Local Patterns

Joel D. Day[*1], Pamela Fleischmann[†2], Florin Manea[‡3], and Dirk Nowotka[§4]

1 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
jda@informatik.uni-kiel.de

2 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
fpa@informatik.uni-kiel.de

3 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
flm@informatik.uni-kiel.de

4 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
dn@informatik.uni-kiel.de

Abstract

A pattern is a word consisting of constants from an alphabet Σ of terminal symbols and variables from a set X. Given a pattern α, the decision-problem whether a given word w may be obtained by substituting the variables in α for words over Σ is called the matching problem. While this problem is, in general, NP-complete, several classes of patterns for which it can be efficiently solved are already known. We present two new classes of patterns, called k-local, and strongly-nested, and show that the respective matching problems, as well as membership can be solved efficiently for any fixed k.

1998 ACM Subject Classification F.4.3 Formal Languages, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Patterns with Variables, Local Patterns, Combinatorial Pattern Matching, Descriptive Patterns

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.24

1 Introduction

A *pattern* is a string (word) that consists of *terminal symbols* (e. g., $\mathtt{a}, \mathtt{b}, \mathtt{c}$), treated as constants, and *variables* (e. g., x_1, x_2, x_3). A pattern is mapped to a word by substituting the variables by strings of terminals. For example, $x_1x_1\mathtt{bab}x_2x_2$ can be mapped to $\mathtt{acacbabcc}$ or $\mathtt{ccbabaa}$ by the substitution $(x_1 \to \mathtt{ac}, x_2 \to \mathtt{c})$ and $(x_1 \to \mathtt{c}, x_2 \to \mathtt{a})$, respectively. If a pattern α can be mapped to a string of terminals w, we say that α matches w.

Patterns with variables appear in various areas of theoretical computer science, such as combinatorics on words (word equations [13, 20], unavoidable patterns [19]), pattern matching (generalized function matching [1, 23]), language theory (pattern languages [2]), learning

[*] The work of Joel Day was supported by the BMBF HPSV grant 01IH15006A.
[†] The work of Pamela Fleischmann was supported by the BMBF HPSV grant 01IH15006A.
[‡] The work of Florin Manea was supported by the DFG grant MA 5725/1-2.
[§] The work of Dirk Nowotka was supported by the DFG grant 872/3-2.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 24; pp. 24:1–24:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

theory (inductive inference [2, 22, 24, 5], PAC-learning [14]), database theory (extended conjunctive regular path queries [3]), as well as in practice, e.g., extended regular expressions with backreferences [4, 12, 10], used in programming languages like Perl, Java, Python, etc. In most of these settings, patterns are used to express combinatorial pattern matching questions, e.g., whether a string contains certain types of regularities, encoded in the pattern. Thus, it is often necessary to solve efficiently the *matching problem*: given a pattern α and a string w, does α match w? The set of words (over a given alphabet of terminal symbols) matched by a pattern α is called the pattern language of α, denoted $L(\alpha)$. Thereby, the matching problem for inputs α and w becomes testing whether $w \in L(\alpha)$.

Unfortunately, the *matching problem* is NP-complete [2] in general. This is especially bad for some computational tasks on patterns which implicitly solve the matching problem and are therefore also intractable. For instance, in the field of algorithmic learning theory, this is the case for the task of computing *descriptive patterns* for finite sets of words [2, 7]: given a set of words S, find a pattern α that matches all words in S (i.e., $S \subseteq L(\alpha)$), and for any other pattern α' with $S \subseteq L(\alpha')$ we have that $L(\alpha') \not\subset L(\alpha)$ (in a sense, α is minimal in the set of patterns matching S, so it describes S as accurately as possible). As illustrated by Angluin in [2], descriptive patterns are useful for the inductive inference of pattern languages, which are important in the context of learning theory since they constitute a prominent example of a language class that is inferable from positive data (see, e. g., [17, 26, 28, 30, 21] and, for a survey, [29]). Moreover, descriptive patterns have also been applied in approaches of learning upper-best approximations of other types of formal languages (see [15, 11]). This, combined the other mentioned applications of pattern matching, provides good reason to identify cases in which this problem becomes tractable, and, moreover, to optimize as far as possible the corresponding algorithms.

In order to facilitate this, one looks at restricted classes of patterns. A thorough analysis [25, 28, 8, 9, 6, 27] of the complexity of the matching problem has provided some subclasses of patterns for which the matching problem is in P, when some (sometimes sophisticated) structural parameters of patterns are bounded by constants.

