattern is equivalent to one that can be easily seen to describe a regular language.

For the sake of a better presentation of our results, we slightly redefine the concept of patterns. A *pattern with regular expressions* is a pattern that may contain regular expressions. Such a regular expressions is then interpreted as a variable with only one occurrence that can only be substituted by words described by the corresponding regular expression. For example $L_{\mathrm{E},\Sigma_2}(x_1\mathtt{b}^*x_1\mathtt{a}^*) = \{h(x_1x_2x_1x_3) \mid h \text{ is a substitution with } h(x_2) \in L(\mathtt{b}^*), h(x_3) \in L(\mathtt{a}^*)\}$. Obviously, patterns with regular expressions exceed the expressive power of classical patterns. However, we shall use this concept exclusively in the case where a classical pattern is equivalent to a pattern with regular expressions. For example, the pattern $x_1 \cdot \mathtt{a} \cdot x_2x_3x_3x_2 \cdot \mathtt{a} \cdot x_4$ is equivalent to the pattern $x_1 \cdot \mathtt{a}(\mathtt{bb})^*\mathtt{a} \cdot x_2$ (see Lemma 6.13).

Next, we present a lemma that states that in special cases whole factors of a pattern can be removed without changing the corresponding pattern language.

Lemma 6.11. *Let* $\alpha := \beta \cdot y \cdot \beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{b} \cdot \delta' \cdot z \cdot \delta$, *where* $\beta, \delta \in (\Sigma_2 \cup X)^*$, $\beta', \gamma, \delta' \in X^*$, $y, z \in X$ *and* $|\alpha|_y = |\alpha|_z = 1$. *Then* $L_{\mathrm{E},\Sigma_2}(\alpha) \subseteq L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{ab} \cdot z \cdot \delta)$. *If, furthermore,* $\mathrm{var}(\beta' \cdot \gamma \cdot \delta') \cap \mathrm{var}(\beta \cdot \delta) = \emptyset$, *then also* $L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{ab} \cdot z \cdot \delta) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha)$.

Proof. Let h be an arbitrary substitution. We obtain a substitution g from h in the following way. For every $x \in \mathrm{var}(\beta \cdot \delta) \setminus \{y, z\}$, we define $g(x) := h(x)$. If the last symbol in $h(\gamma)$ is $\mathtt{a}$, then we define $g(y) := h(y \cdot \beta') \cdot \mathtt{a} \cdot h(\gamma)[1, |h(\gamma)| - 1]$ and $g(z) := h(\delta' \cdot z)$. If the first symbol in $h(\gamma)$ is $\mathtt{b}$, then we define $g(y) := h(y \cdot \beta')$ and $g(z) := h(\gamma)[2, |h(\gamma)|] \cdot \mathtt{b} \cdot h(\delta' \cdot z)$. If the last symbol in $h(\gamma)$ is $\mathtt{b}$ and the first symbol in $h(\gamma)$ is $\mathtt{a}$, then $h(\gamma) = u \cdot \mathtt{a} \cdot \mathtt{b} \cdot v$, $u, v \in \Sigma_2^*$. In this case, we define $g(y) := h(y \cdot \beta') \cdot \mathtt{a} \cdot u$ and $g(z) := v \cdot \mathtt{b} \cdot h(\delta' \cdot z)$. We observe that in all these cases we have $g(\beta \cdot y \cdot \mathtt{a} \cdot \mathtt{b} \cdot z \cdot \delta) = h(\alpha)$ and, thus, $L_{\mathrm{E},\Sigma_2}(\alpha) \subseteq L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{a} \cdot \mathtt{b} \cdot z \cdot \delta)$.

Next, we assume further that $\mathrm{var}(\beta' \cdot \gamma \cdot \delta') \cap \mathrm{var}(\beta \cdot \delta) = \emptyset$. Let g be a substitution. Obviously, $g(\beta \cdot y \cdot \mathtt{a} \cdot \mathtt{b} \cdot z \cdot \delta) = h(\alpha)$, where $h(x) := g(x)$ if $x \in (\mathrm{var}(\beta \cdot \delta) \cup \{y, z\})$ and $h(x) := \varepsilon$ otherwise. This implies $L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{a} \cdot \mathtt{b} \cdot z \cdot \delta) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha)$. □

The fact that $L_{\mathrm{E},\Sigma_2}(x_1 \cdot \mathtt{a} \cdot x_2x_2 \cdot \mathtt{b} \cdot x_3) \in \mathrm{REG}$ has already been stated in Proposition 6.6. We can now note that this result is a simple application of Lemma 6.11, which implies $L_{\mathrm{E},\Sigma_2}(x_1 \cdot \mathtt{a} \cdot x_2x_2 \cdot \mathtt{b} \cdot x_3) = L_{\mathrm{E},\Sigma_2}(x_1 \cdot \mathtt{ab} \cdot x_3)$. It is straightforward to construct more complex applications of Lemma 6.11 and it is also possible to apply it in an iterative way. For example, by applying Lemma 6.11 twice, we can show that

$$\begin{aligned}
&L_{\mathrm{E},\Sigma_2}(x_1x_2x_3 \cdot \mathtt{a} \cdot x_2x_4 \cdot \mathtt{b} \cdot x_3x_4x_5x_6 \cdot \mathtt{b} \cdot x_6x_7 \cdot \mathtt{a} \cdot x_7x_8 \cdot \mathtt{b} \cdot x_9 \cdot \mathtt{a} \cdot x_{10}) = \\
&L_{\mathrm{E},\Sigma_2}(x_1 \cdot \mathtt{ab} \cdot x_5x_6 \cdot \mathtt{b} \cdot x_6x_7 \cdot \mathtt{a} \cdot x_7x_8 \cdot \mathtt{b} \cdot x_9 \cdot \mathtt{a} \cdot x_{10}) = \\
&L_{\mathrm{E},\Sigma_2}(x_1 \cdot \mathtt{ab} \cdot x_5 \cdot \mathtt{ba} \cdot x_8 \cdot \mathtt{b} \cdot x_9 \cdot \mathtt{a} \cdot x_{10}) \in \mathrm{REG}\ .
\end{aligned}$$

In the previous lemma, it is required that the factor γ is delimited by different terminal symbols and, in the following, we shall see that an extension of the statement of Lemma 6.11 for the case that γ is delimited by the same terminal symbols, is much more difficult to prove.

Roughly speaking, Lemma 6.11 holds due to the following reasons. Let $\alpha := y \cdot \beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{b} \cdot \delta' \cdot z$ be a pattern that satisfies the conditions of Lemma 6.11, then, for any substitution h (with respect to Σ_2), $h(\alpha)$ necessarily contains the factor $\mathtt{ab}$. Conversely, since y and z are variables with only one occurrence and there are no terminals in $\beta' \cdot \gamma \cdot \delta'$, α can be mapped to every word that contains the factor $\mathtt{ab}$. On the other hand, for $\alpha' := y \cdot \beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{a} \cdot \delta' \cdot z$, $h(\alpha')$ does not necessarily contain the factor $\mathtt{aa}$ and it is not obvious if the factor $\beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{a} \cdot \delta'$ collapses to some simpler structure, as it is the case for α. In fact, Theorem 6.9 states that if $\beta' = \delta' = \varepsilon$ and $\gamma = x^3$, then $L_{\mathrm{E},\Sigma_2}(\alpha') \notin \mathrm{REG}$.

However, by imposing a further restriction with respect to the factor γ, we can extend Lemma 6.11 to the case where γ is delimited by the same terminal symbol. In order to prove this result, the next lemma is crucial, which states that for any terminal-free pattern that is delimited by two occurrences of symbols $\mathtt{a}$ and that has an even number of occurrences for every variable, if we apply any substitution to this pattern, we will necessarily obtain a word that contains a unary factor over $\mathtt{b}$ of even length that is delimited by two occurrences of $\mathtt{a}$.

Lemma 6.12. *Let $\alpha \in X^*$ such that, for every $x \in \mathrm{var}(\alpha)$, $|\alpha|_x$ is even. Then every $w \in L_{\mathrm{E},\Sigma_2}(\mathtt{a} \cdot \alpha \cdot \mathtt{a})$ contains a factor $\mathtt{ab}^{2n}\mathtt{a}$, $n \in \mathbb{N}_0$.*

Proof. First, we introduce the following definition that is convenient for this proof. A factor of form $\mathtt{ab}^n\mathtt{a}$, $n \in \mathbb{N}_0$, is called a $\mathtt{b}$-*segment*. If n is even, then $\mathtt{ab}^n\mathtt{a}$ is an *even* $\mathtt{b}$-*segment* and if n is odd, then $\mathtt{ab}^n\mathtt{a}$ is an *odd* $\mathtt{b}$-*segment*. In a word $w \in \{\mathtt{a}, \mathtt{b}\}^*$, $\mathtt{b}$-segments that share exactly one occurrence of symbol $\mathtt{a}$ are considered to be distinct $\mathtt{b}$-segments, e. g., in $\mathtt{aab}^2\mathtt{ab}^4\mathtt{abab}^7\mathtt{a}$, there are 5 $\mathtt{b}$-segments, 3 of which are even $\mathtt{b}$-segments.

Before we can prove the statement of the lemma, we first prove the following claim:

Claim (1). Let $w_1 \in (\mathtt{a} \cdot \Sigma_2^*)$, $w_3 \in (\Sigma_2^* \cdot \mathtt{a})$, $w_2, v \in \Sigma_2^*$ and v does not contain any even $\mathtt{b}$-segment. If $w_1 \cdot w_2 \cdot w_3$ has an odd number of even $\mathtt{b}$-segments, then $w_1 \cdot v \cdot w_2 \cdot v \cdot w_3$ has an odd number of even $\mathtt{b}$-segments as well.

Proof. (*Claim* (1)) We assume that for w_1, w_2, w_3 and v the conditions of the lemma are satisfied and, for the sake of convenience, we define $w := w_1 \cdot w_2 \cdot w_3$ and $w' := w_1 \cdot v \cdot w_2 \cdot v \cdot w_3$. Intuitively, the statement of the lemma can be rephrased as follows. No matter where the two occurrences of v are inserted into w, the total number of even $\mathtt{b}$-segments increases or decreases only by an even number. Since v does not contain any even $\mathtt{b}$-segment, only the (possibly empty) prefix or suffix over $\mathtt{b}$ of v can turn odd $\mathtt{b}$-segments of w in even ones or vice versa. We shall first consider the case that $|w_2|_{\mathtt{a}} \geq 1$, i.e., w_2 contains at least one occurrence of symbol $\mathtt{a}$, and we recall that, since $w_1 \in (\mathtt{a} \cdot \Sigma_2^*)$ and $w_3 \in (\Sigma_2^* \cdot \mathtt{a})$, w_1 has a suffix of form $\mathtt{ab}^*$ and w_3 has a prefix of form $\mathtt{b}^*\mathtt{a}$. Furthermore, since $|w_2|_{\mathtt{a}} \geq 1$, w_2 has a prefix of form $\mathtt{b}^*\mathtt{a}$ and a suffix of form $\mathtt{ab}^*$. In summary, this implies that we can write w' as

$$w' = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^n \cdot v \cdot \mathtt{b}^{n'} \cdot w'_2 \cdot \mathtt{b}^m \cdot v \cdot \mathtt{b}^{m'} \cdot \mathtt{a} \cdot w'_3\,,$$

where $n, n', m, m' \in \mathbb{N}_0$, $w'_2[1] = w'_2[-] = \mathtt{a}$, $w_1 = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^n$, $w_2 = \mathtt{b}^{n'} \cdot w'_2 \cdot \mathtt{b}^m$ and $w_3 = \mathtt{b}^{m'} \cdot \mathtt{a} \cdot w'_3$, and, furthermore,

$$w = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^{n+n'} \cdot w'_2 \cdot \mathtt{b}^{m+m'} \cdot \mathtt{a} \cdot w'_3\,.$$

Obviously, all the even $\mathtt{b}$-segments in the factors $w'_1 \cdot \mathtt{a}$, w'_2 and $\mathtt{a} \cdot w'_3$ also occur in w'. Therefore, it is sufficient to compare the number of even $\mathtt{b}$-segments in the factors $\mathtt{a} \cdot \mathtt{b}^{n+n'} \cdot \mathtt{a}$ and $\mathtt{a} \cdot \mathtt{b}^{m+m'} \cdot \mathtt{a}$ with the number of even $\mathtt{b}$-segments in the factors $\mathtt{a} \cdot \mathtt{b}^n \cdot v \cdot \mathtt{b}^{n'} \cdot \mathtt{a}$ and $\mathtt{a} \cdot \mathtt{b}^m \cdot v \cdot \mathtt{b}^{m'} \cdot \mathtt{a}$.

If $v = \mathtt{b}^k$, $k \in \mathbb{N}_0$, then the $\mathtt{b}$-segment $\mathtt{a} \cdot \mathtt{b}^{n+n'} \cdot \mathtt{a}$ is changed into the $\mathtt{b}$-segment $\mathtt{a} \cdot \mathtt{b}^{n+k+n'} \cdot \mathtt{a}$ and the $\mathtt{b}$-segment $\mathtt{a} \cdot \mathtt{b}^{m+m'} \cdot \mathtt{a}$ is changed into the $\mathtt{b}$-segment $\mathtt{a} \cdot \mathtt{b}^{m+k+m'} \cdot \mathtt{a}$. If k is even, then in w' we have the same number of even $\mathtt{b}$-segments as in w, since $n+k+n'$ is even if and only if $n+n'$ is even, and $m+k+m'$ is even if and only if $m+m'$ is even. If, on the other hand, k is odd, then $n+k+n'$ is even if and only if $n+n'$ is odd, and $m+k+m'$ is even if and only if $m+m'$ is odd. Thus, if $n+n'$ and $m+m'$ are both even or both odd, then the number of even $\mathtt{b}$-segments in w' has decreased (or increased, respectively) by 2 compared to the number of even $\mathtt{b}$-segments in w. If, on the other hand, $n+n'$ is even and $m+m'$ is odd or the other way around, then in w' there are as many even $\mathtt{b}$-segments as in w. So we can conclude that if $v = \mathtt{b}^k$, $k \in \mathbb{N}_0$, then the number of even $\mathtt{b}$-segments in w' is odd.

We shall now assume that there is at least one occurrence of $\mathtt{a}$ in v, i.e., $v = \mathtt{b}^k \cdot u \cdot \mathtt{b}^{k'}$, $k, k' \in \mathbb{N}_0$, where $u[1] = u[-] = \mathtt{a}$. This implies

$$w' = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^{n+k} \cdot u \cdot \mathtt{b}^{k'+n'} \cdot w'_2 \cdot \mathtt{b}^{m+k} \cdot u \cdot \mathtt{b}^{k'+m'} \cdot \mathtt{a} \cdot w'_3\,.$$

In the following we shall show that, for all possible choices of $n, n', m, m', k, k' \in \mathbb{N}_0$, the number of even $\mathtt{b}$-segments among the $\mathtt{b}$-segments $\mathtt{a} \cdot \mathtt{b}^{n+k} \cdot \mathtt{a}$, $\mathtt{a} \cdot \mathtt{b}^{k'+n'} \cdot \mathtt{a}$, $\mathtt{a} \cdot \mathtt{b}^{m+k} \cdot \mathtt{a}$ and $\mathtt{a} \cdot \mathtt{b}^{k'+m'} \cdot \mathtt{a}$ is even if and only if the number of even $\mathtt{b}$-segments among the $\mathtt{b}$-segments $\mathtt{a} \cdot \mathtt{b}^{n+n'} \cdot \mathtt{a}$ and $\mathtt{a} \cdot \mathtt{b}^{m+m'} \cdot \mathtt{a}$ is even. To this end, it is sufficient to note that if $(n+n')$ and $(m+m')$ are both even or both odd, then, for all possible choices of $n, n', m, m', k, k' \in \mathbb{N}_0$, either exactly 0, 2 or all 4 of the numbers $(n+k)$, $(k'+n')$, $(m+k)$ and $(k'+m')$ are even. If, on the other hand, one number of $(n+n')$ and $(m+m')$ is even and the other one is odd, then, for all possible choices of $n, n', m, m', k, k' \in \mathbb{N}_0$, either exactly 1 or 3 of the numbers $(n+k)$, $(k'+n')$, $(m+k)$ and $(k'+m')$ are even. This directly implies that the number of even $\mathtt{b}$-segments in w' is odd, since, by assumption, the number of even $\mathtt{b}$-segments in w is odd.

It remains to consider the case that $w_2 = \mathtt{b}^l$, $l \in \mathbb{N}_0$. We note that this implies the following.

$$w' = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^n \cdot v \cdot \mathtt{b}^l \cdot v \cdot \mathtt{b}^m \cdot \mathtt{a} \cdot w'_3\,,$$

where $n, l, m, \in \mathbb{N}_0$, $w_1 = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^n$, $w_2 = \mathtt{b}^l$ and $w_3 = \mathtt{b}^m \cdot \mathtt{a} \cdot w'_3$, and, furthermore,

$$w = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^{n+l+m} \cdot \mathtt{a} \cdot w'_3\,.$$

If $v = \mathtt{b}^k$, $k \in \mathbb{N}_0$, then $w' = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^{n+k+l+k+m} \cdot \mathtt{a} \cdot w'_3$ and, since $(n+k+l+k+m)$ is even if and only if $(n+l+m)$ is even, we can directly conclude that w' has as many even $\mathtt{b}$-segments as w.

If, on the other hand, $v = \mathtt{b}^k \cdot u \cdot \mathtt{b}^{k'}$, $k, k' \in \mathbb{N}_0$, where $u[1] = u[-] = \mathtt{a}$, then

$$w' = w'_1 \cdot \mathtt{a} \cdot \mathtt{b}^{n+k} \cdot u \cdot \mathtt{b}^{k'+l+k} \cdot u \cdot \mathtt{b}^{k'+m} \cdot \mathtt{a} \cdot w'_3\,.$$

Similarly as before, we can show that, for all possible choices of $n, l, m, k, k' \in \mathbb{N}_0$, the number of even $\mathtt{b}$-segments among the $\mathtt{b}$-segments $\mathtt{a} \cdot \mathtt{b}^{n+k} \cdot \mathtt{a}$, $\mathtt{a} \cdot \mathtt{b}^{k'+l+k} \cdot \mathtt{a}$ and $\mathtt{a} \cdot \mathtt{b}^{k'+m} \cdot \mathtt{a}$ is even if and only if $\mathtt{a} \cdot \mathtt{b}^{n+l+m} \cdot \mathtt{a}$ is an odd $\mathtt{b}$-segment. To this end, it is sufficient to note that if $(n+l+m)$ is even, then, for all possible choices of $n, l, m, k, k' \in \mathbb{N}_0$, either exactly 1 or all 3 of the numbers $(n+k)$, $(k'+l+k)$ and $(k'+m)$ are even. If, on the other hand, $(n+l+m)$ is odd, then, for all possible choices of $n, l, m, k, k' \in \mathbb{N}_0$, either exactly 0 or 2 of the numbers $(n+k)$, $(k'+l+k)$ and $(k'+m)$ are even. This directly implies that the number of even $\mathtt{b}$-segments in w' is odd, since, by assumption, the number of even $\mathtt{b}$-segments in w is odd.