Our Contribution. In this paper we continue this line of work, and propose a series of new classes for which both the membership to the class and the matching problem can be decided in polynomial time. The first, k-local patterns, are a natural generalization of *non-cross patterns*, introduced in [28]. In these patterns, no occurrence of a variable $x_2 \neq x_1$ is allowed between any two occurrences of the variable x_1. For instance, $x_1\mathtt{aa}x_1\mathtt{b}x_2\mathtt{a}x_2x_3x_3$ is a non-cross pattern. We can test in linear time whether a pattern is non-cross, and an efficient matching algorithm for non-cross patterns was given in [6]. The general idea behind a matching algorithm for such patterns is rather easy: for a pattern α, one can order its variables in a sequence $x_1, x_2, \ldots, x_k$ such that α can be written as $\alpha_1\alpha_2 \ldots \alpha_k$ with α_i containing only occurrences of x_i and terminals for all $i \in [k]$. To match this to a word w, we first look for all ways to replace x_1 by a factor of w such that α_1 is mapped to some prefix $w[1..i_1]$ of w. Then we look for all ways to replace x_2 by a factor of w such that α_2 is mapped to a factor $w[i_1 + 1..i_2]$ of w with $w[1..i_1]$ being a possible image of α_1, and so on. In general, we try to find all possible ways to map $\alpha_1 \cdots \alpha_j$ to a prefix $w[1..i_j]$ of w looking at how $\alpha_1 \cdots \alpha_{j-1}$ were mapped to a prefix $w[1..i_{j-1}]$ of w. In the end, we check if there is a way to map α to w. This can be clearly implemented in polynomial time using dynamic programming; see [6] for an efficient algorithm based on combinatorics on words insights.

We extend this matching idea to define the class of *k-local patterns*, which are defined in full in Section 3. It is possible to match such patterns by substituting the variables with strings in such an order that at any given point, one has matched (at most) k separate factors (blocks) of the pattern to the same number of factors of the input word. Thus, we can keep track of the k matched factors, which will be extended by assigning the next variable, in the aforementioned order. So, instead of aligning a prefix of the pattern with a prefix of the word, we align $j \leq k$ factors of the pattern to j factors of the word. For fixed k, a dynamic programming strategy will also work in this case to obtain a polynomial algorithm for matching k-local patterns, once we have the order in which the variables of the patterns are to be assigned. However, the definition of k-local patterns is much more involved than the definition of non-cross patterns and leads to a deeper understanding of the combinatorial structure of the pattern, also when compared to, for instance, patterns with a bounded number of variables or a bounded number of repeated variables. As such, it is not surprising that the problem of testing whether a pattern is k-local is a more involved computational task. Provided that k is fixed, we produce two polynomial-time algorithms. The first one decides in time $O(km^{2k})$ whether a pattern of length m is k-local. In this process we can construct the order in which the variables are to be assigned and we derive a second algorithm that, given a pattern α of length m and a word w of length $n \geq m$, decides whether α is k-local and, if so, whether α matches w in time $O(mkn^{3k-1})$.

While the definition of k-local patterns is somehow algorithmic, it does not say much about the syntactic structure of these patterns. We give precise structural characterizations of the 1- and 2-local patterns, and show that in the case of 1-local patterns this leads to a linear algorithm deciding whether a pattern is 1-local (Theorem 10) and an $O(mn^2 \log n)$-time algorithm matching a 1-local pattern of length m to a word of length n (Theorem 11).

These algorithms are not a direct implementation of the straightforward dynamic programming approach, but rather they combine this with some insights in the structure of k-local patterns and use non-trivial string processing data structures. Moreover, while k-local patterns (for fixed k) can be shown also to have bounded treewidth, and hence the techniques from [25] may be used to derive a polynomial time matching algorithm, it is worth noting that both for general k, and especially in the case $k = 1$, the algorithms presented in the current paper provide a significant improvement in complexity.

The palindromic structure of 1-local patterns (see Lemma 9) also gives rise to the idea of separating parts of a pattern with parenthesizing variables. We define the class of *strongly-nested patterns* (Definition 19, Section 5), which are a restriction on the nested patterns, and more generally, the mildly entwined patterns introduced in [25]. This further restriction comes with advantages: we show that one can decide whether a pattern is nested in linear time, and that one can match a pattern of length m to a word of length n in $O(mn^3)$ time, also a significant improvement on the $O(mn^6)$ algorithm known for mildly entwined patterns.