Hence, for all possible choices of w_1, w_2, w_3 and v, w' has an odd number of even $\mathtt{b}$-segments, which concludes the proof. □ (*Claim* (1))

We are now ready to prove the statement of the lemma, i. e., for every $w \in L_{\mathrm{E},\Sigma_2}(\mathtt{a} \cdot \alpha \cdot \mathtt{a})$, w contains an even $\mathtt{b}$-segment. Let h be a substitution with $h(\mathtt{a} \cdot \alpha \cdot \mathtt{a}) = w$. Obviously, if, for some $x \in \mathrm{var}(\alpha)$, $h(x)$ contains an even $\mathtt{b}$-segment, then $h(\mathtt{a} \cdot \alpha \cdot \mathtt{a})$ contains an even $\mathtt{b}$-segment. Consequently, we only have to consider the case that, for every $x \in \mathrm{var}(\alpha)$, $h(x)$ does not contain an even $\mathtt{b}$-segment.

We can note that there are words $u_1, u_2, \ldots, u_k$, such that $u_1 = \mathtt{a} \cdot \mathtt{a}$, $u_k = h(\mathtt{a} \cdot \alpha \cdot \mathtt{a})$ and, for every i, $2 \leq i \leq k$, the word u_i can be obtained by inserting two occurrences of a word v into the word u_{i-1}. More precisely, we start with $u_1 = \mathtt{a} \cdot \mathtt{a}$ and insert two occurrences of $h(x_1)$ into u_1 in order to obtain u_2, then we repeat this step in order to construct u_3 and after $\frac{|\beta|_{x_1}}{2}$ such steps we stop. Next, we do the same for $\frac{|\beta|_{x_2}}{2}$ steps with respect to $h(x_2)$ and so on. Clearly, since, for every $x \in \operatorname{var}(\alpha)$, $|\alpha|_x$ is even, this can be done in such a way that $u_k = h(\mathtt{a} \cdot \alpha \cdot \mathtt{a})$ is satisfied. Furthermore, since u_1 has an odd number of even $\mathtt{b}$-segments, we can conclude with the above claim that, for every i, $1 \leq i \leq k$, the word u_i has an odd number of even $\mathtt{b}$-segments, which implies that $u_k = h(\mathtt{a} \cdot \alpha \cdot \mathtt{a}) = w$ has at least one even $\mathtt{b}$-segments. This concludes the proof. □

By applying Lemma 6.12, we can show that if a pattern $\alpha := \beta \cdot y \cdot \beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{a} \cdot \delta' \cdot z \cdot \delta$ satisfies the conditions of Lemma 6.11, all variables in γ have an even number of occurrences and there is at least one variable in γ that occurs only twice, then the factor $y \cdot \beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{a} \cdot \delta' \cdot z$ can be substituted by a regular expression.

Lemma 6.13. *Let* $\alpha := \beta \cdot y \cdot \beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{a} \cdot \delta' \cdot z \cdot \delta$*, where* $\beta, \delta \in (\Sigma_2 \cup X)^*$*,* $\beta', \gamma, \delta' \in X^*$*,* $y, z \in X$*,* $|\alpha|_y = |\alpha|_z = 1$ *and, for every* $x \in \operatorname{var}(\gamma)$*,* $|\gamma|_x$ *is even. Then* $L_{\mathrm{E},\Sigma_2}(\alpha) \subseteq L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{a}(\mathtt{bb})^*\mathtt{a} \cdot z \cdot \delta)$*. If, furthermore,* $\operatorname{var}(\beta' \cdot \gamma \cdot \delta') \cap \operatorname{var}(\beta \cdot \delta) = \emptyset$ *and there exists a* $z' \in \operatorname{var}(\gamma)$ *with* $|\alpha|_{z'} = 2$*, then also* $L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{a}(\mathtt{bb})^*\mathtt{a} \cdot z \cdot \delta) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha)$*.*

Proof. Let h be an arbitrary substitution. We first note that we can prove $h(\alpha) \in L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{a} \cdot (\mathtt{bb})^* \cdot \mathtt{a} \cdot z \cdot \delta)$ by showing that $h(y \cdot \beta' \cdot \mathtt{a} \cdot \gamma \cdot \mathtt{a} \cdot \delta' \cdot z)$ contains a factor of form $\mathtt{a} \cdot \mathtt{b}^n \cdot \mathtt{a}$, where n is even. We note that Lemma 6.12 directly implies that $h(\mathtt{a} \cdot \gamma \cdot \mathtt{a})$ contains such a factor. Thus, $L_{\mathrm{E},\Sigma_2}(\alpha) \subseteq L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{a} \cdot (\mathtt{bb})^* \cdot \mathtt{a} \cdot z \cdot \delta)$ follows.

In order to prove the second statement of the lemma, we assume that $\operatorname{var}(\beta' \cdot \gamma \cdot \delta') \cap \operatorname{var}(\beta \cdot \delta) = \emptyset$ and there exists a $z' \in (\operatorname{var}(\gamma) \setminus \operatorname{var}(\beta \cdot \beta' \cdot \delta' \cdot \delta))$ with $|\gamma|_{z'} = 2$. Now let h be an arbitrary substitution and let $h(\beta \cdot y \cdot \mathtt{a} \cdot (\mathtt{bb})^* \cdot \mathtt{a} \cdot z \cdot \delta) = h(\beta \cdot y) \cdot \mathtt{a} \cdot \mathtt{b}^{2n} \cdot \mathtt{a} \cdot h(z \cdot \delta)$, $n \in \mathbb{N}_0$. Obviously, $h(\beta \cdot y \cdot \mathtt{a} \cdot (\mathtt{bb})^* \cdot \mathtt{a} \cdot z \cdot \delta) = g(\alpha)$, where, for every $x \in (\operatorname{var}(\beta \cdot \delta) \cup \{y, z\})$, $g(x) := h(x)$, for every $x \in \operatorname{var}(\beta' \cdot \gamma \cdot \delta') \setminus \{z'\}$, $g(x) := \varepsilon$ and $g(z') := \mathtt{b}^n$. This implies $L_{\mathrm{E},\Sigma_2}(\beta \cdot y \cdot \mathtt{a} \cdot (\mathtt{bb})^* \cdot \mathtt{a} \cdot z \cdot \delta) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha)$, which concludes the proof. □

Obviously, Lemmas 6.11 and 6.13 can also be applied in any order in the iterative way pointed out above with respect to Lemma 6.11. We shall illustrate this now in a more general way. Let α be an arbitrary pattern such that

$$\alpha := \beta \cdot y_1 \cdot \beta'_1 \cdot \mathtt{a} \cdot \gamma_1 \cdot \mathtt{a} \cdot \delta'_1 \cdot z_1 \cdot \pi \cdot y_2 \cdot \beta'_2 \cdot \mathtt{b} \cdot \gamma_2 \cdot \mathtt{a} \cdot \delta'_2 \cdot z_2 \cdot \delta \,,$$

with $\beta, \pi, \delta \in (\Sigma_2 \cup X)^*$, $\beta'_1, \beta'_2, \gamma_1, \gamma_2, \delta'_1, \delta'_2 \in X^*$ and $y_1, y_2, z_1, z_2 \in X$. If the factors $y_1 \cdot \beta'_1 \cdot \mathtt{a} \cdot \gamma_1 \cdot \mathtt{a} \cdot \delta'_1 \cdot z_1$ and $y_2 \cdot \beta'_2 \cdot \mathtt{b} \cdot \gamma_2 \cdot \mathtt{a} \cdot \delta'_2 \cdot z_2$ satisfy the conditions of Lemma 6.13 and Lemma 6.11, respectively, then we can conclude that α is equivalent to $\alpha' := \beta \cdot y_1 \cdot \mathtt{a}(\mathtt{bb})^*\mathtt{a} \cdot z_1 \cdot \pi \cdot y_2 \cdot \mathtt{ba} \cdot z_2 \cdot \delta$. This particularly means that the rather strong conditions

1. $\mathrm{var}(\beta'_1 \cdot \gamma_1 \cdot \delta'_1) \cap \mathrm{var}(\beta \cdot \pi \cdot \beta'_2 \cdot \gamma_2 \cdot \delta'_2 \cdot \delta) = \emptyset$,
2. $\mathrm{var}(\beta'_2 \cdot \gamma_2 \cdot \delta'_2) \cap \mathrm{var}(\beta \cdot \beta'_1 \cdot \gamma_1 \cdot \delta'_1 \cdot \pi \cdot \delta) = \emptyset$

must be satisfied. However, we can state that $L_{\mathrm{E},\Sigma_2}(\alpha) = L_{\mathrm{E},\Sigma_2}(\alpha')$ still holds if instead of conditions 1 and 2 from above the weaker condition $\mathrm{var}(\beta'_1 \cdot \gamma_1 \cdot \delta'_1 \cdot \beta'_2 \cdot \gamma_2 \cdot \delta'_2) \cap \mathrm{var}(\beta \cdot \pi \cdot \delta) = \emptyset$ is satisfied. This claim can be easily proved by applying the same argumentations as in the proofs of Lemmas 6.11 and 6.13, and we can extend this result to arbitrarily many factors of form $y_i \cdot \beta'_i \cdot c_1 \cdot \gamma_i \cdot c_2 \cdot \delta'_i \cdot z_i$, $c_1, c_2 \in \Sigma_2$. Next, by the following definition, we formalise this observation in terms of a relation on patterns with regular expressions.

Definition 6.14. For any two patterns with regular expressions α and α', we write $\alpha \rhd \alpha'$ if and only if the following conditions are satisfied.

- α contains factors $\alpha_i \in (\Sigma_2 \cup X)^*$, $1 \leq i \leq k$, where, for every i, $1 \leq i \leq k$, $\alpha_i := y_i \cdot \beta'_i \cdot d_i \cdot \gamma_i \cdot d'_i \cdot \delta'_i \cdot z_i$, with $\beta'_i, \gamma_i, \delta'_i \in X^+$, $y_i, z_i \in X$, $|\alpha|_{y_i} = |\alpha|_{z_i} = 1$, $d_i, d'_i \in \Sigma_2$ and, if $d_i = d'_i$, then, for every $x \in \mathrm{var}(\gamma_i)$, $|\gamma_i|_x$ is even and there exists an $x' \in \mathrm{var}(\gamma_i)$ with $|\alpha|_{x'} = 2$. Furthermore, the factors $\alpha_1, \alpha_2, \ldots, \alpha_k$ can overlap by at most one symbol and the variables in the factors $\alpha_1, \alpha_2, \ldots, \alpha_k$ occur exclusively in these factors.
- α' is obtained from α by substituting every α_i, $1 \leq i \leq k$, by $y_i \cdot d_i d'_i \cdot z_i$, if $d_i \neq d'_i$ and by $y_i \cdot d_i(d''_i d''_i)^* d'_i \cdot z_i$, $d''_i \in \Sigma$, $d''_i \neq d_i$, if $d_i = d'_i$.

By generalising Lemmas 6.11 and 6.13, we can prove that $\alpha \rhd \alpha'$ implies that α and α' describe the same E-pattern language with respect to alphabet Σ_2.

Theorem 6.15. *Let α and α' be patterns with regular expressions. If $\alpha \rhd \alpha'$, then $L_{\mathrm{E},\Sigma_2}(\alpha) = L_{\mathrm{E},\Sigma_2}(\alpha')$.*

Proof. We assume that $\alpha \rhd \alpha'$ is satisfied, which implies that α contains factors $\alpha_i \in (\Sigma_2 \cup X)^*$, $1 \leq i \leq k$, where, for every i, $1 \leq i \leq k$, $\alpha_i := y_i \cdot \beta'_i \cdot d_i \cdot \gamma_i \cdot d'_i \cdot \delta'_i \cdot z_i$, with $\beta'_i, \gamma_i, \delta'_i \in X^+$, $y_i, z_i \in X$, $|\alpha|_{y_i} = |\alpha|_{z_i} = 1$, $d_i, d'_i \in \Sigma_2$ and, if $d_i = d'_i$, then, for every $x \in \mathrm{var}(\gamma_i)$, $|\gamma_i|_x$ is even and there exists an $x' \in \mathrm{var}(\gamma_i)$ with $|\alpha|_{x'} = 2$. Furthermore, the factors $\alpha_1, \alpha_2, \ldots, \alpha_k$ can overlap by at most one symbol and the variables in the factors $\alpha_1, \alpha_2, \ldots, \alpha_k$ occur exclusively in these factors. Moreover,

α' is obtained from α by substituting every α_i, $1 \leq i \leq k$, by $\alpha'_i := y_i \cdot d_i \cdot d'_i \cdot z_i$, if $d_i \neq d'_i$ and by $\alpha'_i := y_i \cdot d_i \cdot (d''_i d''_i)^* \cdot d'_i \cdot z_i$, $d''_i \neq d_i$, if $d_i = d'_i$.

By Lemmas 6.11 and 6.13, we can conclude that $L_{\mathrm{E},\Sigma_2}(\alpha) \subseteq L_{\mathrm{E},\Sigma_2}(\pi_1)$, where π_1 is obtained from α by substituting α_1 by α'_1. In the same way, we can also conclude that $L_{\mathrm{E},\Sigma_2}(\pi_1) \subseteq L_{\mathrm{E},\Sigma_2}(\pi_2)$, where π_2 obtained from π_1 by substituting α_2 by α'_2. By repeating this argumentation, $L_{\mathrm{E},\Sigma_2}(\alpha) \subseteq L_{\mathrm{E},\Sigma_2}(\alpha')$ follows.

It remains to prove that $L_{\mathrm{E},\Sigma_2}(\alpha') \subseteq L_{\mathrm{E},\Sigma_2}(\alpha)$. To this end, let h be an arbitrary substitution. We shall show that $h(\alpha') \in L_{\mathrm{E},\Sigma_2}(\alpha)$ by defining another substitution g that satisfies $h(\alpha') = g(\alpha)$. First, let $A \subseteq \{1, 2, \ldots, k\}$ be such that, for every i, $1 \leq i \leq k$, $d_i = d'_i$ if and only if $i \in A$. Moreover, for every $i \in A$, let x_i be a variable that satisfies $x_i \in \operatorname{var}(\gamma_i)$ with $|\alpha|_{x_i} = 2$. Now, for every $x \in \operatorname{var}(\alpha) \setminus (\bigcup_{i=1}^{k} \operatorname{var}(\beta'_i \cdot \gamma_i \cdot \delta'_i))$, we define $g(x) := h(x)$. For every $x \in (\bigcup \operatorname{var}(\beta'_i \cdot \gamma_i \cdot \delta'_i) \setminus \{x_i \mid i \in A\})$, we define $g(x) := \varepsilon$. So it only remains to define $g(x_i)$, for every $x_i \in A$. To this end, we first note that, for every $i \in A$, $\alpha'_i = y_i \cdot d_i \cdot (d''_i d''_i)^* \cdot d'_i \cdot z_i$. Now, for every $i \in A$, let $n_i \in \mathbb{N}_0$ be such that h maps $(d''_i d''_i)^*$ to $(d''_i)^{n_i}$. Finally, for every $i \in A$, we define $g(x_i) := (d''_i)^{n_i}$. It can be easily verified that $g(\alpha) = h(\alpha')$. Thus, $L_{\mathrm{E},\Sigma_2}(\alpha') \subseteq L_{\mathrm{E},\Sigma_2}(\alpha)$, which concludes the proof. $\square$

We conclude this section by discussing a more complex example that illustrates how Definition 6.14 and Theorem 6.15 constitute a sufficient condition for the regularity of the E-pattern language of a pattern with respect to Σ_2. Let α be the following pattern.

$$\underbrace{x_1\mathtt{a}x_2x_3^2\mathtt{b}x_4x_3x_5x_6}_{\alpha_1 := y_1 \cdot \beta'_1 \cdot \mathtt{a} \cdot \gamma_1 \cdot \mathtt{b} \cdot \delta'_1 \cdot z_1}\, x_7^2\, \underbrace{x_8x_9x_5x_3\mathtt{a}x_4x_5x_4x_9x_{10}\mathtt{b}x_{11}}_{\alpha_2 := y_2 \cdot \beta'_2 \cdot \mathtt{a} \cdot \gamma_2 \cdot \mathtt{b} \cdot \delta'_2 \cdot z_2}\, \mathtt{a}x_{12}\mathtt{b}x_{13}\mathtt{a}\, \underbrace{x_{14}x_{15}\mathtt{b}x_{15}^2x_{16}^2\mathtt{b}x_{17}}_{\alpha_3 := y_3 \cdot \beta'_3 \cdot \mathtt{a} \cdot \gamma_3 \cdot \mathtt{b} \cdot \delta'_3 \cdot z_3}\,.$$

By Definition 6.14, $\alpha \rhd \beta$ holds, where β is obtained from α by substituting the above defined factors α_1, α_2 and α_3 by factors $x_1 \cdot \mathtt{ab} \cdot x_6$, $x_8 \cdot \mathtt{ab} \cdot x_{11}$ and $x_{14} \cdot \mathtt{b}(\mathtt{aa})^*\mathtt{b} \cdot x_{17}$, respectively, i. e.,

$$\beta := x_1\mathtt{ab}x_6x_7x_7x_8\mathtt{ab}x_{11}\mathtt{a}x_{12}\mathtt{b}x_{13}\mathtt{a}x_{14}\mathtt{b}(\mathtt{aa})^*\mathtt{b}x_{17}\,.$$

Furthermore, by Theorem 6.15, we can conclude that $L_{\mathrm{E},\Sigma_2}(\alpha) = L_{\mathrm{E},\Sigma_2}(\beta)$. However, we can also apply the same argumentation to different factors of α, as pointed out below:

$$x_1\mathtt{a}\, \underbrace{x_2x_3^2\mathtt{b}x_4x_3x_5x_6x_7^2x_8x_9x_5x_3\mathtt{a}x_4x_5x_4x_9x_{10}}_{\alpha_1 := y_1 \cdot \beta'_1 \cdot \mathtt{a} \cdot \gamma_1 \cdot \mathtt{b} \cdot \delta'_1 \cdot z_1}\, \mathtt{b}x_{11}\mathtt{a}x_{12}\mathtt{b}x_{13}\mathtt{a}\, \underbrace{x_{14}x_{15}\mathtt{b}x_{15}^2x_{16}^2\mathtt{b}x_{17}}_{\alpha_2 := y_2 \cdot \beta'_2 \cdot \mathtt{a} \cdot \gamma_2 \cdot \mathtt{b} \cdot \delta'_2 \cdot z_2}\,.$$

Now, again by Definition 6.14, $\alpha \rhd \beta'$ is satisfied, where

$$\beta' := x_1 \mathtt{a} x_2 \mathtt{ba} x_{10} \mathtt{b} x_{11} \mathtt{a} x_{12} \mathtt{b} x_{13} \mathtt{a} x_{14} \mathtt{b} (\mathtt{aa})^* \mathtt{b} x_{17} \,.$$

Since every variable of β' has only one occurrence, it can be easily seen that $L_{\mathrm{E},\Sigma_2}(\beta') \in \mathrm{REG}$ and, by Theorem 6.15, $L_{\mathrm{E},\Sigma_2}(\alpha) \in \mathrm{REG}$ follows.