Additionally, we show that there is no constant k such that all strongly-nested patterns are k-local. In particular, we construct a simple algorithm based on combinatorial insights which computes, for a nested pattern α, the minimum ℓ such that α is ℓ-local. It is straightforward to infer from the algorithm that there are nested patterns of length m that are $\Theta(\log m)$-local.

Shinohara Classes. In addition to considering restricted classes of patterns, one can restrict the concept of descriptiveness to any given class Π of patterns: a pattern α is Π-descriptive if $\alpha \in \Pi$, $S \subseteq L(\alpha)$ and there is no other pattern $\beta \in \Pi$ with $S \subseteq L(\beta) \subset L(\alpha)$. Based on this idea, in [28], Shinohara initiated a line of research by providing a polynomial-time algorithm, for a (very) restricted class of patterns and a finite set of strings. This approach was extended in [7] to the notion of a *Shinohara-class* of patterns: a class of patterns Π is a *Shinohara-class* if it contains the set $\{x_1 x_2 \cdots x_k \mid k \in \mathbb{N}\}$ and, for every $\alpha \in \Pi$, the pattern

α' obtained by substituting some length i suffix of α by a sequence of new variables $y_1y_2\cdots y_i$ is also in Π. Within the set of Shinohara-classes of patterns, it was shown that Π-descriptive patterns can be computed in polynomial time if and only if the question whether $\alpha \in \Pi$ and the matching problem for Π are polynomial-time decidable. Hence, it is interesting to identify Shinohara classes for which the matching problem can be solved efficiently. It is easily seen that both k-local patterns and strongly-nested patterns are Shinohara classes (provided $k \geq 1$). Thus, we can conclude that, provided k is treated as constant, descriptive patterns can be computed for both these classes in polynomial time.

2 Basic Definitions

For detailed definitions regarding combinatorics on words we refer to [18], [19]. For $n, i, j \in \mathbb{N}_0$ with $i \leq j$, let $[n] = \{1, \ldots, n\}$ and $[i, j] = \{i, i+1, \ldots, j-1, j\}$. In this paper, $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ denotes a finite alphabet of *terminal symbols* and $X = \{x_1, x_2, \ldots\}$ a potentially infinite alphabet of *variables*. We assume $\Sigma \cap X = \emptyset$. Words in $(X \cup \Sigma)^*$ are *patterns*, while words in Σ^* are *terminal words* (usually just words). Patterns in X^* are called *terminal-free*. We denote the set of patterns by $\textsc{Pat}_\Sigma = (X \cup \Sigma)^*$ and define $\textsc{Pat} = \bigcup_\Sigma \textsc{Pat}_\Sigma$. The *empty word* is denoted by ε and the *length* of a word w by $|w|$. Given a pattern α, let $\mathrm{alph}(\alpha)$ and $\mathrm{var}(\alpha)$ be respectively the smallest sets $\Delta \subseteq \Sigma$ and $Y \subseteq X$ such that $\alpha \in (\Delta \cup Y)^*$. Given a word $w = \mathtt{a}_1\mathtt{a}_2 \ldots \mathtt{a}_n$, the reversal w^R of w is $\mathtt{a}_n\mathtt{a}_{n-1} \ldots \mathtt{a}_1$. For $w \in \Sigma^*$ and each $i, j \in [|w|]$ with $i \leq j$, let $w[i..j] = w[i] \cdots w[j]$, where $w[k]$ represents the k^{th} letter of w for $k \in [|w|]$. Each word $w[i..j]$ is a *factor* of w. If $0 < |w[i..j]| < |w|$ then $w[i..j]$ is a *proper factor* of w. We use the term 'blocks' for factors which satisfy a property but which are not contained strictly within a larger factor satisfying the same property. Given a variable x and pattern α, a *block* of x is a factor $\beta = \alpha[i..j]$ with $\mathrm{var}(\beta) = \{x\}$ such that either $i = 1$ or $\alpha[i-1] = y_1 \neq x$ and either $j = |\alpha|$ or $\alpha[j+1] = y_2 \neq x$. *Marked blocks* are defined similarly in Section 3. Given a variable x, we denote a block of x by $[x]^{\mathtt{b}}$ (resp. $x^{\mathtt{b}}$ if it necessarily contains at least one x). Using this notation we can define classes of patterns, e.g., $\alpha \in [x]^{\mathtt{b}} y z^{\mathtt{b}}$ implies that α has the form given by the regular expression $(\Sigma \cup \{x\})^* y (\Sigma \cup \{z\})^+$.

A *substitution* (*for* α) is a mapping $h : \mathrm{var}(\alpha) \to \Sigma^+$. For every $x \in \mathrm{var}(\alpha)$, we say that x *is substituted by* $h(x)$. The word obtained by substituting every occurrence of a variable x in α by $h(x)$ and leaving the terminals unchanged is denoted by $h(\alpha)$. For instance, we consider the pattern $\beta = x_1\mathtt{a}x_2\mathtt{b}x_2$ and the words $u = \mathtt{bacabca}$, $v = \mathtt{aaaabaa}$. It can be verified that $h(\beta) = u$, where $h(x_1) = \mathtt{b}$, $h(x_2) = \mathtt{ca}$ and $g(\beta) = v$, where $g(x_1) = \mathtt{a}$ and $g(x_2) = \mathtt{aa}$. Given a pattern α, the set $\{h(\alpha) \mid h$ is a substitution$\}$ is the *pattern language* of α, denoted $L(\alpha)$. The *matching problem*, denoted by $\textsc{Match}$, is to decide for a given pattern α and word w, whether there exists a substitution h with $h(\alpha) = w$.[1] For any $P \subseteq \textsc{Pat}$, the *matching problem for* P is to decide for a given pattern $\alpha \in P$ and word w, whether there exists a substitution h with $h(\alpha) = w$.

A pattern α is *regular* if each variable $x \in X$ occurs at most once. Given a pattern α and $y \in \mathrm{var}(\alpha)$, the *scope of* y *in* α is defined by $\mathrm{sc}_\alpha(y) = [i, j]$, where i is the leftmost and j the rightmost occurrence of y in α. The scopes of some variables $y_1, y_2, \ldots, y_k \in \mathrm{var}(\alpha)$ *coincide in* α if $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. We denote the *scope coincidence degree* (scd for short) of α by $\mathrm{scd}(\alpha)$, which is the maximum number of variables in α such that their scopes coincide. For example, the scopes of all variables coincide in $\alpha_1 = x_1x_2x_1x_2x_3x_1x_2x_3$, but the scopes of

[1] There exist variants of the matching problem where substitutions can also *erase* variables by mapping them to ε. Here we only consider non-erasing substitutions.

x_1 and x_3 do not coincide in $\alpha_2 = x_1x_2x_1x_2x_3x_2x_3x_3$; thus, $\mathrm{scd}(\alpha_1) = 3$ and $\mathrm{scd}(\alpha_2) = 2$. For every $k \in \mathbb{N}$, let $\textsc{Pat}_{\mathrm{scd}\leq k}$ denote the set of patterns α with $\mathrm{scd}(\alpha) \leq k$. The class of *non-cross* patterns (see [28])) coincides exactly with $\textsc{Pat}_{\mathrm{scd}\leq 1}$. A set $\Pi \subseteq (X \cup \Sigma)^*$ is a *Shinohara class* if $\{x_1x_2\cdots x_k \mid k \in \mathbb{N}\} \subseteq \Pi$ and, for every $\alpha \in \Pi$ and $i \in [|\alpha|]$, we have $\alpha' = \alpha[1..i]y_1y_2\ldots y_{|\alpha|-i} \in \Pi$, where $y_1, y_2, \ldots, y_{|\alpha|-i} \in X\backslash \mathrm{var}(\alpha)$ with $y_j \neq y_k$ for $1 \leq j < k \leq |\alpha|-i$. For a class Π of patterns and a finite set of words $S \subset \Sigma^*$, a pattern $\alpha \in \Pi$ is $\Pi-$*descriptive* of S if there does not exist a pattern $\beta \in \Pi$ such that $S \subseteq L(\beta) \subset L(\alpha)$.

3 k-Local Patterns

In the following section, the main ideas surrounding k-locality are presented along with the formal definition and some initial observations. At the end of the section, two of the main results motivating the idea of k-locality are presented: namely that if k is a fixed constant, then the membership and matching problems may be solved efficiently for k-local patterns.

Intuitively, the notion of k-locality involves marking the variables in the pattern in some arbitrary order until all the variables are marked. The pattern is k-local if this marking can be done while never creating more than k marked blocks. Variables which only occur adjacent to those which are already marked can be marked "for free" – without creating any new blocks, and thus a valid marking sequence allows a sort-of parsing of the pattern whilst maintaining a degree of closeness (locality) to the parts already parsed.