The above example demonstrates how the relation $\rhd$ and Theorem 6.15 can be used in order to show that a pattern describes a regular E-pattern language. Hence, in order to solve the membership problem for a pattern language $L_{\mathrm{E},\Sigma_2}(\alpha)$, it might be worthwhile to first check whether or not $\alpha \rhd \beta$ for some pattern β with regular expressions and every variable of β has only one occurrence. If this is the case, then we can conclude that $L_{\mathrm{E},\Sigma_2}(\alpha)$ is a regular language, which allows the membership problem for $L_{\mathrm{E},\Sigma_2}(\alpha)$ to be solved faster. If, on the other hand, this is not the case, then we cannot conclude that $L_{\mathrm{E},\Sigma_2}(\alpha)$ is not regular, but we might still be able to find a pattern β with $\alpha \rhd \beta$ and β has much fewer variables than α (as it is the case in the example above), which again helps to solve the membership problem for $L_{\mathrm{E},\Sigma_2}(\alpha)$.

Chapter 7

Beyond Pattern Languages

As mentioned in Section 2.2.2, due to their simple definition, pattern languages have connections to many areas of theoretical computer science. In particular, there exist numerous language generating devices that also use the most fundamental mechanism of patterns, i. e., the homomorphic substitution of symbols, as a basic element. A prominent example for such language generators are the well-known L systems (see Kari et al. [43] for a survey), but also many types of grammars as, e. g., Wijngaarden grammars, macro grammars, Indian parallel grammars or deterministic iteration grammars, use homomorphic substitution as a central concept (cf. Albert and Wegner [2] and Bordihn et al. [9] and the references therein). Albert and Wegner [2] introduce H-systems, which use homomorphic substitution in a more puristic way, without any grammar like mechanisms. A language generating device of practical importance, which can be easily seen to be related to pattern languages and which has already been briefly described in Section 2.2.3.1, are the extended regular expressions with backreferences (denoted by REGEX). More recent models like pattern expressions (Câmpeanu and Yu [12]), synchronized regular expressions (Della Penna et al. [59]) and EH-expressions (Bordihn et al. [9]) are mainly inspired directly by REGEX.

Compared to most of the above devices, pattern languages use the concept of homomorphic substitution in a rather basic way. Hence, the question arises whether insights into pattern languages can be extended to other, more general language generating devices. For negative results of pattern languages as, e. g., the NP-hardness of their membership problem or the undecidability of their inclusion problem, this is straightforward: every language generating device the corresponding language class of which contains the full class of pattern languages as, e. g., REGEX, shares these negative results. On the other hand, if we want to apply the proof techniques of Chapters 3 and 5 in order to identify parameters of REGEX that, if restricted, allow the membership problem for REGEX to be solved efficiently, then a deeper understanding of the role that homomorphic substitutions play for REGEX is required. We shall now point out that for REGEX, it is surprisingly difficult to gain insights in this regard.

To this end, we recall that, intuitively speaking, a backreference in a REGEX points back to an earlier subexpression, meaning that it has to be matched to the same word the earlier subexpression has been matched to. For example, $r := (_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1 \cdot \mathtt{c} \cdot \backslash 1$ is a REGEX, where $\backslash 1$ is a *backreference* to the *referenced subexpression* between the parentheses $(_1$ and $)_1$. The language described by r, denoted by $\mathcal{L}(r)$, is the set of all words $w\mathtt{c}w$, $w \in \{\mathtt{a}, \mathtt{b}\}^*$.

From an intuitive point of view, REGEX are a combination of the concept of homomorphic substitution and regular expressions. For example, the REGEX r can also be given as a string $x\mathtt{c}x$, where the symbol x can be substituted by words from $\{\mathtt{a}, \mathtt{b}\}^*$, i. e., both occurrences of x must be substituted by the same word. However, due to the possible nesting of referenced subexpressions, the concepts of regular expressions and substitutions seem to be inherently entangled and there is no easy way to treat them separately. We illustrate this with the example $t := (_1\ \mathtt{a}^*\ )_1 \cdot (_2\ (\mathtt{b} \cdot \backslash 1)^*\ )_2 \cdot \backslash 2 \cdot \backslash 1$. The language $\mathcal{L}(t) := \{\mathtt{a}^n (\mathtt{b}\mathtt{a}^n)^m (\mathtt{b}\mathtt{a}^n)^m \mathtt{a}^n \mid n, m \geq 0\}$ cannot that easily be described in terms of a single string and substitutions, e. g., by the string $xyyx$, where x can be substituted by words from $\{\mathtt{a}^n \mid n \geq 0\}$, and y by words of form $\{(\mathtt{b}\mathtt{a}^n)^m \mid n, m \geq 0\}$, since then we can obtain words $\mathtt{a}^n(\mathtt{b}\mathtt{a}^{n'})^m(\mathtt{b}\mathtt{a}^{n'})^m\mathtt{a}^n$ with $n \neq n'$. In fact, two separate steps of substitution seem necessary, i. e., we first substitute y by words from $\{(\mathtt{b}z)^n \mid n \geq 0\}$ and after that we substitute x and z by words from $\{\mathtt{a}^n \mid n \geq 0\}$, with the additional requirement that x and z are substituted by the same word. More intuitively speaking, the nesting of referenced subexpressions require *iterated* homomorphic substitution, but we also need to carry on information from one step of homomorphic substitution to the next one.

The above considerations indicate that in REGEX there can be complex interdependencies between the concepts of regular expressions and homomorphic substitution. Therefore, in order to extend results about the complexity of the membership problem for pattern languages to the membership problem for REGEX languages, it might be necessary to further research this issue. To this end, we study alternative possibilities to combine regular expressions and homomorphic substitutions, with the objective of reaching the expressive power of REGEX as closely as possible, without exceeding it. More precisely, we combine patterns with regular expressions by first adding the alternation and star operator to patterns and, furthermore, by letting their variables be typed by regular languages, i. e., the words variables are replaced with are from given regular sets. Then we iterate this step by using this new class of languages again as types for variables and so on.

We also take a closer look at *pattern expressions*, which are introduced by Câmpeanu and Yu [12] as a convenient tool to define REGEX languages. In [12], many examples are provided that show how to translate a REGEX into an equivalent pattern expression and vice versa. It is also stated that this is possible in general, but a formal proof for this statement is not provided (in fact, from the following Theorem 7.13 and 7.17 it follows that there are indeed REGEX that describe languages that cannot be described by pattern expressions). In the present chapter, we show that pattern expressions are much weaker than REGEX and they describe a proper subset of the class of REGEX languages (in fact, they are even weaker than REGEX that do not contain referenced subexpressions under a star). These limits in expressive power are caused by the above described difficulties due to the nesting of referenced subexpressions.

On the other hand, pattern expressions still describe an important and natural subclass of REGEX languages, that has been independently defined in terms of other models and, as shown in this work, also coincides with the class of languages resulting from the modification of patterns described above. We then refine the way of how pattern expressions define languages in order to accommodate the nesting of referenced subexpressions and we show that the thus obtained class of languages coincides with the class of languages given by REGEX that do not contain a referenced subexpression under a star.

Finally, we briefly discuss the membership problem for REGEX with a restricted number of backreferences, which, in the unrestricted case, is NP-complete. Although it seems trivial that this problem can be solved in polynomial time, the situation is complicated by subexpressions that occur and are referenced under a star, which represent arbitrarily many distinct subexpressions with individual backreferences.

7.1 Patterns with Regular Operators and Types

In this section, we combine pattern languages with regular languages and regular expressions. More precisely, we first define pattern languages, the variables of which are typed by regular languages[1] and after that we add the regular operators of alternation and star.

Let PAT $:= \{\alpha \mid \alpha \in (\Sigma \cup X)^+\}$. We always assume that, for every $i \in$

[1]We recall that at the end of Section 3.3.2, we already use regular-typed patterns and in Section 6.3, we use regular expressions in patterns, which, technically, are regular-typed patterns, too. Furthermore, the learnability of patterns with types has been investigated by Wright [87] and Koshiba [46].

$\mathbb{N}$, $x_i \in \mathrm{var}(\alpha)$ implies $\{x_1, x_2, \ldots, x_{i-1}\} \subseteq \mathrm{var}(\alpha)$. For an arbitrary class $\mathfrak{L}$ of languages and a pattern α with $|\mathrm{var}(\alpha)| = m$, an *$\mathfrak{L}$-type for α* is a tuple $\mathcal{T} := (T_{x_1}, T_{x_2}, \ldots, T_{x_m})$, where, for every i, $1 \leq i \leq m$, $T_{x_i} \in \mathfrak{L}$ and T_{x_i} is called the *type language of (variable) x_i*. A substitution h *satisfies* $\mathcal{T}$ if and only if, for every i, $1 \leq i \leq m$, $h(x_i) \in T_{x_i}$.

Definition 7.1. Let $\alpha \in \mathrm{PAT}$, let $\mathfrak{L}$ be a class of languages and let $\mathcal{T}$ be an $\mathfrak{L}$-type for α. The *$\mathcal{T}$-typed pattern language of α* is defined by $\mathcal{L}_{\mathcal{T}}(\alpha) := \{h(\alpha) \mid h \text{ is a substitution that satisfies } \mathcal{T}\}$. For any class of languages $\mathfrak{L}$, $\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT}) := \{\mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \in \mathrm{PAT}, \mathcal{T} \text{ is an } \mathfrak{L}\text{-type for } \alpha\}$ is the *class of $\mathfrak{L}$-typed pattern languages.*

We note that $\{\Sigma^*\}$-typed and $\{\Sigma^+\}$-typed pattern languages correspond to the classes of E-pattern languages and NE-pattern languages, respectively, as defined in Chapter 2. It is easy to see that $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$ is contained in the class of REGEX languages. The substantial difference between these two classes is that the backreferences of a REGEX can refer to subexpressions that are again REGEX and, thus, may describe non-regular languages, while REG-typed pattern languages are given by patterns all the variables of which are typed by regular languages. Hence, in order to increase the expressive power of typed patterns in this regard, it seems necessary to type the variables with languages from $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$ instead of REG and then using the thus obtained languages again as type languages and so on. However, as demonstrated by the following proposition, this approach leads to a dead end:

Proposition 7.2. *For any class of languages $\mathfrak{L}$, $\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT}) = \mathcal{L}_{\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT})}(\mathrm{PAT})$.*

Proof. Let $\mathfrak{L}' := \mathcal{L}_{\mathfrak{L}}(\mathrm{PAT})$. We first show that $\mathfrak{L}' \subseteq \mathcal{L}_{\mathfrak{L}'}(\mathrm{PAT})$. To this end, let $L \in \mathfrak{L}'$. Obviously, $L = \mathcal{L}_{(L)}(x_1)$, where (L) is an $\mathfrak{L}'$-type for the pattern x_1. Thus, $L \in \mathcal{L}_{\mathfrak{L}'}(\mathrm{PAT})$ and $\mathfrak{L}' \subseteq \mathcal{L}_{\mathfrak{L}'}(\mathrm{PAT})$ follows.

To prove $\mathcal{L}_{\mathfrak{L}'}(\mathrm{PAT}) \subseteq \mathfrak{L}'$, we let $L' \in \mathcal{L}_{\mathfrak{L}'}(\mathrm{PAT})$. This implies that there exists a pattern β and an $\mathfrak{L}'$-type $\mathcal{T}' := (T_{x_1}, T_{x_2}, \ldots, T_{x_m})$ for β with $\mathcal{L}_{\mathcal{T}'}(\beta) = L'$. Furthermore, since $\mathcal{T}'$ is an $\mathfrak{L}'$-type, for every i, $1 \leq i \leq m$, there exists a pattern α_i and an $\mathfrak{L}$-type $\mathcal{T}'_i$ such that $\mathcal{L}_{\mathcal{T}'_i}(\alpha_i) = T_{x_i}$. Now, for every i, $1 \leq i \leq m$, we transform α_i into α'_i by uniformly renaming the variables in α_i in such a way that, for every i, j with $1 \leq i < j \leq m$, $\mathrm{var}(\alpha'_i) \cap \mathrm{var}(\alpha'_j) = \emptyset$, and $\bigcup_{1 \leq i \leq m} \mathrm{var}(\alpha'_i) = \{x_1, x_2, \ldots, x_k\}$, where $k = \sum_{i=1}^{m} |\mathrm{var}(\alpha_i)|$. We can now obtain a pattern α from β by substituting every occurrence of x_i in β by α_i, $1 \leq i \leq m$. We note that $\mathcal{L}_{\widehat{\mathcal{T}}}(\alpha) = \mathcal{L}_{\mathcal{T}'}(\beta)$, where $\widehat{\mathcal{T}}$ is an $\mathfrak{L}$-type for α that is constructed by combining all the $\mathfrak{L}$-types $\mathcal{T}'_i$, $1 \leq i \leq m$. This implies that $L' \in \mathfrak{L}'$ and, thus, $\mathcal{L}_{\mathfrak{L}'}(\mathrm{PAT}) \subseteq \mathfrak{L}'$, which concludes the proof. □

Proposition 7.2 demonstrates that typed pattern languages are invariant with respect to iteratively typing the variables of the patterns. This suggests that if we want to extend pattern languages in such a way that they can describe larger subclasses of the class of REGEX languages, then the regular aspect cannot completely be limited to the type languages of the variables. This observation brings us to the definition of $\mathrm{PAT}_{\mathrm{ro}} := \{\alpha \mid \alpha$ is a regular expression over $(\Sigma \cup X')$, where X' is a finite subset of $X\}$, the set of *patterns with regular operators.* For the sake of convenience, in the remainder of this chapter, whenever we use a regular expression over the alphabet $(\Sigma \cup X)$, we actually mean a regular expression over $(\Sigma \cup X')$, for some finite subset X' of X. In order to define the language given by a pattern with regular operators, we extend the definition of types to patterns with regular operators in the obvious way.

Definition 7.3. Let $\alpha \in \mathrm{PAT}_{\mathrm{ro}}$ and let $\mathcal{T}$ be a type for α. The *$\mathcal{T}$-typed pattern language of* α is defined by $\mathcal{L}_{\mathcal{T}}(\alpha) := \bigcup_{\beta \in \mathcal{L}(\alpha)} \mathcal{L}_{\mathcal{T}}(\beta)$. For any class of languages $\mathfrak{L}$, we define $\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT}_{\mathrm{ro}}) := \{\mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \in \mathrm{PAT}_{\mathrm{ro}}, \mathcal{T}$ is an $\mathfrak{L}$-type for $\alpha\}$.

Patterns with regular operators are also used in the definition of pattern expressions (see Câmpeanu and Yu [12] and Section 7.2) and have been called *regular patterns* by Bordihn et al. in [9]. As an example, we define $\alpha := (x_1\mathtt{a}x_1 \mid x_2\mathtt{b}x_2)^* \in \mathrm{PAT}_{\mathrm{ro}}$ and $\mathcal{T} := (\mathcal{L}(\mathtt{c}^*), \mathcal{L}(\mathtt{d}^*))$. The language $\mathcal{L}_{\mathcal{T}}(\alpha)$ can be generated in two steps. We first construct $\mathcal{L}(\alpha) = \{\beta_1 \cdot \beta_2 \cdot \dots \cdot \beta_n \mid n \in \mathbb{N}_0, \beta_i \in \{x_1\mathtt{a}x_1, x_2\mathtt{b}x_2\}, 1 \leq i \leq n\}$ and then $\mathcal{L}_{\mathcal{T}}(\alpha)$ is the union of all typed pattern languages $\mathcal{L}_{\mathcal{T}}(\beta)$, where $\beta \in \mathcal{L}(\alpha)$. Thus, $\mathcal{L}_{\mathcal{T}}(\alpha) = \{w_1 \cdot w_2 \cdot \dots \cdot w_n \mid n \in \mathbb{N}_0, w_i \in \{\mathtt{c}^m\mathtt{a}\mathtt{c}^m, \mathtt{d}^m\mathtt{b}\mathtt{d}^m \mid m \in \mathbb{N}_0\}, 1 \leq i \leq n\}$.

It seems reasonable to assume that REG-typed patterns with regular operators are strictly more powerful than REG-typed patterns without regular operators. In the following proposition, we formally prove this intuition.

Proposition 7.4. $\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}_{\mathrm{ro}})$.

Proof. The inclusions follow from the definitions and we only have to show that they are proper. We can first note that since there are regular languages not in $\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT})$, e. g., all finite languages with cardinality at least 2, and $\mathrm{REG} \subseteq \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$, we can conclude $\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$.

It remains to show that there exists a language that is in $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}_{\mathrm{ro}})$, but not in $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$. To this end, we define $\alpha := (x_1 \cdot \mathtt{c} \cdot x_1 \mid \varepsilon) \in \mathrm{PAT}_{\mathrm{ro}}$ and $\mathcal{T} := (\mathtt{a}^+)$. Clearly, $\mathcal{L}_{\mathcal{T}}(\alpha) = \{\mathtt{a}^n \cdot \mathtt{c} \cdot \mathtt{a}^n \mid n \in \mathbb{N}\} \cup \{\varepsilon\}$ and $\mathcal{L}_{\mathcal{T}}(\alpha) \in \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}_{\mathrm{ro}})$. We shall now prove that $\mathcal{L}_{\mathcal{T}}(\alpha) \notin \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$. To this end, we assume to the contrary that there exists a pattern β and a REG-type $\mathcal{T}_r := (T_{x_1}, T_{x_2}, \dots, T_{x_m})$ for β such that $\mathcal{L}_{\mathcal{T}_r}(\beta) = \mathcal{L}_{\mathcal{T}}(\alpha)$. Without loss of generality, we can assume that, for every

i, $1 \leq i \leq m$, $T_{x_i} \neq \{\varepsilon\}$. We note that since $\varepsilon \in \mathcal{L}_{\mathcal{T}_r}(\beta)$, for every i, $1 \leq i \leq m$, $\varepsilon \in T_{x_i}$ and $\beta \in X^+$. We can further note that if there exists an i, $1 \leq i \leq m$, such that T_{x_i} contains a non-empty word u without an occurrence of $\mathtt{c}$, then we can produce a non-empty word w without any occurrence of $\mathtt{c}$ by substituting every variable in β by ε except x_i, which is substituted by u. Since such a word is not in $\mathcal{L}_{\mathcal{T}_r}(\beta)$, this is a contradiction and we can assume that, for every i, $1 \leq i \leq m$, and for every non-empty $u \in T_{x_i}$, there is exactly one occurrence of $\mathtt{c}$ in u. This implies that if there is more than one occurrence of a variable in β, then we can produce a word with at least two occurrences of $\mathtt{c}$. Thus, $\beta = x_1$ holds. Now, since we assume $\mathcal{L}_{\mathcal{T}_r}(\beta) = \mathcal{L}_{\mathcal{T}}(\alpha)$, it follows that $T_{x_1} = \mathcal{L}_{\mathcal{T}}(\alpha)$, which is a contradiction, since $\mathcal{L}_{\mathcal{T}}(\alpha)$ is not a regular language, but T_{x_1} is. □

The invariance of typed patterns – represented by Proposition 7.2 – does not hold anymore with respect to patterns with regular operators. Before we formally prove this claim, we shall define an infinite hierarchy of classes of languages given by typed patterns with regular operators. The bottom of this hierarchy are the REG-typed pattern languages with regular operators. Each level of the hierarchy is then given by patterns with regular operators that are typed by languages from the previous level of the hierarchy and so on.

Definition 7.5. Let $\mathfrak{L}_{\mathrm{ro},0} := \mathrm{REG}$ and, for every $i \in \mathbb{N}$, we define $\mathfrak{L}_{\mathrm{ro},i} := \mathcal{L}_{\mathfrak{L}_{\mathrm{ro},i-1}}(\mathrm{PAT}_{\mathrm{ro}})$. Furthermore, we define $\mathfrak{L}_{\mathrm{ro},\infty} = \bigcup_{i=0}^{\infty} \mathfrak{L}_{\mathrm{ro},i}$.

It follows by definition, that the classes $\mathfrak{L}_{\mathrm{ro},i}$, $i \in \mathbb{N}_0$, form a hierarchy and we strongly conjecture that it is proper. However, here we only separate the first three levels of that hierarchy.

Theorem 7.6. $\mathfrak{L}_{\mathrm{ro},0} \subset \mathfrak{L}_{\mathrm{ro},1} \subset \mathfrak{L}_{\mathrm{ro},2} \subseteq \mathfrak{L}_{\mathrm{ro},3} \subseteq \mathfrak{L}_{\mathrm{ro},4} \subseteq \ldots$.

Proof. The inclusions follow by definition and it is obvious that $\mathfrak{L}_{\mathrm{ro},0}$, which is the set of regular languages, is properly included in $\mathfrak{L}_{\mathrm{ro},1}$. Hence, it only remains to show that there exists a language in $\mathfrak{L}_{\mathrm{ro},2}$ that is not in $\mathfrak{L}_{\mathrm{ro},1}$. To this end, we define $L := \{(\mathtt{a}^n\mathtt{c}\mathtt{a}^n)^m\mathtt{d}(\mathtt{a}^n\mathtt{c}\mathtt{a}^n)^m \mid n, m \in \mathbb{N}\}$ and first note that $\mathcal{L}_{(L_1)}(x_1 \cdot \mathtt{d} \cdot x_1) = L$, where $L_1 := \mathcal{L}_{(\mathcal{L}(\mathtt{a}^+))}((x_1 \cdot \mathtt{c} \cdot x_1)^+)$, which shows that $L \in \mathfrak{L}_{\mathrm{ro},2}$.

We now assume that $L \in \mathfrak{L}_{\mathrm{ro},1}$ and show that this assumption leads to a contradiction. If $L \in \mathfrak{L}_{\mathrm{ro},1}$, then there exists a pattern with regular operators α and a regular type $\mathcal{T} := (T_{x_1}, T_{x_2}, \ldots, T_{x_m})$ for α such that $\mathcal{L}_{\mathcal{T}}(\alpha) = L$. We shall first assume that $\mathcal{L}(\alpha)$ is finite. Since the number of occurrences of $\mathtt{c}$ as well as the length of unary factors over $\mathtt{a}$ is unbounded in the words of L, there must exist at least one variable x such that, for every $n \in \mathbb{N}$, there exists a word in T_x containing a factor $\mathtt{c} \cdot \mathtt{a}^{n'} \cdot \mathtt{c} \cdot \mathtt{a}^{n'} \cdot \mathtt{c}$, where $n \leq n'$. This particularly implies that there also exists a word in T_x containing a factor $\mathtt{c} \cdot \mathtt{a}^{n'} \cdot \mathtt{c} \cdot \mathtt{a}^{n'} \cdot \mathtt{c}$, where n' is greater than the constant of Lemma 2.2 (see page 9) with respect to the regular language T_x. By applying Lemma 2.2, we can show that in T_x there exists a word containing a factor $\mathtt{c} \cdot \mathtt{a}^m \cdot \mathtt{c} \cdot \mathtt{a}^{m'} \cdot \mathtt{c}$, $m \neq m'$, which is a contradiction, since this implies that there is a word in L that contains the factor $\mathtt{c} \cdot \mathtt{a}^m \cdot \mathtt{c} \cdot \mathtt{a}^{m'} \cdot \mathtt{c}$.

Next, we assume that $\mathcal{L}(\alpha)$ is infinite. We further assume that α does not contain any terminal symbols and, furthermore, for every i, $1 \leq i \leq m$, $T_{x_i} \neq \{\varepsilon\}$. This is not a loss of generality, since terminal symbols can be easily represented by variables with only a single occurrence and a type language of form $\{b\}$, $b \in \Sigma$, and any variable that is typed by $\{\varepsilon\}$ can be erased without changing $\mathcal{L}_{\mathcal{T}}(\alpha)$. Since every word of L contains exactly one occurrence of $\mathtt{d}$, we can conclude that in α there are variables $y_1, y_2, \ldots, y_l$, $l \in \mathbb{N}$, such that, for every i, $1 \leq i \leq l$, T_{y_i} contains at least one word with exactly one occurrence of $\mathtt{d}$. Furthermore, for every $\beta \in \mathcal{L}(\alpha)$, there exists a j, $1 \leq j \leq l$, such that $\beta = \delta \cdot y_j \cdot \gamma$ and $\mathrm{var}(\delta \cdot \gamma) \cap \{y_1, y_2, \ldots, y_l\} = \emptyset$. This is due to the fact that if a $\beta \in \mathcal{L}(\alpha)$ contains more than one occurrence of a variable y_i, $1 \leq i \leq l$, then $\mathcal{L}(\alpha)$ contains a word with more than one occurrence of $\mathtt{d}$. Since $\mathcal{L}(\alpha)$ is infinite, for some j, $1 \leq j \leq l$, there exists a word $\delta \cdot y_j \cdot \gamma$ in $\mathcal{L}(\alpha)$ such that $\mathrm{var}(\delta \cdot \gamma) \cap \{y_1, y_2, \ldots, y_l\} = \emptyset$ and $|\delta| > k$ or $|\gamma| > k$, where k is the constant of Lemma 2.2 with respect to the regular language $\mathcal{L}(\alpha)$. This implies that δ (or γ, respectively) can be arbitrarily pumped and, since every type language contains at least one non-empty word, this implies that there is a word in $\mathcal{L}_{\mathcal{T}}(\alpha)$ of form $u \cdot \mathtt{d} \cdot v$ with $|u| > |v|$ (or $|u| < |v|$, respectively), which is a contradiction. This shows that in fact $L \notin \mathfrak{L}_{\mathrm{ro},1}$ and, thus, $\mathfrak{L}_{\mathrm{ro},1} \subset \mathfrak{L}_{\mathrm{ro},2}$ is implied. □

In the following section, we take a closer look at the class $\mathfrak{L}_{\mathrm{ro},\infty}$. We shall show that it coincides with the class of languages that are defined by the already mentioned pattern expressions and we formally prove it to be a proper subset of the class of REGEX languages.

7.2 Pattern Expressions

We define pattern expressions as introduced by Câmpeanu and Yu [12], but we use a slightly different notation.

Definition 7.7. A *pattern expression* is a tuple $(x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$, where, for every i, $1 \leq i \leq n$, $r_i \in \mathrm{PAT}_{\mathrm{ro}}$ and $\mathrm{var}(r_i) \subseteq \{x_1, x_2, \ldots, x_{i-1}\}$. The set of all pattern expressions is denoted by PE.

In [12], the language of a pattern expression $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$ is defined in the following way. Since, by definition, r_1 is a classical regular expression, it describes a regular language L. The language L is then interpreted as a type for variable x_1 in every r_i, $2 \leq i \leq n$. This step is then repeated, i. e., $\mathcal{L}_{(L)}(r_2)$ is the type for x_2 in every r_j, $3 \leq j \leq n$, and so on.

Definition 7.8. Let $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$ be a pattern expression. We define $L_{p,x_1} := \mathcal{L}(r_1)$ and, for every i, $2 \leq i \leq n$, $L_{p,x_i} := \mathcal{L}_{\mathcal{T}_i}(r_i)$, where $\mathcal{T}_i := (L_{p,x_1}, L_{p,x_2}, \ldots, L_{p,x_{i-1}})$ is a type for r_i. The *language generated by p with respect to iterated substitution* is defined by $\mathcal{L}_{\text{it}}(p) := L_{p,x_n}$ and $\mathcal{L}_{\text{it}}(\text{PE}) := \{\mathcal{L}_{\text{it}}(p) \mid p \in \text{PE}\}$.

We illustrate the above definition with an example. Let

$$q := (x_1 \to \mathtt{a}^*, x_2 \to x_1(\mathtt{c} \mid \mathtt{d})x_1, x_3 \to x_1\mathtt{c}x_2)$$

be a pattern expression. According to the above definition, $\mathcal{L}_{\text{it}}(q) = \{\mathtt{a}^k\mathtt{c}\mathtt{a}^m u \mathtt{a}^m \mid k, m \in \mathbb{N}_0, u \in \{\mathtt{c}, \mathtt{d}\}\}$. We note that in a word $\mathtt{a}^k\mathtt{c}\mathtt{a}^m u\mathtt{a}^m \in \mathcal{L}_{\text{it}}(q)$, both $\mathtt{a}^k$ and $\mathtt{a}^m$ are substitution words for the same variable x_1 from the type language L_{q,x_1}. However, $k \neq m$ is possible, since, intuitively speaking, $\mathtt{a}^k$ is picked first from L_{q,x_1} as the substitution word for x_1 in $x_1\mathtt{c}x_2$ and then $\mathtt{a}^m$ is picked from L_{q,x_1} as substitution word for x_1 in $x_1(\mathtt{c} \mid \mathtt{d})x_1$ in order to construct the substitution word $\mathtt{a}^m u\mathtt{a}^m$ for x_2 in $x_1\mathtt{c}x_2$. Consequently, occurrences of the same variable in different elements of the pattern expression do not need to be substituted by the same word. We shall later see that this behaviour essentially limits the expressive power of pattern expressions.

As mentioned before, the class of languages described by pattern expressions with respect to iterated substitution coincides with the class $\mathfrak{L}_{\text{ro},\infty}$ of the previous section.

Theorem 7.9. $\mathfrak{L}_{\text{ro},\infty} = \mathcal{L}_{\text{it}}(\text{PE})$.

Proof. Let $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$ be a pattern expression and, for every i, $1 \leq i \leq n$, let the languages L_{p,x_i} be defined as in Definition 7.8. We prove by induction that, for every i, $1 \leq i \leq n$, $L_{p,x_i} \in \mathfrak{L}_{\text{ro},i-1}$, which implies $\mathcal{L}_{\text{it}}(\text{PE}) \subseteq \mathfrak{L}_{\text{ro},\infty}$. First, we note that $L_{p,x_1} \in \text{REG}$ and, thus, $L_{p,x_1} \in \mathfrak{L}_{\text{ro},0}$. Next, we assume that for some i, $2 \leq i \leq n$, and for every j, $1 \leq j < i$, $L_{p,x_j} \in \mathfrak{L}_{\text{ro},j-1}$. This implies that $\mathcal{T} := (L_{p,x_1}, L_{p,x_2}, \ldots, L_{p,x_{i-1}})$ is an $\mathfrak{L}_{\text{ro},i-2}$-type for r_i. Thus, $\mathcal{L}_{\mathcal{T}}(r_i) \in \mathfrak{L}_{\text{ro},i-1}$ and, since $\mathcal{L}_{\mathcal{T}}(r_i) = L_{p,x_i}$, we can conclude that $L_{p,x_i} \in \mathfrak{L}_{\text{ro},i-1}$.

Next, we shall prove by induction that, for every $i \in \mathbb{N}_0$, $\mathfrak{L}_{\text{ro},i} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$. Obviously, $\mathfrak{L}_{\text{ro},0}$ is included in $\mathcal{L}_{\text{it}}(\text{PE})$. Now we assume that for some $k \in \mathbb{N}$, $\mathfrak{L}_{\text{ro},k-1} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$ holds and we show that this implies $\mathfrak{L}_{\text{ro},k} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$. To this end, we let $L \in \mathfrak{L}_{\text{ro},k}$, which implies that there exists a pattern α_k with regular operators with $\text{var}(\alpha_k) = \{x_1, x_2, \ldots, x_m\}$ and, for every i, $1 \leq i \leq m$, there exists a language $T_{x_i} \in \mathfrak{L}_{\text{ro},k-1}$, such that $\mathcal{L}_{(T_{x_1},\dots,T_{x_m})}(\alpha_k) = L$. Now, for every i, $1 \leq i \leq m$, let p_i be a pattern expression with $\mathcal{L}_{\text{it}}(p_i) = T_{x_i}$. Such pattern expressions exist since $T_{x_i} \in \mathfrak{L}_{\text{ro},k-1} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$, $1 \leq i \leq m$. We assume that, for every i, $1 \leq i \leq m$, the last element of p_i is $x_i \rightarrow \beta_i$ and the sets of variables used in the pattern expressions p_i, $1 \leq i \leq m$, as well as the set of variables in α_k are pairwise disjoint. We construct a pattern expression $\widehat{p}$ by adding all the elements of the pattern expressions p_i, $1 \leq i \leq m$, to a new tuple in such a way that, for every i, $1 \leq i \leq m$, the relative order of all the elements in p_i is not changed. Furthermore, we add the element $z \rightarrow \alpha_k$ to the right of $\widehat{p}$, where z is a new variable. By definition, for every i, $1 \leq i \leq m$, $L_{\widehat{p},x_i} = \mathcal{L}_{\text{it}}(p_i) = T_{x_i}$. This directly implies that $\mathcal{L}_{\text{it}}(\widehat{p}) = \mathcal{L}_{(T_{x_1},\dots,T_{x_m})}(\alpha_k)$ and, thus, $L \in \mathcal{L}_{\text{it}}(\text{PE})$ and $\mathfrak{L}_{\text{ro},k} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$ follows. This implies that $\mathfrak{L}_{\text{ro},\infty} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$, which concludes the proof. $\square$

In the following, we define an alternative way of how pattern expressions can describe languages, i. e., instead of substituting the variables by words in an iterative way, we substitute them uniformly. It shall be shown later own that this amendment increases the expressive power of pattern expressions.

Definition 7.10. Let $p := (x_1 \rightarrow r_1, x_2 \rightarrow r_2, \dots, x_n \rightarrow r_n) \in \text{PE}$. A word $w \in \Sigma^*$ is in the *language generated by p with respect to uniform substitution* ($\mathcal{L}_{\text{uni}}(p)$, for short) if and only if there exists a substitution h such that $h(x_n) = w$ and, for every i, $1 \leq i \leq n$, there exists an $\alpha_i \in \mathcal{L}(r_i)$ with $h(x_i) = h(\alpha_i)$.

For the pattern expression q from above, a word w is in $\mathcal{L}_{\text{uni}}(q)$ if there is a substitution h with $h(x_3) = w$ and there exist $\alpha_1 \in \mathcal{L}(\mathtt{a}^*)$, $\alpha_2 \in \mathcal{L}(x_1(\mathtt{c} \mid \mathtt{d})x_1)$ and $\alpha_3 \in \mathcal{L}(x_1\mathtt{c}x_2)$, such that $h(x_1) = h(\alpha_1)$, $h(x_2) = h(\alpha_2)$ and $h(x_3) = h(\alpha_3)$. Since $\alpha_1 = \mathtt{a}^n$, $n \in \mathbb{N}_0$, $\alpha_2 = x_1 u x_1$, $u \in \{\mathtt{c}, \mathtt{d}\}$, and $\alpha_3 = x_1\mathtt{c}x_2$, this implies that w is in $\mathcal{L}_{\text{uni}}(q)$ if there is a substitution h and an $\alpha := x_1\mathtt{c}x_1ux_1$, $u \in \{\mathtt{c}, \mathtt{d}\}$, such that $w = h(\alpha)$ and h satisfies the type $(\mathcal{L}(\mathtt{a}^*))$. Thus, $\mathcal{L}_{\text{uni}}(q) = \{\mathtt{a}^n\mathtt{c}\mathtt{a}^n u \mathtt{a}^n \mid n \in \mathbb{N}_0, u \in \{\mathtt{c}, \mathtt{d}\}\}$, which is a proper subset of $\mathcal{L}_{\text{it}}(q)$.

For an arbitrary pattern expression $p := (x_1 \rightarrow r_1, x_2 \rightarrow r_2, \dots, x_n \rightarrow r_n)$, the language $\mathcal{L}_{\text{uni}}(p)$ can also be defined in a more constructive way. We first choose a word $u \in \mathcal{L}(r_1)$ and, for all i, $1 \leq i \leq n$, if variable x_1 occurs in r_i, then we substitute all occurrences of x_1 in r_i by u. Then we delete the element $x_1 \rightarrow r_1$ from the pattern expression. If we repeat this step with respect to variables $x_2, x_3, \dots, x_{n-1}$, then we obtain a pattern expression of form $(x_n \rightarrow r'_n)$, where r'_n is a regular expression over Σ. The language $\mathcal{L}_{\text{uni}}(p)$ is the union of the languages given by all these regular expression.

The language $\mathcal{L}_{\text{it}}(q)$ can be defined similarly. We first choose a word $u_1 \in \mathcal{L}(r_1)$ and then we substitute all occurrences of x_1 in r_2 by u_1. After that, we choose a *new* word $u_2 \in \mathcal{L}(r_1)$ and substitute all occurrences of x_1 in r_3 by u_2 and so on until there are no more occurrences of variable x_1 in q and then we delete the element $x_1 \rightarrow r_1$. Then this step is repeated with respect to $x_2, x_3, \dots, x_{n-1}$.

The above considerations yield the following proposition:

Proposition 7.11. *Let $p := (x_1 \rightarrow r_1, x_2 \rightarrow r_2, \dots, x_m \rightarrow r_m)$ be a pattern expression. Then $\mathcal{L}_{\text{uni}}(p) \subseteq \mathcal{L}_{\text{it}}(p)$ and if, for every i, j, $1 \leq i < j \leq m$, $\text{var}(r_i) \cap \text{var}(r_j) = \emptyset$, then also $\mathcal{L}_{\text{it}}(p) \subseteq \mathcal{L}_{\text{uni}}(p)$.*

Proof. We recall that every word $w \in \mathcal{L}_{\text{it}}(p)$ is a member of a language given by a classical regular expression that can be constructed by applying *procedure* 1: We first choose a word $u_1 \in \mathcal{L}(r_1)$ and then we substitute all occurrences of x_1 in r_2 by u_1. After that, we choose a new word $u_2 \in \mathcal{L}(r_1)$ and substitute all occurrences of x_1 in r_3 by u_2 and so on until there are no more occurrences of variable x_1 in q and then we delete the element $x_1 \rightarrow r_1$. Then we repeat this step with respect to variables $x_2, x_3, \dots, x_{m-1}$.