As with various other classes of patterns motivated by efficient matching algorithms (bounded scd, non-cross, etc.), k-locality deals only with the relative positions of variables, while the terminal symbols are not taken into account. Hence, before we introduce k-locality formally, it is convenient to consider the underlying pattern consisting only of variables – the *skeleton*. For example, the skeleton of $\beta = \mathtt{aa}xx\mathtt{c}y\mathtt{ba}z\mathtt{a}y\mathtt{a}y$ would be $\alpha = xxyzyy$.

▶ **Definition 1.** Let $\beta \in (X \cup \Sigma)^*$. The *skeleton* of β is the (unique) pattern $\alpha = y_1 \ldots y_n$ with $y_i \in X$ for $i \in [n], n \in \mathbb{N}$, such that there exist words $a_0, \ldots, a_n \in \Sigma^*$ with $\beta = a_0y_1a_1 \ldots a_{n-1}y_na_n$.

Next, the idea of marking a variable is formalized. For each variable $x \in X$, we produce a marked version $\overline{x}$. Marking x corresponds to substituting *every* occurrence of x in a pattern with its marked equivalent $\overline{x}$.

▶ **Definition 2.** Let $\overline{X} = \{\overline{x} \mid x \in X\}$ be the set of *marked variables* (with $\overline{X} \cap X = \emptyset$). For the skeleton α of a pattern $\beta \in (X \cup \Sigma)^*$, a *marking sequence* of the variables occurring in β, is an enumeration $x_1, x_2, \ldots, x_{|\operatorname{var}(\beta)|}$ of $\mathrm{var}(\beta)$. A variable x_i is called *marked at point* $k \in \mathbb{N}$ (both in β and α) if $i \leq k$. Moreover, we define α_k, *the marked skeleton of* β *at point* k, as the string obtained from α by replacing all x_i with $i \leq k$ by $\overline{x_i}$. A factor of α_k is a *marked block* if it consists of one or more marked variables and is maximal in the sense that it is not contained within another such factor.

Using the idea of a marking sequence, we can now define the k-locality of a pattern.

▶ **Definition 3.** A pattern $\beta \in (X \cup \Sigma)^*$, with skeleton α, is k*-local* for $k \in \mathbb{N}_0$ if there exists a marking sequence $x_1, \ldots, x_\ell$ of $\mathrm{var}(\beta)$, such that, for all $i \leq \ell$ we have that α_i, the marked skeleton of β at point i, has at most k marked blocks. A pattern is called *strictly* k-local if it is k-local but not $(k-1)$-local. Let $\textsc{Pat}_{k\text{-loc}}$ denote the class of k-local patterns.

Consider the pattern $\beta = \mathtt{a}x\mathtt{a}yx\mathtt{b}y\mathtt{a}z\mathtt{ab}xz$, whose skeleton is $\alpha = xyxyzxz$. If we consider the marking sequence y, x, z, then we obtain the following (partially) marked patterns:

$$\alpha_1 = x\ \overline{y}\ x\ \overline{y}\ z\ x\ z, \quad \alpha_2 = \overline{x\ y\ x\ y}\ z\ \overline{x}\ z, \quad \alpha_3 = \overline{x\ y\ x\ y\ z\ x\ z}.$$

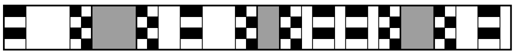

Figure 1 Lemma 5: if i blocks are already marked (grey areas), the next variable in the marking sequence may appear at $2i$ positions next to the marked blocks (chessboard), and at $k-2i$ positions not connected to marked blocks (striped).

Note that the final pattern (in this case α_3) will always be completely marked, and hence has exactly one marked block. However, since α_1 and α_2 both have exactly two marked blocks, β (and α) are 2-local. On the other hand, they are not 1-local since every alternative marking results in at least two blocks: the other possibilities for α_1 are $\overline{x}\,y\,\overline{x}\,y\,z\,\overline{x}\,z$ and $x\,y\,x\,y\,\overline{z}\,x\,\overline{z}$. The particular cases of 1- and 2-local patterns are considered in more detail in Section 4 where structural characterizations are given.

Before moving on to the first main results regarding the membership and matching problems, a few basic observations for k-local patterns are presented. Firstly the relationship between a k-local pattern and its factors is considered. Lemma 5 is illustrated by Figure 1.

▶ **Lemma 4.** *Let $\beta \in (X \cup \Sigma)^*$ be k-local. Then every factor of β is k-local. Moreover, if β is strictly k-local and $k \geq 2$, then there exists a proper factor of β which is not $(k-2)$-local.*

▶ **Lemma 5.** *Let $\beta \in (X \cup \Sigma)^*$ be a strictly k-local pattern with skeleton α. For all $x \in \mathrm{var}(\beta)$, the number of blocks $x^{\mathfrak{b}}$ occurring in α is at most $2k$. Moreover, there exists a variable $y \in \mathrm{var}(\beta)$ such that the number of blocks $y^{\mathfrak{b}}$ occurring in α is at most k.*

With regards to existing classes of patterns for which the matching problem may be efficiently solved, k-local patterns generalize the non-cross patterns in a significant manner. In fact, it follows from the results in Section 4 that non-cross (and therefore regular) patterns are 1-local. On the other hand, it can be seen with a little effort that for fixed k, k-local patterns also have treewidth bounded by $2k$. Thus are a more restricted subclass of the patterns considered in [25]. Furthermore, k-locality is incomparable to the notion of bounded scope coincidence degree, as witnessed by the following examples. The pattern $(xy)^{k+1}$ has scope coincidence degree one, while it is strictly $(k+1)$-local: marking either x or y first will result in exactly $k+1$ marked blocks. On the other hand, the pattern $x_1x_2\dots x_{n-1}x_nx_{n-1}\dots x_2x_1$ is 1-local (simply mark the x_is in decreasing order), but has scope coincidence degree n provided $x_i \neq x_j$ for all $i,j \in [n]$. Note also the special degenerate case of 0-local patterns.

▶ Remark 6. Let $\beta \in (X \cup \Sigma)^*$ be a pattern. Then β is 0-local if and only if $\beta \in \Sigma^*$.

Finally, the main results of this section are given. Theorems 7 and 8 show that, if k is fixed, it is possible to decide whether a pattern is k-local and also to match a k-local pattern to a word in polynomial time. The degree of the polynomial, however, depends on k. In particular, Theorem 8 provides an improvement when compared to patterns with bounded treewidth in general.

▶ **Theorem 7.** *Given a pattern $\beta \in (X \cup \Sigma)^*$ of length m, we can decide in $O(m^{2k}k)$ time whether $\beta \in \mathrm{PAT}_{\text{k-loc}}$. If the answer is positive, we can produce in the same time a marking sequence witnessing that β is k-local.*

The algorithm deciding whether a pattern is k-local keeps track of all possibilities having $j \leq k$ marked blocks in the skeleton of β, after exactly i variables were marked. Then, for each possibility, a new (the $(i+1)^{th}$) variable is selected, its (at most $2k$, see Lemma 5) blocks are marked, and they are merged with the already marked blocks. If the resulting skeleton has at most k marked blocks, it is saved and will be processed later, when the next variable has to be marked. The pattern is k-local if and only if its entire skeleton can be marked in this way. To get the stated running time, one has to be careful when a new variable is selected: if we already have k marked blocks in our skeleton (and there are $O(m^{2k})$ possibilities), then the new variable has to be adjacent to one of them. If there are less than k marked blocks, then we are not so restricted on how to choose it; however, in this case, the number of skeletons with $j < k$ marked blocks that we consider is lower, only $O(m^{2k-2})$.

We are also able to take advantage of the k-locality structure to solve the matching problem more efficiently than the more general (but less direct) approach given in [25].

▶ **Theorem 8.** *Match for $\mathrm{Pat}_{\text{k-loc}}$ can be decided in $O(mkn^{\max(3k-1,2k+1)})$ time, where m is the length of the input pattern and n is the length of the input word.*

To solve the matching problem for $\mathrm{Pat}_{k\text{-loc}}$ we use essentially the same idea as above. We now have the order in which the variables have to be assigned, so also the marked factors in the pattern, but we need to keep track to which factors of the input word they correspond. Then we try to assign every new variable so that it fits nicely around the already matched factors. This is done efficiently using a data structure from [16]: given a word w and a one-variable pattern γ (so, $|\operatorname{var}(\gamma)| = 1$), one can produce a compact representation of all the g factors of w matching γ in $O(|\gamma||w|)$ time; moreover, we can obtain all the g factors of w matching γ in $O(|g|)$ time. This allows us to test efficiently which factors of w match any of the one-variable blocks of β, and, ultimately, to assign a value to each variable.