On the other hand, every word in $\mathcal{L}_{\text{uni}}(p)$ is a member of a language given by a classical regular expression that can be constructed by applying *procedure* 2: We choose a word $u \in \mathcal{L}(r_1)$ and, for all i, $1 \leq i \leq n$, if variable x_1 occurs in r_i, then we substitute all occurrences of x_1 in r_i by u and delete the element $x_1 \rightarrow r_1$ from the pattern expression. Then we repeat this step with respect to variables $x_2, x_3, \dots, x_{m-1}$.

Obviously, every classical regular expression constructed by procedure 2 can also be constructed by procedure 1 by substituting every occurrence of a variable x_i by exactly the same word. Hence, $\mathcal{L}_{\text{uni}}(p) \subseteq \mathcal{L}_{\text{it}}(p)$. Furthermore, if, for every i, j, $1 \leq i < j \leq m$, $\text{var}(r_i) \cap \text{var}(r_j) = \emptyset$, then, for every i, $1 \leq i \leq m-1$, there is exactly one j, $i < j \leq m$, such that x_i occurs in r_j, and, thus, procedure 1 and procedure 2 are identical, which implies $\mathcal{L}_{\text{it}}(p) = \mathcal{L}_{\text{uni}}(p)$. $\square$

The interesting question is whether or not there exists a language $L \in \mathcal{L}_{\text{uni}}(\text{PE})$ with $L \notin \mathcal{L}_{\text{it}}(\text{PE})$ or vice versa. Intuitively, for any pattern expression p, it seems obvious that it is not essential for the language $\mathcal{L}_{\text{it}}(p)$ that there exist occurrences of the same variable in different elements of p and it should be possible to transform p into an equivalent pattern expression p', the elements of which have disjoint sets of variables and, thus, by Proposition 7.11, $\mathcal{L}_{\text{it}}(p) = \mathcal{L}_{\text{uni}}(p')$. Hence, for the language generated by a pattern expression with respect to iterated substitution, the possibility of using the same variables in different elements of a pattern expression can be considered as mere syntactic sugar that keeps pattern expressions concise. On the other hand, the question of whether or not, for every pattern expression p, we can find a pattern expression p' with $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}_{\text{it}}(p')$, is not that easy to answer. The following lemma states that there are in fact languages that can be expressed by some pattern expression with respect to uniform substitution, but not by any pattern expression with respect to iterated substitution.

Lemma 7.12. *There exists a language $L \in \mathcal{L}_{\text{uni}}(\text{PE})$ with $L \notin \mathcal{L}_{\text{it}}(\text{PE})$.*

Proof. We define the language $L := \{(\mathtt{a}^n\mathtt{c})^m \cdot \mathtt{b} \cdot \mathtt{a}^n \cdot \mathtt{d} \cdot (\mathtt{a}^n\mathtt{c})^m \mid n, m \in \mathbb{N}\}$ and a pattern expression $p := (x_1 \to \mathtt{a}^+, x_2 \to (x_1 \cdot \mathtt{c})^+, x_3 \to x_2 \cdot \mathtt{b} \cdot x_1 \cdot \mathtt{d} \cdot x_2)$. Obviously, $\mathcal{L}_{\text{uni}}(p) = L$ and, thus, $L \in \mathcal{L}_{\text{uni}}(\text{PE})$. In the following we shall show that $L \notin \mathcal{L}_{\text{it}}(\text{PE})$. To this end, we first prove the following claim.

Claim (1). Let q be a pattern expression. There exists a pattern expression $q' := (x_1 \to t'_1, x_2 \to t'_2, \ldots, x_{m'} \to t'_{m'})$, such that, for every i, $1 \leq i \leq m' - 1$, $\mathcal{L}(t'_i)$ is infinite and $\mathcal{L}_{\text{it}}(q) = \mathcal{L}_{\text{it}}(q')$.

Proof. (*Claim* (1)) Let $q := (x_1 \to t_1, x_2 \to t_2, \ldots, x_m \to t_m)$ and let l, $1 \leq l \leq m$, be the smallest number such that $\mathcal{L}(t_l) := \{\beta_1, \beta_2, \ldots, \beta_k\}$ is finite. If l is not defined because, for every i, $1 \leq i \leq m$, $\mathcal{L}(t_i)$ is infinite or if $l = m$, then q already satisfies the condition of the lemma, i. e., for every i, $1 \leq i \leq m - 1$, $\mathcal{L}(t'_i)$ is infinite. If, on the other hand, $l \leq m - 1$, then we can transform q into a pattern expression $q'' := (x_1 \to t''_1, x_2 \to t''_2, \ldots, x_{l-1} \to t''_{l-1}, x_{l+1} \to t''_{l+1}, \ldots, x_m \to t''_m)$ in the following way. For every i, $l+1 \leq i \leq m$, we replace t_i by $(t_{i,1} \mid t_{i,2} \mid \ldots \mid t_{i,k})$, where, for every j, $1 \leq j \leq k$, $t_{i,j}$ is obtained from t_i by substituting every occurrence of x_l by β_j. For every i, $1 \leq i \leq l - 1$, we do not change t_i, i. e., $t''_i := t_i$. Hence, the element $x_l \to t_l$ has been removed from q. It is straightforward to see that $\mathcal{L}_{\text{it}}(q) = \mathcal{L}_{\text{it}}(q'')$. Furthermore, by repeating this procedure, q can be transformed into $q' = (x_1 \to t'_1, x_2 \to t'_2, \ldots, x_{m'} \to t'_{m'})$, where, for every i, $1 \leq i \leq m' - 1$, $\mathcal{L}(t'_i)$ is infinite and $\mathcal{L}_{\text{it}}(q) = \mathcal{L}_{\text{it}}(q')$. □ (*Claim* (1))

We now assume contrary to the statement of the lemma, that there exists a pattern expression $p' := (x_1 \to r_1, x_2 \to r_2, \ldots, x_m \to r_m)$ with $\mathcal{L}_{\text{it}}(p') = L$, which shall lead to a contradiction. For every i, $1 \leq i \leq m$, let L_{p',x_i} be the language as introduced in Definition 7.8. By the above claim, we can also assume that, for every i, $1 \leq i \leq m - 1$, $\mathcal{L}(r_i)$ is infinite. Next, we prove the following claim.

Claim (2). For every i, $1 \leq i \leq m$, if L_{p',x_i} contains a word with an occurrence of $\mathtt{b}$ or $\mathtt{d}$, then $\mathcal{L}(r_i)$ is finite.

Proof. (*Claim* (2)) We first show that if, for some i, $1 \leq i \leq m$, there exists a word $w \in L_{p',x_i}$ with $|w|_{\mathtt{b}} \geq 1$, then all words in L_{p',x_i} contain exactly one occurrence of $\mathtt{b}$. To this end, we note that every word in L_{p',x_i} has at most one occurrence of $\mathtt{b}$, since every word in L contains at most one occurrence of $\mathtt{b}$. Furthermore, it is not possible that there exists a word $w \in L_{p',x_i}$ with $|w|_{\mathtt{b}} = 1$ and another word $w' \in L_{p',x_i}$ with $|w'|_{\mathtt{b}} = 0$, since this implies that there are two words in $\mathcal{L}_{\text{it}}(p')$

with a different number of occurrences of $\mathtt{b}$, which is a contradiction. In the same way we can show that if there exists at least one word in L_{p',x_i} with an occurrence of symbol $\mathtt{d}$, then every word in L_{p',x_i} contains exactly one occurrence of symbol $\mathtt{d}$.

Next, we assume that for some l, $1 \leq l \leq m$, L_{p',x_l} contains a word with an occurrence of $\mathtt{b}$ or $\mathtt{d}$, but, contrary to the above claim, $\mathcal{L}(r_l)$ is infinite. Now if L_{p',x_l} contains a word with an occurrence of $\mathtt{b}$, then, as pointed out above, all the words of L_{p',x_l} contain exactly one occurrence of $\mathtt{b}$, which implies that, for every $\beta \in \mathcal{L}(r_l)$, $\beta = \gamma \cdot z \cdot \gamma'$, where either $z = \mathtt{b}$ or $z = x_j$, $1 \leq j < l$, such that all the words of L_{p',x_j} contain exactly one occurrence of $\mathtt{b}$. Moreover, since $\mathcal{L}(r_l)$ is infinite, we can assume that $|\gamma|$ or $|\gamma'|$ exceeds the constant of Lemma 2.2 (see page 9) for the regular language $\mathcal{L}(r_l)$. Consequently, by applying the Lemma 2.2, we can produce a word $\widehat{\gamma} \cdot z \cdot \gamma' \in \mathcal{L}(r_l)$ with $|\gamma| < |\widehat{\gamma}|$ or a word $\gamma \cdot z \cdot \widehat{\gamma} \in \mathcal{L}(r_l)$ with $|\gamma'| < |\widehat{\gamma}|$, respectively. Since, without loss of generality, we can assume that, for every i, $1 \leq i \leq m$, $L_{p,x_i} \neq \{\varepsilon\}$, this directly implies that there exists a word $w \in \mathcal{L}_{\text{it}}(p')$ that is of form $w = u \cdot \mathtt{b} \cdot v$, where it is not satisfied that there exist $n, m \in \mathbb{N}$ with $|u| = (n+1)m$ and $|v| = (n+1)m+n+1$, which is a contradiction. If L_{p',x_l} contains a word with an occurrence of $\mathtt{d}$ and $\mathcal{L}(r_l)$ is infinite, then we can obtain a contradiction in an analogous way. Consequently, if any L_{p',x_i}, $1 \leq i \leq m$, contains a word with an occurrence of $\mathtt{b}$ or $\mathtt{d}$, then $\mathcal{L}(r_i)$ is finite. $\square$ (*Claim* (2))

The above claim particularly implies that, since L_{p',x_m} clearly contains words with $\mathtt{b}$ and $\mathtt{d}$, $\mathcal{L}(r_m)$ is finite and, for every i, $1 \leq i \leq m-1$, since $\mathcal{L}(r_i)$ is infinite, L_{p',x_i} does not contain a word with an occurrence of $\mathtt{b}$ or $\mathtt{d}$. Hence, without loss of generality, we can assume that $r_m := (\beta_1 \mid \beta_2 \mid \ldots \mid \beta_k)$ with $\beta_i := \gamma_i \cdot \mathtt{b} \cdot \gamma'_i \cdot \mathtt{d} \cdot \gamma''_i \in$ PAT, $1 \leq i \leq k$. For every i, $1 \leq i \leq k$, and for every j, $1 \leq j \leq m-1$, let $\widehat{L}_i$ be the set of all words that can be obtained by substituting every occurrence of x_j in β_i by some word from L_{p',x_j}, i. e., $\widehat{L}_i := \mathcal{L}_{(L_{p',x_1},\ldots,L_{p',x_{m-1}})}(\beta_i)$. Obviously, $\mathcal{L}_{\text{it}}(p') = \widehat{L}_1 \cup \widehat{L}_2 \cup \ldots \cup \widehat{L}_k$. This implies that there must exist at least one s, $1 \leq s \leq k$, such that the number of occurrences of $\mathtt{c}$ and the length of the factor between the occurrence of $\mathtt{b}$ and $\mathtt{d}$ is unbounded in the words of $\widehat{L}_s$. More precisely, there must exist at least one s, $1 \leq s \leq k$, such that, for every $n \in \mathbb{N}$, there exists a word $w \in \widehat{L}_s$ with $|w|_{\mathtt{c}} > n$ and a word $w' \in \widehat{L}_s$ with $w' = u \cdot \mathtt{b} \cdot \mathtt{a}^{n'} \cdot \mathtt{d} \cdot v$, for some n', $n < n'$. This implies that in γ_s there must occur a variable x_j such that the number of occurrences of $\mathtt{c}$ is unbounded in L_{p',x_j}. Moreover, we can assume that there is also an occurrence of x_j in γ''_s, since otherwise there would be a word in $\widehat{L}_s$ with a different number of occurrences of $\mathtt{c}$ to the left of $\mathtt{b}$ than to the right of $\mathtt{d}$. Similarly, in γ'_s there must occur a variable $x_{j'}$ such that $L_{p',x_{j'}}$ is an infinite unary language over $\{\mathtt{a}\}$, which particularly implies that $j \neq j'$. We note further that in L_{p',x_j}, there exists a word u with a factor $\mathtt{c} \cdot \mathtt{a}^n \cdot \mathtt{c}$, for some $n \in \mathbb{N}$. We can now obtain β'_s by substituting every occurrence of x_j in β_s by u. Next, we obtain β''_s from β'_s by substituting every occurrence of $x_{j'}$ by a word $\mathtt{a}^{n'}$ with $n < n'$. Next, for every i, $1 \leq i \leq m-1$, we substitute all occurrences of variable x_i in β''_s by some word from L_{p',x_i}. The thus constructed word is in $\widehat{L}_s$, but not in L, since it contains both a factor $\mathtt{c} \cdot \mathtt{a}^n \cdot \mathtt{c}$ and $\mathtt{b} \cdot \mathtt{a}^{n''} \cdot \mathtt{d}$ with $n < n' \leq n''$. This is a contradiction. $\square$

From Lemma 7.12 we can conclude the main result of this section, i. e., the class of languages given by pattern expressions with respect to iterated substitution is a proper subset of the class of languages given by pattern expressions with respect to uniform substitution.

Theorem 7.13. $\mathcal{L}_{\text{it}}(\text{PE}) \subset \mathcal{L}_{\text{uni}}(\text{PE})$.

Proof. Since, by Theorem 7.9, $\mathfrak{L}_{\text{ro},\infty} = \mathcal{L}_{\text{it}}(\text{PE})$ holds, it is sufficient to show $\mathfrak{L}_{\text{ro},\infty} \subseteq \mathcal{L}_{\text{uni}}(\text{PE})$ in order to conclude $\mathcal{L}_{\text{it}}(\text{PE}) \subseteq \mathcal{L}_{\text{uni}}(\text{PE})$, which can be done in exactly the same way as $\mathfrak{L}_{\text{ro},\infty} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$ has been shown in the proof of Theorem 7.9. From Lemma 7.12, we can then conclude that this inclusion is proper. $\square$

We conclude this section by mentioning that in Bordihn et al. [9], it has been shown that $\mathcal{H}^*(\text{REG}, \text{REG})$, a class of languages given by an iterated version of H-systems (see Albert and Wegner [2] and Bordihn et al. [9]), also coincides with $\mathcal{L}_{\text{it}}(\text{PE})$, which implies $\mathfrak{L}_{\text{ro},\infty} = \mathcal{L}_{\text{it}}(\text{PE}) = \mathcal{H}^*(\text{REG}, \text{REG}) \subset \mathcal{L}_{\text{uni}}(\text{PE})$.

In the following section, we take a closer look at the larger class $\mathcal{L}_{\text{uni}}(\text{PE})$ and compare it to the class of REGEX languages.

7.3 REGEX

We use a slightly different notation for REGEX compared to the one used by Câmpeanu et al. in [11].

A REGEX is a regular expression, the subexpressions of which can be numbered by adding an integer index to the parentheses delimiting the subexpression (i. e., $(_n \ \ldots \ )_n$, $n \in \mathbb{N}$). This is done in such a way that there are no two different subexpressions with the same number. The subexpression that is numbered by $n \in \mathbb{N}$, which is called the n^{th} *referenced subexpression*, can be followed by arbitrarily many *backreferences* to that subexpression, denoted by $\backslash n$. For example, $(_1 \ \mathtt{a} \mid \mathtt{b}\ )_1 \cdot (_2 \ (\mathtt{c} \mid \mathtt{a})^* \ )_2 \cdot (\backslash 1)^* \cdot \backslash 2$ is a REGEX, whereas $r_1 := (_1 \ \mathtt{a} \mid \mathtt{b}\ )_1 \cdot (_1 \ (\mathtt{c} \mid \mathtt{a})^* \ )_1 \cdot (\backslash 1)^* \cdot \backslash 2$ and $r_2 := (_1 \ \mathtt{a} \mid \mathtt{b}\ )_1 \cdot \backslash 2 \cdot (_2 \ (\mathtt{c} \mid \mathtt{a})^* \ )_2 \cdot (\backslash 1)^* \cdot \backslash 2$ are not a REGEX, since in r_1 there are two different subexpressions numbered by 1 and in r_2 there is an occurrence of a backreference $\backslash 2$ before the second referenced subexpression.

There are two aspects of REGEX that need to be discussed in a bit more detail. For the REGEX $((_1\ \mathtt{a}^+\ )_1 \mid \mathtt{b}) \cdot \mathtt{c} \cdot \backslash 1$, if we choose the option $\mathtt{b}$ in the alternation, then $\backslash 1$ points to a subexpression that has not been "initialised". Normally, such a backreference is then interpreted as the empty word, which seems to be the only reasonable way to handle this situation, but, on the other hand, conflicts with the intended semantics of backreferences, particularly in the above example, since it actually means that $\backslash 1$ can be the empty word, whereas the referenced subexpression $(_1\ \mathtt{a}^+\ )_1$ does not match the empty word.

Another particularity appears whenever a backreference points to a subexpression under a star, e. g., $s := ((_1\ \mathtt{a}^*\ )_1 \cdot \mathtt{b} \cdot \backslash 1)^* \cdot \mathtt{c} \cdot \backslash 1$. One might expect s to define the set of all words of form $(\mathtt{a}^n\mathtt{b}\mathtt{a}^n)^m\mathtt{c}\mathtt{a}^n$, $n, m \geq 0$, but s really describes the set $\{\mathtt{a}^{n_1}\mathtt{b}\mathtt{a}^{n_1} \cdot \mathtt{a}^{n_2}\mathtt{b}\mathtt{a}^{n_2} \cdot \cdots \cdot \mathtt{a}^{n_m}\mathtt{b}\mathtt{a}^{n_m} \cdot \mathtt{c} \cdot \mathtt{a}^{n_m} \mid m \geq 1, n_i \geq 0, 1 \leq i \leq m\} \cup \{\mathtt{c}\}$. This is due to the fact that the star operation repeats a subexpression several times without imposing any dependencies between the single iterations. Consequently, in every iteration of the second star in s, the referenced subexpression $(_1\ \mathtt{a}^*\ )_1$ is treated as an individual instance and its scope is restricted to the current iteration. Only the factor that $(_1\ \mathtt{a}^*\ )_1$ matches in the very last iteration is then referenced by any backreference $\backslash 1$ outside the star. A way to see that this behaviour, which is often called *late binding* of backreferences (see Câmpeanu and Yu [12]), is reasonable, is to observe that if we require $(_1\ \mathtt{a}^*\ )_1$ to take exactly the same value in every iteration of the star, then, for some REGEX r, this may lead to $\mathcal{L}(r^*) \neq (\mathcal{L}(r))^*$.

A formal definition of the language described by a REGEX is provided by Câmpeanu et al. in [11]. Here, we stick to the more informal definition which has already been briefly outlined in the introduction to Chapter 7 and that we now recall in a bit more detail.

A word w is in $\mathcal{L}(r)$ if and only if we can obtain it from r in the following way. We move over r from left to right. We treat alternations and stars as it is done for classical regular expressions and we note down every terminal symbol that we read. When we encounter the i^{th} referenced subexpression, then we store the factor u_i that is matched to it and from now on we treat every occurrence of $\backslash i$ as u_i. However, there are two special cases we need to take care of. Firstly, when we encounter the i^{th} referenced subexpression for a second time, which is possible since the i^{th} referenced subexpression may occur under a star, then we overwrite u_i with the possible new factor that is now matched to the i^{th} referenced subexpression. This entails the late binding of backreferences, which has been described in the introduction of the present chapter. Secondly, if a backreference $\backslash i$

occurs and there is no factor u_i stored that has been matched to the i^{th} referenced subexpression, then $\backslash i$ is interpreted as the empty word.

We also define an alternative way of how a REGEX describes a language, that shall be useful for our proofs. The *language with necessarily initialised subexpressions* of a REGEX r, denoted by $\mathcal{L}_{\text{nis}}(r)$, is defined in a similar way as $\mathcal{L}(r)$ above, but if a backreference $\backslash i$ occurs and there is currently no factor u_i stored that has been matched to the i^{th} referenced subexpression, then instead of treating $\backslash i$ as the empty word, we interpret it as the i^{th} referenced subexpression, we store the factor u_i that is matched to it and from now on every occurrence of $\backslash i$ is treated as u_i. For example, let $r := ((_1\, \mathtt{a}^*\,)_1 \mid \varepsilon) \cdot \mathtt{b} \cdot \backslash 1 \cdot \mathtt{b} \cdot \backslash 1$. Then $\mathcal{L}(r) := \{\mathtt{a}^n\mathtt{b}\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}_0\}$ and $\mathcal{L}_{\text{nis}}(r) := \mathcal{L}(r) \cup \{\mathtt{b}\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}_0\}$.

We can note that the late binding of backreferences as well as non-initialised referenced subexpressions is caused by referenced subexpression under a star or in an alternation. Next, we define REGEX that are restricted in this regard.

Definition 7.14. A REGEX r is *alternation confined* if and only if the existence of a referenced subexpression in the option of an alternation implies that all the corresponding backreferences occur in the same option of the same alternation. A REGEX r is *star-free initialised* if and only if every referenced subexpression does not occur under a star. Let REGEX_{ac} and $\text{REGEX}_{\text{sfi}}$ be the sets of REGEX that are alternation confined and star-free initialised, respectively. Furthermore, let $\text{REGEX}_{\text{sfi,ac}} := \text{REGEX}_{\text{ac}} \cap \text{REGEX}_{\text{sfi}}$.

We now illustrate the above definition. The REGEX $((_1\, \mathtt{a}^*\,)_1\, \mathtt{b}\, \backslash 1 \mid (_2\, \mathtt{a} \mid \mathtt{b}\,)_2\, \mathtt{aa}\, \backslash 2)$ is alternation confined, whereas $((_1\, \mathtt{a}^*\,)_1\, \mathtt{b}\, \backslash 1 \mid (_2\, \mathtt{a} \mid \mathtt{b}\,)_2\, \mathtt{aa}\, \backslash 2)\, \backslash 1$ is not. On the other hand, The REGEX $((\mathtt{a}^*\mathtt{bc})^* \mid (_1\, \mathtt{ac} \mid \mathtt{abb}\,)_1)\, \backslash 1$ is star-free initialised, whereas $(((_1\, \mathtt{a}^*\mathtt{b}\,)_1\, \mathtt{c})^* \mid \mathtt{ac})\, \backslash 1$ is not.

We can show that the condition of being alternation confined does not impose a restriction on the expressive power of a star-free initialised REGEX. The same holds with respect to their languages with necessarily initialised subexpressions. Furthermore, for every star-free initialised REGEX r, the language $\mathcal{L}(r)$ can also be given as the language with necessarily initialised subexpressions of a star-free initialised REGEX and vice versa. This is formally stated in the next lemma, which shall be useful for proving the main result of this section.

Lemma 7.15.

$$\mathcal{L}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}(\text{REGEX}_{\text{sfi,ac}}) = \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})\,.$$

Proof. We first note that, since $\text{REGEX}_{\text{sfi,ac}} \subseteq \text{REGEX}_{\text{sfi}}$, $\mathcal{L}(\text{REGEX}_{\text{sfi,ac}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi}})$ and $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}})$ trivially hold. Next, we observe that if a REGEX r is star-free initialised and alternation confined, then $\mathcal{L}_{\text{nis}}(r) = \mathcal{L}(r)$. This is due to the fact that if r is star-free initialised and alternation confined, then it is impossible that, while matching r to some word, a backreference occurs that points to a referenced subexpression that has not been initialised. This particularly implies $\mathcal{L}(\text{REGEX}_{\text{sfi,ac}}) = \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$. In order to conclude the proof, it is sufficient to show that $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi,ac}})$ and $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$.

In the following, we say that an alternation $(s \mid t)$ of a REGEX is *confining*, if all referenced subexpressions in s are referenced only in s and all referenced subexpressions in t are referenced only in t. Otherwise, an alternation is called *non-confining*.

We first prove that $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi,ac}})$. To this end, let r be a star-free initialised REGEX that is not alternation confined, which implies that $r := r_1 \cdot (r_2 \mid r_3) \cdot r_4$, where the alternation $(r_2 \mid r_3)$ is non-confining. For the sake of concreteness, let r_2 contain exactly the $l_1^{\text{th}}, l_2^{\text{th}}, \ldots, l_k^{\text{th}}$ referenced subexpressions and let r_3 contain exactly the $m_1^{\text{th}}, m_2^{\text{th}}, \ldots, m_n^{\text{th}}$ referenced subexpressions. We now define t_1, which is a copy of $r_1 \cdot r_2 \cdot r_4$, where all backreferences $\backslash m_i$, $1 \leq i \leq n$, have been deleted and t_2, which is a copy of $r_1 \cdot r_3 \cdot r_4$, where all backreferences $\backslash l_i$, $1 \leq i \leq k$ have been deleted. We note that t_1 and t_2 are valid REGEX and, since r is star-free initialised, $(r_2 \mid r_3)$ is not under a star in r, which implies that $\mathcal{L}(r) = \mathcal{L}(t_1) \cup \mathcal{L}(t_2)$. Next, let t'_1 and t'_2 be obtained from t_1 and t_2, respectively, by renaming all referenced subexpressions and their corresponding backreferences such that in t'_1 and t'_2 there are no referenced subexpressions that are numbered by the same number. We can note that $r' := (t'_1 \mid t'_2)$ is a valid REGEX and, since $\mathcal{L}(t'_1) = \mathcal{L}(t_1)$ and $\mathcal{L}(t'_2) = \mathcal{L}(t_2)$, $\mathcal{L}(r') = \mathcal{L}(r)$ is implied. Moreover, r' is star-free initialised, the alternation $(t'_1 \mid t'_2)$ is confining and in each t'_1 and t'_2 there is one fewer alternation compared to r. Consequently, by repeating the above construction, we can transform r into a REGEX r'' that is star-free initialised, alternation confined and $\mathcal{L}(r'') = \mathcal{L}(r)$ holds. This proves $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi,ac}})$.

Next, we prove $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$. Again, let r be a star-free initialised REGEX that is not alternation confined, which implies that $r := r_1 \cdot (r_2 \mid r_3) \cdot r_4$, where the alternation $(r_2 \mid r_3)$ is non-confining. Let $({}_{p_1}\ r_{p_1}\ )_{p_1}$, $({}_{p_2}\ r_{p_2}\ )_{p_2}$, …, $({}_{p_k}\ r_{p_k}\ )_{p_k}$ be exactly the referenced subexpressions in r_3 and we assume them to be ordered with respect to their nesting, i. e., for every i, j, $1 \leq i < j \leq k$, if the p_i^{th} referenced subexpression occurs in the p_j^{th} referenced subexpression, then $p_j \leq p_i$ is implied. We now obtain s' from $r_1 \cdot r_2 \cdot r_4$ in the following way. We first substitute the leftmost occurrence of $\backslash p_1$ by $({}_{p_1}\ r_{p_1}\ )_{p_1}$. Next, if there does not already exist an occurrence of $({}_{p_2}\ r_{p_2}\ )_{p_2}$ (which might be the case if $({}_{p_2}\ r_{p_2}\ )_{p_2}$ is contained in r_{p_1}), then we substitute the leftmost occurrence of $\backslash p_2$ by $(_{p_2}\ r_{p_2}\ )_{p_2}$. This step is then repeated with respect to the referenced subexpressions $(_{p_3}\ r_{p_3}\ )_{p_3}, \ldots, (_{p_k}\ r_{p_k}\ )_{p_k}$. We observe that, for every i, $1 \leq i \leq k$, there is at most one occurrence of $(_{p_i}\ r_{p_i}\ )_{p_i}$ in s' and if there exists a backreference $\backslash p_i$, then it occurs to the right of $(_{p_i}\ r_{p_i}\ )_{p_i}$. This implies that s' is a valid REGEX. Next, we transform $r_1 \cdot r_3 \cdot r_4$ into t' in the same way, just with respect to the referenced subexpressions in r_2. Finally, s and t are obtained from s' and t', respectively, by renaming all referenced subexpressions and the corresponding backreferences in such a way that s and t do not have any referenced subexpressions labeled by the same number. We define $r' := (s \mid t)$ and we can note that r' is a valid star-free initialised REGEX. Furthermore, since r is star-free initialised, $(r_2 \mid r_3)$ is not under a star in r, which implies that $\mathcal{L}_{\text{nis}}(r') = \mathcal{L}_{\text{nis}}(r)$. We further note that the alternation $(s \mid t)$ is confining and in each s and t there is one fewer alternation compared to r. This implies that by successively applying the above transformation now to s and t and so on, r can be transformed into a star-free initialised REGEX r'' that is also alternation confined and $\mathcal{L}_{\text{nis}}(r'') = \mathcal{L}_{\text{nis}}(r)$. This proves $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$. □

In the following, we take a closer look at the task of transforming a pattern expression p into a REGEX r, such that $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}(r)$. Although, this is possible in general, a few difficulties arise, that have already been pointed out by Câmpeanu and Yu in [12] (with respect to $\mathcal{L}_{\text{it}}(p)$).

The natural way to transform a pattern expression into an equivalent REGEX is to successively substitute the occurrences of variables by referenced subexpressions and appropriate backreferences. However, this is not always possible. For example, consider the pattern expression $q := (x_1 \to (\mathtt{a} \mid \mathtt{b})^*, x_2 \to x_1^* \cdot \mathtt{c} \cdot x_1 \cdot \mathtt{d} \cdot x_1)$. If we simply transform q into $r_q := (_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1^* \cdot \mathtt{c} \cdot \backslash 1 \cdot \mathtt{d} \cdot \backslash 1$, then we obtain an incorrect REGEX, since $\mathcal{L}_{\text{uni}}(q) \neq \mathcal{L}(r_q)$. This is due to the fact that the referenced subexpression is under a star. To avoid this, we can first rewrite q to $q' := (x_1 \to (\mathtt{a} \mid \mathtt{b})^*, x_2 \to (x_1 \cdot x_1^* \mid \varepsilon) \cdot \mathtt{c} \cdot x_1 \cdot \mathtt{d} \cdot x_1)$, which leads to $r_{q'} := ((_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1 \cdot (\backslash 1)^* \mid \varepsilon) \cdot \mathtt{c} \cdot \backslash 1 \cdot \mathtt{d} \cdot \backslash 1$. Now we encounter a different problem: $\mathcal{L}_{\text{uni}}(q')$ contains the word `cabadaba`, but in $\mathcal{L}(r_{q'})$ the only word that starts with `c` is `cd`. This is due to the fact that if we choose the second option of $((_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1 \cdot (\backslash 1)^* \mid \varepsilon)$, then all $\backslash 1$ are set to the empty word. However, we note that the language with necessarily initialised subexpressions of $r_{q'}$ is exactly what we want, since $\mathcal{L}_{\text{nis}}(r_{q'}) = \mathcal{L}_{\text{uni}}(q)$. Hence, we can transform any pattern expression p to a REGEX r_p that is star-free initialised and $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}_{\text{nis}}(r_p)$.

Lemma 7.16. *For every pattern expression p, there exists a star-free initialised* REGEX r *with* $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}_{\text{nis}}(r)$.

Proof. Let $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_m \to r_m)$ be an arbitrary pattern expression. Now we assume that, for some i, $1 \leq i \leq m$, r_i contains a subexpression $(q)^*$, where $(q)^*$ is not under a star and q contains the leftmost occurrence of a variable. We can obtain r''_i from r_i by substituting $(q)^*$ by $(q \cdot (q)^* \mid \varepsilon)$. It can be easily verified that $\mathcal{L}(r_i) = \mathcal{L}(r''_i)$. Furthermore, we can repeat this step until we obtain an r'_i from r_i such that $\mathcal{L}(r_i) = \mathcal{L}(r'_i)$ and the leftmost occurrence of any variable in r'_i does not occur under a star. By applying this construction to every r_i, $1 \leq i \leq m$, we can transform p into a pattern expression $p' := (x_1 \to r'_1, x_2 \to r'_2, \ldots, x_m \to r'_m)$, where, for every i, $1 \leq i \leq m$, the leftmost occurrence of any variable in r'_i does not occur under a star. Furthermore, for every i, $1 \leq i \leq m$, $\mathcal{L}(r_i) = \mathcal{L}(r'_i)$, which implies $\mathcal{L}_{\text{uni}}(p') = \mathcal{L}_{\text{uni}}(p)$.

Next, we construct a REGEX t with $\mathcal{L}_{\text{nis}}(t) = \mathcal{L}_{\text{uni}}(p')$ in the following way. First, we transform r_m into t_{m-1} by substituting the leftmost occurrence of x_{m-1} by $(_{m-1}\ r_{m-1}\ )_{m-1}$ and all other occurrences of x_{m-1} by $\backslash m-1$. Since we can assume that there is at least one occurrence of x_{m-2} in r_m or in r_{m-1}, we can conclude that in t_{m-1} there is at least one occurrence of variable x_{m-2}. Next, we obtain t_{m-2} from t_{m-1} by substituting the leftmost occurrence of x_{m-2} by $(_{m-2}\ r_{m-2}\ )_{m-2}$ and all other occurrences of x_{m-1} by $\backslash m-1$. In the same way as before, we can conclude that in t_{m-2} there exists at least one occurrence of variable x_{m-3}. This procedure is now repeated until we obtain t_1 and we observe that in t_1 there is no occurrence of a variable, for every i, $1 \leq i \leq m-1$, there is exactly one subexpression labeled by i and all occurrence of $\backslash i$ occur to the right of this subexpression. Consequently, t_1 is a valid REGEX. Moreover, since, for every i, $1 \leq i \leq m$, the leftmost occurrence of any variable in r'_i does not occur under a star, we can conclude that t_1 is star-free initialised. For the sake of convenience, we shall call t_1 simply t.

It remains to show that $\mathcal{L}_{\text{nis}}(t) = \mathcal{L}_{\text{uni}}(p')$ holds. Let $(_1\ s_1\ )_1$, $(_2\ s_2\ )_2$, $\ldots, (_{m-1}\ s_{m-1}\ )_{m-1}$ be the referenced subexpressions in t. By definition of t, for every i, j, $1 \leq i < j \leq m-1$, the j^{th} referenced subexpression does not occur in the i^{th} referenced subexpression. This particularly implies that s_1 is a classical regular expression. Now let s be a classical regular expression that is obtained from t in the following way: We substitute the first referenced subexpression and all backreferences $\backslash 1$ by some word $u_1 \in \mathcal{L}(s_1)$ (as explained above s_1 is a classical regular expression). After that, we substitute the second referenced subexpression and all backreferences $\backslash 2$ by some word $u_2 \in \mathcal{L}(s'_2)$, where s'_2 is a classical regular expression that is obtained from s_2 by substituting the first referenced subexpression and all backreferences $\backslash 1$ by u_1. This step is then repeated, i. e., for every $3 \leq i \leq m-1$, we substitute the i^{th} referenced subexpression and all backreferences $\backslash i$ by some word $u_i \in \mathcal{L}(s'_i)$, where s'_i is a classical regular expression that is obtained from s_i by substituting, for every j, $1 \leq j < i$, the j^{th} referenced subexpression and all backreferences $\backslash j$ by u_j. We note that in a similar way, we can also transform p' into s, i. e., by substituting all occurrences of variable x_1 in the elements r'_i, $2 \leq i \leq m$, by the word $u_1 \in \mathcal{L}_{\text{uni}}(r'_1)$ and then we repeat this step with respect to variables $x_2, x_3, \ldots, x_m$ and words $u_2, u_3, \ldots, u_m$. Moreover, every classical regular expression s that can be obtained in this way from p' can also be obtained from t by the above described construction.

By definition of the language of a pattern expression with respect to uniform substitution, $\mathcal{L}_{\text{uni}}(p')$ is the union of all $\mathcal{L}(s)$, where s is a classical regular expression that can be obtained from p' in the above described way. Moreover, since t is star-free initialised, $\mathcal{L}_{\text{nis}}(t)$ is the union of all $\mathcal{L}(s)$, where s is a classical regular expression that can be obtained from t in the above described way. We note that this is only true since we consider the language with necessarily initialised subexpressions of t. This directly implies that $\mathcal{L}_{\text{uni}}(p') = \mathcal{L}_{\text{nis}}(t)$, which concludes the proof. $\square$

We recall that Lemma 7.15 states that every star-free initialised REGEX r can be transformed into a star-free initialised REGEX r' with $\mathcal{L}_{\text{nis}}(r) = \mathcal{L}(r')$. Consequently, Lemmas 7.15 and 7.16 imply that every pattern expression p can be transformed into a star-free initialised REGEX r with $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}(r)$. For example, the pattern expression q introduced on page 196 can be transformed into the REGEX $t_q := ((_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1 \cdot (\backslash 1)^* \cdot \mathtt{c} \cdot \backslash 1 \cdot \mathtt{d} \cdot \backslash 1 \mid \mathtt{c} \cdot (_2\ (\mathtt{a} \mid \mathtt{b})^*\ )_2 \cdot \mathtt{d} \cdot \backslash 2)$, which finally satisfies $\mathcal{L}_{\text{uni}}(q) = \mathcal{L}(t_q)$.

Theorem 7.17. $\mathcal{L}_{\text{uni}}(\text{PE}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi}})$.