In comparison to the algorithm from [25] for patterns of bounded treewidth, which firstly constructs relational structures (which may be thought of as graphs) from α and w, and solves the homomorphism problem on these relational structures, the above algorithm exploits directly the locality structure present in the patterns. The advantage of this more focussed approach is that it allows for a considerable improvement in the required time, reducing the exponent of n from $4k + 4$ to $3k - 1$.

In [8, 9] it was shown that for many choices of numerical parameters, the matching problem is $W[1]$-hard. In particular, the problem is $W[1]$-hard when parameterized by the number of variables and the maximum number of occurrences of a variable. Since these values bound the total number of variables (i.e., the length of the skeleton), they also bound the strict k-locality of a pattern. Consequently, the matching problem is also $W[1]$-hard when parameterized by strict k-locality. In addition to this observation, we add the following two conjectures: deciding whether a pattern is k-local, when given k as input together with the pattern, is NP-complete; and thus that computing the minimum k such that a pattern is k-local is a computationally hard problem as well. Finally, we conjecture that the problem of deciding whether a pattern is k-local, parametrized in k, is also $W[1]$-hard.

4 1-Local and 2-Local Patterns

For 0-local patterns, a straightforward structural characterization exists: they are precisely the patterns without variables. The general case, however, is considerably more complex. Nevertheless, for small k (1 and 2), it is possible to give some recursive, structural characterizations. Firstly, given a 1-local pattern α, and a variable $x \notin \operatorname{var}(\alpha)$, we can add occurrences of x to the start of α, the end, or both, while preserving 1-locality. Moreover, this operation is sufficient to characterize 1-local patterns recursively. Since k-locality depends only on the skeleton of a pattern, in the following lemma, only terminal-free patterns are considered.

▶ **Lemma 9.** *A terminal-free pattern $\alpha \in X^*$ is 1-local if and only if α is empty or there exist a shorter, 1-local pattern α' and a variable $x \notin \operatorname{var}(\alpha')$ such that $\alpha \in [x]^{\mathfrak{b}}\alpha'[x]^{\mathfrak{b}}$ holds.*

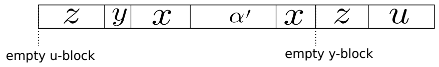

Figure 2 Extending a 1-local structure at both sides simultaneously by a possibly empty block of the same variable preserves the 1-locality.

It follows from this characterization that the structure of 1-local patterns has at its core a sort-of palindromic structure. In particular, if each new variable x is added to both sides of α, then, ignoring the number of repetitions, we get a palindrome in the variables. More generally, if a new variable x is added to only one side of α, then this may be viewed as adding a zero-repetition of x to the other, and hence it is still possible to infer an underlying palindromic structure (Figure 2). This structure, along with Lemma 9 is useful because it leads to more efficient membership and matching algorithms for 1-local patterns.

▶ **Theorem 10.** *Given a pattern $\beta \in (X \cup \Sigma)^*$ of length m, we can decide in $O(m)$ time whether $\beta \in \textsc{Pat}_{1\text{-loc}}$. If the answer is positive, we can produce in the same time a marking sequence witnessing that β is 1-local.*

While the theorem above follows immediately, as only a stack is needed to check that the variables occurring in more than one block in the skeleton of β form this palindromic structure, solving the matching problem efficiently requires a much finer combinatorial analysis of the way we can assign values to the variables of the pattern β. Intuitively, matching a 1-local pattern β, of length m, to a word w, of length n, is done as follows. We first get the marking sequence witnessing the 1-locality of β and then start assigning values to its variables, in the order indicated by this sequence. Doing this, after trying to assign the first s variables, we identify all possibilities to match the factor β' of β, which contains all the occurrences of the s assigned variables, to factors $w[i..j]$ of w. Now, if the next variable to be assigned is x, then the blocks of this variable x must match factors adjacent to $w[i..j]$. Our algorithm has now two phases. First, for each pair i, j such that $w[i..j]$ was a match for β', we identify in $O(\log n)$ time some basic ways to assign the variable x, as described above. This gives us some new factors $w[i'..j']$ that match β'', the factor of β that includes β' and all occurrences of x. In overall $O(n^2 \log n)$ time, we can extend these factors using their combinatorial properties (for instance, periodicity) to find all pairs i'', j'' such that $w[i''..j'']$ matches β''. We extend, thus, the matchings constructed by assigning one variable at a time, in the order of the marking sequence, just as we did in the algorithm of Theorem 8, but this time we do it more efficiently using combinatorics on words insights.