Proof. We can note that Lemma 7.16 implies $\mathcal{L}_{\text{uni}}(\text{PE}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}})$ and Lemma 7.15 states $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi}})$. Consequently, $\mathcal{L}_{\text{uni}}(\text{PE}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi}})$. $\square$

In the remainder of this section, we show the converse of Theorem 7.17, i. e., every star-free initialised REGEX r can be transformed into a pattern expression that describes the language $\mathcal{L}(r)$ with respect to uniform substitution. However, this cannot be done directly if r is not alternation confined. As an example, we consider $r := ((_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1 \mid (_2\ \mathtt{c}^*\ )_2) \cdot (\backslash 1)^* \cdot \backslash 2$. Now the natural way to transform r into a pattern expression is to substitute the first and second referenced subexpression and the corresponding backreferences by variables x_1 and x_2, respectively, and to introduce elements $x_1 \to (\mathtt{a} \mid \mathtt{b})$ and $x_2 \to \mathtt{c}^*$, i. e., $p_r := (x_1 \to (\mathtt{a} \mid \mathtt{b}), x_2 \to \mathtt{c}^*, x_3 \to (x_1 \mid x_2) \cdot (x_1)^* \cdot x_2)$. Now $\mathcal{L}_{\text{uni}}(p_r)$ contains the word $\mathtt{cccabababccc}$, whereas every word in $\mathcal{L}(r)$ that starts with $\mathtt{c}$ does not contain any occurrence of $\mathtt{a}$ or $\mathtt{b}$, thus, $\mathcal{L}_{\text{uni}}(p_r) \neq \mathcal{L}(r)$. So in order to transform star-free initialised REGEX into equivalent pattern expressions, again Lemma 7.15 is very helpful, which states that we can transform every star-free initialised REGEX into an equivalent one that is also alternation confined.

Theorem 7.18. $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{uni}}(\text{PE})$.

Proof. In order to prove the statement of the theorem, we shall use a combination of pattern expressions and REGEX, i. e., pattern expressions $p := (x_1 \rightarrow r_1, x_2 \rightarrow r_2, \ldots, x_m \rightarrow r_m)$, where r_m is not a pattern with regular operators, but a REGEX with possible occurrences of variables $x_1, x_2, \ldots, x_{m-1}$. The language $\mathcal{L}_{\text{uni}}(p)$ is then defined in a similar way as for standard pattern expressions, i. e., it is the union of all $\mathcal{L}(r)$, where r is a REGEX (without variables), that can be obtained from p in the following way. We first choose a word $u \in \mathcal{L}(r_1)$ and, for all i, $1 \leq i \leq m$, if variable x_1 occurs in r_i, then we substitute all occurrences of x_1 in r_i by u and delete the element $x_1 \rightarrow r_1$ from the pattern expression. This step is then repeated with respect to the variables $x_2, x_3, \ldots, x_{m-1}$ until we obtain a REGEX.

Now let $p := (x_1 \rightarrow r_1, x_2 \rightarrow r_2, \ldots, x_m \rightarrow r_m)$ be an arbitrary such pattern expression, where r_m is a star-free initialised and alternation confined REGEX. Furthermore, let the i^{th} referenced subexpression in r be $(_i\ q_i\ )_i$, where q_i is a classical regular expression. Obviously, there must exist at least one such referenced subexpression. We note that q_i may contain variable symbols and we assume that x_j occurs in q_i and, for every l, $j < l \leq m-1$, x_l does not occur in q_i. Now, we transform p into $p' := (x_1 \rightarrow r_1, \ldots, x_j \rightarrow r_j, z \rightarrow q_i, x_{j+1} \rightarrow r_{j+1}, \ldots, x_m \rightarrow r'_m)$, where r'_m is obtained from r_m by substituting $(_i\ q_i\ )_i$ and all occurrences of $\backslash i$ by z. If q_i does not contain a variable, then $p' := (z \rightarrow q_i, x_1 \rightarrow r_1, \ldots, x_m \rightarrow r'_m)$. We can observe, that since r_m is alternation confined and star-free initialised, by applying the above described method, we can obtain exactly the same REGEX from p and p', which implies that $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}_{\text{uni}}(p')$.

By successively applying this construction, we can transform an arbitrary star-free initialised and alternation confined REGEX r into a pattern expression p with $\mathcal{L}(r) = \mathcal{L}_{\text{uni}}(p)$. Since, by Lemma 7.15, every star-free initialised REGEX can be transformed into an equivalent one that is also alternation confined, $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{uni}}(\text{PE})$ follows. □

From Theorems 7.17 and 7.18, we can conclude that the class of languages described by pattern expressions with respect to uniform substitution coincides with the class of languages given by regular expressions that are star-free initialised.

Corollary 7.19. $\mathcal{L}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}_{\text{uni}}(\text{PE})$.

The previous result in particular shows that pattern expressions can be used in order to describe a large class of REGEX languages, i. e., the class of languages given by star-free initialised REGEX. Since pattern expressions have a simple and clear structure (in fact, syntactically, they are tuples of regular expressions) we conjecture that, in many cases, they provide a more suitable means to describe star-free initialised REGEX languages and they particularly represent the nested structure of backreferences more clearly.

In Sections 7.1 and 7.2 and in the present section, we have investigated several proper subclasses of the class of REGEX languages and their mutual relations, which can be summarised in the following way:

$$\begin{aligned}&\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}) \subset \mathfrak{L}_{\mathrm{ro},1} \subset \mathfrak{L}_{\mathrm{ro},2} \subseteq \mathfrak{L}_{\mathrm{ro},3} \subseteq \ldots \subseteq \mathfrak{L}_{\mathrm{ro},\infty} = \\ &\mathcal{H}^*(\mathrm{REG}, \mathrm{REG}) = \mathcal{L}_{\mathrm{it}}(\mathrm{PE}) \subset \mathcal{L}_{\mathrm{uni}}(\mathrm{PE}) = \mathcal{L}(\mathrm{REGEX}_{\mathrm{sfi}}) \subseteq \mathcal{L}(\mathrm{REGEX})\,.\end{aligned}$$

We conclude this section by discussing these relations. First, we can note that, indicated by $\mathfrak{L}_{\mathrm{ro},\infty} = \mathcal{H}^*(\mathrm{REG}, \mathrm{REG}) = \mathcal{L}_{\mathrm{it}}(\mathrm{PE}) \subset \mathcal{L}(\mathrm{REGEX})$, several natural ways to combine regular expressions with homomorphic substitution do not lead to the class of REGEX languages, but to the substantially smaller class of languages that are described by pattern expressions with respect to iterated substitution. The relation $\mathcal{L}_{\mathrm{it}}(\mathrm{PE}) \subset \mathcal{L}_{\mathrm{uni}}(\mathrm{PE})$ demonstrates that the lack of expressive power of language generating devices such as typed patterns with regular operators (see Section 7.1), iterated H-systems (see Albert and Wegner [2] and Bordihn et al. [9]) and pattern expressions with respect to iterated substitution (see Section 7.2 and Câmpeanu and Yu [12]) seems to be caused by their lack of handling the nested structure of backreferences.

7.4 REGEX with a Bounded Number of Backreferences

Since the class of pattern languages is properly included in the class of REGEX languages, it is not surprising that the membership problem for REGEX languages is NP-complete, which is formally proved by Aho [1]. It can be easily shown that the membership problem for pattern languages can be solved by a naive algorithm in time that is exponential only in the number of different variables (see Section 2.2.2.1). Hence, the membership problem for patterns with a bounded number of variables can be solved in polynomial time and Aho claims that the same holds for REGEX with respect to the number of backreferences. More precisely, in Aho [1] it is stated that the membership problem for REGEX languages can be solved in time that is exponential only in the number of backreferences in the following way. Let k be the number of referenced subexpressions in a REGEX r and let w be an input word. We first choose k factors $u_1, u_2, \dots, u_k$ of w and then try to match r to w in such a way that, for every i, $1 \leq i \leq k$, the i^{th} referenced subexpression is matched to u_i. This is done with respect to all possible k factors of w. For this procedure we only need to keep track of the k possible factors of w, thus, time $\mathrm{O}(|w|^{2k})$ is sufficient. However, this approach is incorrect (which is a known fact in the language theory community), since it ignores the possibility that the referenced subexpressions under a star (and their backreferences) can be matched to a different factor in every individual iteration of the star. On the other hand, if we first iterate every expression under a star that contains a referenced subexpression an arbitrary number of times, then, due to the late binding of backreferences, we introduce arbitrarily many new referenced subexpressions and backreferences, so there is an arbitrary number of factors to keep track of.

The question of whether or not the membership problem for REGEX with a bounded number of backreferences can be solved in polynomial time is arguably the first question that comes to mind when we try to identify subclasses of REGEX with a polynomial time membership problem. Consequently, an answer to this question is of considerable importance.

We give a positive answer to that question, by showing that for any REGEX r, a nondeterministic two-way multi-head automaton (see Section 2.3.1) can be constructed that accepts exactly $\mathcal{L}(r)$ with a number of input heads that is bounded by the number of referenced subexpressions in r and a number of states that is bounded by the length of r.

Lemma 7.20. *Let r be a* REGEX *with k referenced subexpressions. There exists a nondeterministic two-way $(3k+2)$-head automaton with* $\mathrm{O}(|r|)$ *states that accepts $\mathcal{L}(r)$.*

Proof. We assume that r is completely parenthesised. We shall now define a nondeterministic two-way $(2 + 3k)$-head automaton M that accepts $\mathcal{L}(r)$. This automaton uses $2k$ input heads in order to implement k individual counters as described in Observation 2.11 (page 21). One input head is the *main head*, another one is the *auxiliary head* and the remaining k heads are enumerated from 1 to k. Initially, all input heads are located at the left endmarker. The finite state control contains a special pointer, referred to as the *r-pointer*, that is initially located at the left end of r. In the computation of M, this r-pointer is moved over r and every time it enters or leaves a new subexpression, i. e., whenever it is moved over a left or right parenthesis, respectively, a sequence of operations is triggered. Before we define these operations more precisely, we shall informally explain how

M accepts words from $\mathcal{L}(r)$ and how it handles backreferences.

The main head is moved over the input from left to right, checking whether or not the input word satisfies r, just as it is done by a classical nondeterministic finite automaton that accepts the language given by a classical regular expression. Simultaneously, the r-pointer is moved over r. When the r-pointer enters the referenced subexpression i, then we move head i to the position of the main head, we start counting every following step of the main head on counter i and we stop counting as soon as the r-pointer has left the referenced subexpression i. This means that we store the length of the factor that has been matched to the referenced subexpression i in counter i, whereas head i now scans the position where this factor starts. Now if the r-pointer encounters a backreference $\backslash i$, it is checked whether or not at the positions scanned by the main head and head i the same factor occurs with the length stored by counter i. It is also possible that $\backslash i$ is encountered without having visited the referenced subexpression i. In this case, counter i stores 0, which means that $\backslash i$ is treated as the empty word. If the r-pointer encounters the referenced subexpression i for a second time, which is possible since it can occur under a star, then counter i and head i are simply reset and then the referenced subexpression i is handled in exactly the same way as before. This ensures that in different iterations of a star every referenced subexpression is treated individually and only the factor that is matched to it in the very last iteration is stored for future backreferences.

We are now ready to define the operations that M performs when the r-pointer enters or leaves a subexpression. In the following definitions, we say that the r-pointer visits a subexpression if it is located somewhere between its delimiting parenthesis.

We assume that the r-pointer has just entered a new subexpression r'. If r' is the i^{th} referenced subexpression, then we set counter i to zero and we move head i to the position of the main head, which can be done by using the auxiliary head. If, on the other hand, r' is not a referenced subexpression, we simply skip the aforementioned operations. Next, we perform the operations listed below and we define $p_1, p_2, \ldots, p_m$ to be exactly the referenced subexpressions the r-pointer visits at the moment.

- If $r' = (\varepsilon)$, then the r-pointer leaves r'.
- If $r' = (a)$, for some $a \in \Sigma$, then we move the main head a step to the right and reject the input if the symbol on that new position does not equal a. Furthermore, for every i, $1 \leq i \leq m$, counter p_i is incremented. Finally, the r-pointer leaves r'.
- If $r' = (s \mid s')$, then we nondeterministically choose to enter either s or s'

with the r-pointer.

- If $r' = (s)^*$, then we enter s with the r-pointer.
- If $r' = (\backslash i)$, then we move the main head and head i simultaneously to the right for l steps, where l is the value of counter i, and reject the input if they do not scan the same symbols in every step. After that, head i is moved back to the left for l steps and, for every j, $1 \leq j \leq m$, counter p_j is incremented by l. If in this procedure the main head is moved over the right endmarker, then the input is rejected. Finally, the r-pointer leaves r'.

If the r-pointer leaves a subexpression r', then the following operations are performed.

- If r' is not followed by a star or by symbol "|", then we move the r-pointer over the next parenthesis.
- If r' is followed by symbol "|", then we can conclude that some subexpression s follows that is followed by a right parenthesis. In this case we move the r-pointer completely over the part "$|\ s)$".
- If r' is followed by a star, then we nondeterministically choose to move the r-pointer over the next parenthesis or to re-enter subexpression r'.
- If the r-pointer has reached the end of r and the main head scans the right endmarker, then M accepts its input. If, on the other hand, the main head does not scan the right endmarker, then the input is rejected.

It can be easily verified that a word is in $\mathcal{L}(r)$ if and only if it is possible that M accepts that word.

Since the finite state control only needs to keep track of the position of the r-pointer, a number of $\mathrm{O}(|r|)$ states are sufficient. $\square$

In the above proof, we use a number of input heads of a nondeterministic two-way multi-head automaton as mere counters in order to keep track of lengths of factors. This corresponds to the way we use the modulo counters of Janus automata in order to recognise pattern languages (see Section 3.3). However, the 2NFA that is used in the proof of Lemma 7.20 also uses several input heads for scanning the input and, thus, in the context of this proof a 2NFA is more convenient than an NBMCA or a Janus automaton.

Since we can solve the acceptance problem of a given two-way multi-head automaton M and a given word w in time that is exponential only in the number of input heads, we can conclude the following result:

Theorem 7.21. *Let $k \in \mathbb{N}$. The membership problem for* REGEX *with at most k referenced subexpressions can be solved in polynomial time.*

Proof. Let r be a REGEX with k referenced subexpressions and let w be an arbitrary word. By the proof of Lemma 7.20, we can transform r into a $(3k+2)$-head automaton M_r that accepts exactly $\mathcal{L}(r)$. Furthermore, this transformation can be done in polynomial time and M_r has $\mathrm{O}(|r|)$ states. We can check whether or not w is accepted by M_r in the following way. We interpret every possible configuration of M_r on input w as a vertex of a graph $\mathcal{G}$ and there is a directed edge from a vertex c to a vertex c' if and only if M_r can change from the configuration c to the configuration c'. Now w is accepted by M_r if and only if there exists a path in $\mathcal{G}$ from the start configuration to an accepting configuration. This can be checked in time linear in the size of $\mathcal{G}$. Since there are at most $\mathrm{O}(|r| \times |w|^{3k+2})$ configurations of M_r on input w, we can conclude that the size of $\mathcal{G}$ is $\mathrm{O}((|r| \times |w|^{3k+2})^2)$. This implies that we can decide in polynomial time on whether or not $w \in \mathcal{L}(r)$. $\square$

Consequently, the above result generalises the polynomial time solvability of the membership problem for patterns with a bounded number of variables to the class of REGEX with a bounded number of referenced subexpressions, which constitutes a possible starting point for further research on the complexity of the membership problem for REGEX languages. It is particularly worth mentioning that while it is trivial to show that the membership problem for patterns with a bounded number of variables can be solved in polynomial time, it requires some more effort to show the analogue with respect to REGEX and the number of referenced subexpressions. We conjecture that identifying more complicated structural parameters of REGEX that, if restricted, yield a polynomial time membership problem is more challenging than for pattern languages.

Chapter 8

Conclusions and Discussion

We first summarise the content of this thesis chapter by chapter (except for Chapters 1 and 2) in Section 8.1 and then, in Section 8.2, we provide a more general discussion of our main results. In Section 8.3, we investigate some ideas to generalise our results, which are left for future research.

8.1 Summary of the Thesis

Chapter 3

In this chapter, we use finite automata as a tool to solve the membership problem for pattern languages. In Section 3.1, we give an overview of how pattern languages can be recognised by multi-head automata. More precisely, we show how nondeterministic two-way, nondeterministic one-way and deterministic two-way multi-head automata can recognise pattern languages, while we are not able to show whether or not pattern languages can be recognised by deterministic one-way multi-head automata. Based on these considerations, we introduce a new automata model in Section 3.2, the nondeterministically bounded modulo counter automata (NBMCA) and, as a more specialised version of NBMCA that is tailored to the following application, we introduce the Janus automata.

Section 3.3 contains our first approach to the task of finding classes of patterns for which the membership problem can be solved efficiently and, in this regard, our main result states that any class of patterns with a bounded variable distance provides this property. Our proof technique is based on Janus automata, and we also show that, under a natural assumption, this approach is optimal. Nevertheless, it is briefly outlined how our automata based approach can be improved and it is shown that these improvements lead to substantial technical difficulties. Moreover, we point out that for further improvements it is crucial to harmonise the movements of the two input heads of Janus automata in a clever way, which leads to a scheduling problem that we investigate on its own in Section 3.4. This scheduling problem can also be stated as the problem of computing shuffle words with minimum scope coincidence degree. We present a polynomial time algorithm for this problem.

Chapter 4

Section 4.1 is devoted to a thorough investigation of the model of NBMCA, which, in its more specialised variant of the Janus automata, has been proved to be a useful tool in the context of Chapter 3. Our main research questions concern expressive power, decidability questions and stateless variants of NBMCA. We first show that NBMCA can be simulated by classical nondeterministic two-way multi-head automata (2NFA) and vice versa. The simulation of NBMCA by 2NFA is straightforward, whereas the simulation of 2NFA by NBMCA involves some technical hassle, mainly because this simulation is quite economical with respect to the numbers of required counters, which allows us to conclude a hierarchy result for the language classes defined by NBMCA. More precisely, it is shown that the class of languages corresponding to NBMCA with k counters is properly included in the class of languages corresponding to NBMCA with $k + 2$ counters.

Since all interesting problems are undecidable for the unrestricted class of NBMCA, we consider NBMCA for which the input head reversals, counter reversals and counter resets are bounded by a constant. It is shown that for this class of NBMCA the emptiness, infiniteness and disjointness problem are decidable. However, if only the counter reversals are unbounded, then again all interesting problems are undecidable.

For stateless variants of NBMCA, we show that a finite state control can be simulated by a number of counters, which implies that stateless NBMCA can simulate NBMCA with a finite state control. The technical challenges for such simulations are caused by the strong restriction of the modulo counters. Furthermore, for a very restricted version of stateless NBMCA, i. e., stateless one-way NBMCA with only one counter, which can be reset only a constant number of k times (1SL-NBMCA$_k$(1)), it is shown that there are languages that can be recognised by some 1SL-NBMCA$_k$(1), but no 1SL-NBMCA$_{k'}$(1), $k' \neq k$, can recognise the same language.

In Section 4.2, we introduce and investigate the nondeterministically initialised multi-head automata (IFA), which form a special variant of multi-head automata. These IFA work similarly to classical deterministic two-way multi-head automata, but their input heads are initially nondeterministically distributed over the input word. This model is motivated by how multi-head automata can recognise pattern languages, described in Section 3.1. It is shown that IFA can be determinised, i. e., they have the same expressive power as deterministic two-way multi-head automata. As an immediate result it follows that pattern languages are in DL, the class of languages that can be recognised in deterministic logarithmic space.

Chapter 5

This chapter contains our second approach to the task of finding classes of patterns for which the membership problem can be solved efficiently. The two respective main results of this chapter are that any class of patterns with a bounded scope coincidence degree and the class of mildly entwined patterns have a polynomial time membership problem. Our proof technique differs quite substantially from the automata based approach of Chapter 3 and is much more general. In Section 5.1, we first introduce a way to encode patterns and words into relational structures and then we show that these encodings constitute a reduction of the membership problem for pattern languages to the homomorphism problem for relational structures. This allows us to apply the concept of the treewidth to these relational structures, which results in a meta-theorem stating that if a class of patterns satisfies that the treewidth of the corresponding encodings as relational structures is bounded by a constant, then the membership problem with respect to this class of patterns can be solved in polynomial time. The two main results mentioned above, which are presented in Sections 5.2 and 5.3, respectively, are direct applications of this meta-theorem. It is also briefly outlined how our meta-theorem could be used to identify further classes of patterns with a membership problem that can be solved in polynomial time.

Chapter 6

In this chapter, we investigate the phenomenon that, with respect to alphabets of size 2 or 3, patterns can describe regular or context-free languages in an unexpected way. This particularly implies that for these patterns it is substantially easier to solve the membership problem. In Section 6.1, we give an overview of what is currently known about this phenomenon. Then, in Section 6.2, we provide some strong necessary conditions for the regularity of pattern languages and we give numerous examples that demonstrate the hardness of finding characterisations of the regular pattern languages, with respect to alphabets of size 2 and 3. A necessary condition for the regularity of E-pattern languages over an alphabet of size 2, which particularly takes terminal symbols into account, is given in Section 6.3.

Chapter 7

In Chapter 7, we investigate several possibilities to extend pattern languages with regular languages and regular expressions, in order to describe subclasses of the class of REGEX languages. We consider several classes of typed pattern languages in Section 7.1 and languages given by pattern expressions in Section 7.2. We extend the original definition of how pattern expressions describe languages and it is shown that our versions of typed pattern languages coincide with the class of languages given by pattern expressions. On the other hand, our refined versions of pattern expressions are strictly more powerful and in Section 7.3 it is shown that they coincide with the class of languages that are given by REGEX that do not contain referenced subexpressions under a star. In Section 7.4, we prove that the membership problem for languages that are given by REGEX with a bounded number of backreferences can be solved in polynomial time.

8.2 Discussion of the Results

With respect to the membership problem of pattern languages, the presented classes of patterns for which the membership problem can be solved efficiently are the most important results. It is convenient to state these results in form of the structural properties or parameters that need to be restricted in order to obtain a polynomial time membership problem. In this regard, we identify the parameters of the variable distance and the scope coincidence degree as well as the property for patterns of being mildly entwined as substantially contributing to the complexity of the membership problem for pattern languages. These results constitute much deeper insights into the complexity of the membership problem than the ones provided by the parameter of the number of variables or the properties for patterns of being regular or non-cross.

As mentioned in Section 2.2.1 (Lemma 2.5), the scope coincidence degree is a lower bound for the variable distance, which implies that any class of patterns with a bounded variable distance also has a bounded scope coincidence degree. Consequently, the result that the membership problem can be solved efficiently if the scope coincidence degree is bounded (see Theorem 5.10, page 146) implies that the membership problem can be solved efficiently if the variable distance is bounded (see Theorem 3.20, page 54). However, the conceptual aspect of the results given in Chapter 3, i. e., the automaton based approach, is not covered by the results of Chapter 5. Moreover, while for our Janus automaton further improvements, e. g., an extension to regular-typed pattern languages, as described at the end of Section 3.3.2, can be easily implemented, it is not straightforward to see how the encodings of patterns and words as relational structures given in Chapter 5 need to be modified in order to accommodate similar amendments. This aspect shall be discussed in more detail in the following section.

8.3 Further Research Ideas

Regarding our two main approaches presented in Chapters 3 and 5, we have already outlined possible improvements and we have mentioned some open problems in Sections 3.3.3 and 5.4, respectively.

From a more applied point of view, the probably most promising research task is to find a way to generalise the results presented in Chapters 3 and 5 to extended regular expressions with backreferences. In the following, we wish to investigate this task a little further.

We call a REGEX r *unnested* if and only if r does not contain a referenced subexpression within a referenced subexpression. This implies that in an unnested REGEX, every backreference points to a classical regular expressions. Hence, in terms of pattern languages, *unnested* REGEX languages correspond to the class of REG-typed pattern languages ($\mathcal{L}_{\text{REG}}(\text{PAT})$) introduced in Chapter 7 (Definition 7.1, page 183). It is straightforward to generalise the transformation of patterns into Janus automata presented in Chapter 3 (Theorem 3.10, page 38) to unnested REGEX. Informally speaking, this can be done by using the finite state control of the Janus automaton in order to check whether or not the factors that are matched are members of a regular language.

Generalising the approach of encoding patterns and words as relational structures, described in Chapter 5, to the class of unnested REGEX is not as straightforward. The problem is that we have to refine the encodings given in Definitions 5.1 and 5.3 in the following way. Let $\mathcal{A}_\alpha$ be an α-structure for some pattern α and let $\mathcal{A}_w$ be the NE-w-structure for some word w. Furthermore, let A_α and A_w be the universes of $\mathcal{A}_\alpha$ and $\mathcal{A}_w$, respectively. We now interpret α as an unnested REGEX, i. e., we simply add a regular type $\mathcal{T} := (T_{x_1}, T_{x_2}, \dots, T_{x_m})$ to α. Obviously, it is now possible that there exists a homomorphism from $\mathcal{A}_\alpha$ and $\mathcal{A}_w$, but $w \notin \mathcal{L}_{\mathcal{T}}(\alpha)$. Consequently, in order to reduce the membership problem for nested REGEX languages to the homomorphism problem for relational structures, we have to make sure that a homomorphism can map an element $i \in A_\alpha$ to an element $(j, j') \in A_w$ only if $w[j, j'] \in T_{x_{\alpha[i]}}$. This could be achieved by introducing, for every i, $1 \leq i \leq m$, a unary relation symbol $\widehat{T}_{x_i}$ that is interpreted in the following way: $\widehat{T}_{x_i}^{A_\alpha} := \{j \mid \alpha[j] = x_i\}$ and $\widehat{T}_{x_i}^{A_w} := \{(j, j') \mid w[j, j'] \in T_{x_i}\}$. However, this requires an unbounded number of relation symbols, which contradicts the definition of the homomorphism problem for relational structures.

A simpler way to generalise the results of Chapter 5 is to modify the algorithm described in the proof of Theorem 5.11 (page 147) that solves the membership problem with respect to patterns with a bounded scope coincidence degree in the following way. In the step where, for every i, $1 \leq i \leq |\alpha|$, we inductively compute the set H_i, we do not only check for every tuple C of size $|B_i|$ containing elements from A_w whether or not the mapping $\mathrm{ord}(B_i) \mapsto C$ satisfies condition $(*)$ (defined on page 147) and the set H_{i-1} contains a tuple C' such that the mappings $\mathrm{ord}(B_i) \mapsto C$ and $B_{i-1} \mapsto C'$ are compatible, but also whether or not the mappings $\mathrm{ord}(B_i) \mapsto C$ and $B_{i-1} \mapsto C'$ map an element $i \in A_\alpha$ to an element $(j, j') \in A_w$ only if $w[j, j'] \in T_{x_{\alpha[i]}}$. This can be done by checking the membership of a word to a regular language.

Consequently, we can conclude that our two main approaches to the membership problem for pattern languages can be generalised to the membership problem for REG-typed pattern languages (or unnested REGEX languages) with little effort. Thus, the difficult task is to achieve a generalisation to the full class of REGEX languages. We anticipate that for both, the automaton based approach and the reduction to the homomorphism problem for relational structures, this is not trivial.

References

[1] A. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, pages 255–300. MIT Press, 1990.

[2] J. Albert and L. Wegner. Languages with homomorphic replacements. *Theoretical Computer Science*, 16:291–305, 1981.

[3] A. Amir, Y. Aumann, R. Cole, M. Lewenstein, and E. Porat. Function matching: Algorithms, applications, and a lower bound. In *Proc. 30th International Colloquium on Automata, Languages and Programming, ICALP 2003*, pages 929–942, 2003.

[4] A. Amir and I. Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.

[5] D. Angluin. Finding patterns common to a set of strings. In *Proc. 11th Annual ACM Symposium on Theory of Computing*, pages 130–141, 1979.

[6] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[7] B. S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52:28–42, 1996.

[8] H.L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report RUU-CS-86-22, Department of Information and Computing Sciences, Utrecht University, 1986.

[9] H. Bordihn, J. Dassow, and M. Holzer. Extending regular expressions with homomorphic replacement. *RAIRO Theoretical Informatics and Applications*, 44:229–255, 2010.

[10] J. Bremer and D. D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. In *Proc. 14th International Conference on Developments in Language Theory, DLT 2010*, volume 6224 of *Lecture Notes in Computer Science*, pages 100–111, 2010.

[11] C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.

[12] C. Câmpeanu and S. Yu. Pattern expressions and pattern automata. *Information Processing Letters*, 92:267–274, 2004.

[13] J. H. Chang, O. H. Ibarra, M. A. Palis, and B. Ravikumar. On pebble automata. *Theoretical Computer Science*, 44:111–121, 1986.

[14] R. Clifford, A. W. Harrow, A. Popa, and B. Sach. Generalised matching. In *Proc. 16th International Symposium on String Processing and Information Retrieval, SPIRE 2009*, volume 5721 of *Lecture Notes in Computer Science*, pages 295–301, 2009.

[15] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling.* Addison-Wesley Publishing Company, Reading, Mass., 1967.

[16] A. Ehrenfeucht and G. Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9:86–88, 1979.

[17] P. C. Fischer and C. M. R. Kintala. Real-time computations with restricted nondeterminism. *Mathematical Systems Theory*, 12:219–231, 1979.

[18] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39:207–229, 1992.

[19] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[20] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. 8th National Conference on Artificial Intelligence, AAAI 1990*, pages 4–9, 1990.

[21] D.D. Freydenberger and D. Reidenbach. Bad news on decision problems for patterns. *Information and Computation*, 208:83–96, 2010.

[22] D.D. Freydenberger, D. Reidenbach, and J.C. Schneider. Unambiguous morphic images of strings. *International Journal of Foundations of Computer Science*, 17:601–628, 2006.

[23] J. E. F. Friedl. *Mastering Regular Expressions.* O'Reilly, Sebastopol, CA, third edition, 2006.

[24] P. Frisco and O. H. Ibarra. On stateless multihead finite automata and multihead pushdown automata. In *Proc. Developments in Language Theory 2009*, volume 5583 of *Lecture Notes in Computer Science*, pages 240–251, 2009.

[25] V. Geffert, C. Mereghetti, and G. Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205:1173–1187, 2007.

[26] M. Geilke and S. Zilles. Learning relational patterns. In *Proc. 22nd International Conference on Algorithmic Learning Theory, ALT 2011*, volume 6925 of *Lecture Notes in Computer Science*, pages 84–98, 2011.

[27] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[28] T. Harju and J. Karhumäki. Morphisms. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 7, pages 439–510. Springer, 1997.

[29] M. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1978.

[30] J. Hartmanis. On non-determinancy in simple computing devices. *Acta Informatica*, 1:336–344, 1972.

[31] M. Holzer, M. Kutrib, and A. Malcher. Complexity of multi-head finite automata: Origins and directions. *Theoretical Computer Science*, 412:83–96, 2011.

[32] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2000.

[33] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[34] L. P. Horwitz, R. M. Karp, R. E. Miller, and S. Winograd. Index register allocation. *Journal of the ACM*, 13:43–61, 1966.

[35] O. Ibarra, T.-C. Pong, and S. Sohn. A note on parsing pattern languages. *Pattern Recognition Letters*, 16:179–182, 1995.

[36] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.

[37] O. H. Ibarra and Ö. Eğecioğlu. Hierarchies and characterizations of stateless multicounter machines. In *Computing and Combinatorics*, volume 5609 of *Lecture Notes in Computer Science*, pages 408–417, 2009.

[38] O. H. Ibarra, J. Karhumäki, and A. Okhotin. On stateless multihead automata: Hierarchies and the emptiness problem. *Theoretical Computer Science*, 411:581–593, 2010.

[39] O. H. Ibarra and B. Ravikumar. On partially blind multihead finite automata. *Theoretical Computer Science*, 356:190–199, 2006.

[40] S. Jain, Y. S. Ong, and F. Stephan. Regular patterns, regular languages and context-free languages. *Information Processing Letters*, 110:1114–1119, 2010.

[41] T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *International Journal of Computer Mathematics*, 50:147–163, 1994.

[42] T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Decision problems for patterns. *Journal of Computer and System Sciences*, 50:53–63, 1995.

[43] L. Kari, G. Rozenberg, and A. Salomaa. L systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 5, pages 253–328. Springer, 1997.

[44] C. M. R. Kintala. Refining nondeterminism in context-free languages. *Mathematical Systems Theory*, 12:1–8, 1978.

[45] S.C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, volume 34 of *Annals of Mathematics Studies*, pages 3–41. Princeton University Press, 1956.

[46] T. Koshiba. Typed pattern languages and their learnability. In *Proc. 2nd European Conference on Computational Learning Theory, EUROCOLT 1995*, volume 904 of *Lecture Notes in Computer Science*, pages 367–379, 1995.

[47] M. Kutrib, H. Messerschmidt, and F. Otto. On stateless two-pushdown automata and restarting automata. *International Journal of Foundations of Computer Science*, 21:781–798, 2010.

[48] S. Lange and R. Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.

[49] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336, 1978.

[50] A. Mateescu and A. Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique théoretique et Applications*, 28:233–253, 1994.

[51] A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 230–242. Springer, 1997.

[52] M. Minsky. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74:437–455, 1961.

[53] B. Monien. Two-way multihead automata over a one-letter alphabet. *RAIRO Informatique theoretique*, 14:67–82, 1980.

[54] A. Muscholl, M. Samuelides, and L. Segoufin. Complementing deterministic tree-walking automata. *Information Processing Letters*, 99:33–39, 2006.

[55] Y.K. Ng and T. Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397:150–165, 2008.

[56] E. Ohlebusch and E. Ukkonen. On the equivalence problem for E-pattern languages. *Theoretical Computer Science*, 186:231–248, 1997.

[57] Oxford Dictionaries Online. `http://oxforddictionaries.com/definition/pattern?q=pattern` (08/06/2012).

[58] H.C. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1995.

[59] G. Della Penna, B. Intrigila, E. Tronci, and M. Venturini Zilli. Synchronized regular expressions. *Acta Informatica*, 39:31–70, 2003.

[60] F. M. Q. Pereira. A survey on register allocation. 2008. `http://compilers.cs.ucla.edu/fernando/publications/drafts/survey.pdf` (08/06/2012).

[61] H. Petersen. Automata with sensing heads. In *Proc. Theory of Computing and Systems*, pages 150 – 157, 1995.

[62] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3, 1959.

[63] D. Reidenbach. *The Ambiguity of Morphisms in Free Monoids and its Impact on Algorithmic Properties of Pattern Languages*. PhD thesis, Fachbereich Informatik, Technische Universität Kaiserslautern, 2006. Logos Verlag, Berlin.

[64] D. Reidenbach. A non-learnable class of E-pattern languages. *Theoretical Computer Science*, 350:91–102, 2006.

[65] D. Reidenbach. An examination of Ohlebusch and Ukkonen's conjecture on the equivalence problem for E-pattern languages. *Journal of Automata, Languages and Combinatorics*, 12:407–426, 2007.

[66] D. Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.

[67] D. Reidenbach and M. L. Schmid. Finding shuffle words that represent optimal scheduling of shared memory access. *International Journal of Computer Mathematics*. To appear.

[68] D. Reidenbach and M. L. Schmid. Finding shuffle words that represent optimal scheduling of shared memory access. In *Proc. 5th International Conference on Language and Automata Theory and Applications, LATA 2011*, volume 6638 of *Lecture Notes in Computer Science*, pages 465–476, 2011.

[69] D. Reidenbach and M. L. Schmid. A polynomial time match test for large classes of extended regular expressions. In *Proc. 15th International Conference on Implementation and Application of Automata, CIAA 2010*, volume 6482 of *Lecture Notes in Computer Science*, pages 241–250, 2011.

[70] D. Reidenbach and M. L. Schmid. Automata with modulo counters and nondeterministic counter bounds. In *Proc. 17th International Conference on Implementation and Application of Automata, CIAA 2012*, volume 7381 of *Lecture Notes in Computer Science*, pages 361–368, 2012.

[71] D. Reidenbach and M. L. Schmid. On multi-head automata with restricted nondeterminism. *Information Processing Letters*, 112:572–577, 2012.

[72] D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. In *Proc. 6th International Conference on Language and Automata Theory and Applications, LATA 2012*, volume 7183 of *Lecture Notes in Computer Science*, pages 468–479, 2012.

[73] D. Reidenbach and M. L. Schmid. Regular and context-free pattern languages over small alphabets. In *Proc. 16th International Conference on Developments in Language Theory, DLT 2012*, volume 7410 of *Lecture Notes in Computer Science*, pages 130–141, 2012.

[74] A. L. Rosenberg. On multi-head finite automata. *IBM Journal of Research and Development*, 10, 1966.

[75] P. Rossmanith and T. Zeugmann. Stochastic finite learning of the pattern languages. *Machine Learning*, 44:67–91, 2001.

[76] A. Salomaa. *Formal Languages.* Academic Press, New York, London, 1973.

[77] M. L. Schmid. Inside the class of regex languages. In *Proc. 16th International Conference on Developments in Language Theory, DLT 2012*, volume 7410 of *Lecture Notes in Computer Science*, pages 73–84, 2012.

[78] J. C. Schneider. Entscheidungsprobleme für Patternsprachen und kombinatorische eigenschaften von Pattern. Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern, 2006. In German.

[79] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *Proc. RIMS Symposium on Software Science and Engineering*, volume 147 of *Lecture Notes in Computer Science*, pages 115–127, 1982.

[80] T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science*, pages 191–209, 1982.

[81] M. Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10:335–338, 1980.

[82] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, Boston, 1997.

[83] F. Stephan, R. Yoshinaka, and T. Zeugmann. On the parameterised complexity of learning patterns. In *Proc. 26th International Symposium on Computer and Information Sciences, ISCIS 2011*, pages 277–281.

[84] I.H. Sudborough. On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10:62–76, 1975.

[85] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7, pages 389–455. Springer, 1997.

[86] K. Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11, 1968.

[87] K. Wright. Inductive identification of pattern languages with restricted substitutions. In *Proc. 3rd Annual Workshop on Computational Learning Theory, COLT 1990*, pages 111–121. Morgan Kaufmann, 1990.

[88] L. Yang, Z. Dang, and O. H. Ibarra. On stateless automata and p systems. *International Journal of Foundations of Computer Science*, 19:1259–1276, 2008.

[89] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 2, pages 41–110. Springer, 1997.