▶ **Theorem 11.** $\textsc{Match}$ *for* $\textsc{Pat}_{1\text{-loc}}$ *can be decided in $O(mn^2 \log n)$ time, where m is the length of the input pattern and n is the length of the input word.*

The “palindromic” structure present in 1-local patterns, while simple, is also an integral part of understanding patterns with larger values k, since it describes precisely when additional variables may be marked without creating new marked blocks and hence the idea of locality. The characterization given in the lemma rests on the fact that, when marking a 1-local pattern, at any point we have a single marked block. Since two marked blocks are not allowed, it is then only possible to mark variables which only occur directly to either side of the current block. In the more general case, we may have a number of blocks which are not 1-local, but rather k-local for some values k. Then, using the same idea, it is possible to continue to mark variables only occurring adjacent to each of the blocks in order, and we get the same palindromic structure. Formally, this generalization is as follows.

▶ **Definition 12.** Patterns $\gamma_1, \gamma_2, \ldots, \gamma_n \in (X \cup \Sigma)^*$ are *homogeneously ordered* if for every $x, y \in X$, $x \neq y$ such that $\gamma_i = \delta_1 x \delta_2 y \delta_3$ for some $i \in [n]$ and $\delta_1, \delta_2, \delta_3 \in (X \cup \Sigma)^*$, there does not exist $j \in [n]$ such that $\gamma_j = \delta'_1 y \delta'_2 x \delta'_3$ where $\delta'_1, \delta'_2, \delta'_3 \in (X \cup \Sigma)^*$. A pair of patterns (γ_1, γ_2) is called *mutually-palindromic* if γ_1 and γ_2^R are homogeneously ordered.

Essentially, patterns are homogeneously ordered if they are non-cross, and the variables appear in the same order from left to right in every pattern (though some variables may not appear in every pattern). Pairs in which one pattern is reversed provide the necessary outward palindromic structure. For example, the patterns $xyzw, xw, yw$ and w are homogeneously ordered, while the patterns $xxyzz$ and $zxyy$ are not. Similarly, the pair $(xxyzzz, zzxx)$ is mutually-palindromic, since $xxzz$ and $xxyzzz$ are homogeneously ordered.

▶ Remark 13. Lemma 9 implies that for every non-empty 1-local pattern $\alpha \in (X \cup \Sigma)^*$ there exists a (not necessarily unique) mutually-palindromic pair (γ_1, γ_2) such that $\alpha = \gamma_1 \gamma_2$. Moreover, concatenating a mutually-palindromic pair always gives a 1-local pattern.

A more general version of this observation demonstrates how mutually-palindromic patterns allow for blocks (or patterns) to be extended without increasing the k-locality.

▶ **Lemma 14.** *Let $\alpha, \beta_1, \beta_2 \in (X \cup \Sigma)^*$ such that (β_1, β_2) is a mutually-palindromic pair and* $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta_1 \beta_2) = \emptyset$. *Then $\beta_1 \alpha \beta_2$ is k-local if and only if α is k-local, for all $k \in \mathbb{N}$.*

Before giving the characterization of 2-local patterns, which, as is to be expected, is more involved than for 1-local patterns, the idea of 2-local pairs is considered.

▶ **Definition 15.** Let $\alpha_1, \alpha_2 \in (X \cup \Sigma)^*$. Then (α_1, α_2) is a 2*-local pair* if there exists a marking sequence of the variables in $\mathrm{var}(\alpha_1) \cup \mathrm{var}(\alpha_2)$ such that when applied simultaneously to mark α_1 and α_2, the sum of the number of blocks in the marked skeletons at every point is at most two.

For example, $(xyxyz, x)$ is a 2-local pair, due to the marking sequence y, z, x, while $(xyxy, xy)$ is not, since whichever of x, y is marked first, there will be a total of three marked blocks (two blocks in the first pattern and one in the second). Moreover, if (α_1, α_2) is a 2-local pair, then for a pattern $\beta_1 \alpha_1 \beta_2 \alpha_2 \beta_3$ such that $\mathrm{var}(\alpha_1 \alpha_2) \cap \mathrm{var}(\beta_1 \beta_2 \beta_3) = \emptyset$, there exists a marking sequence which marks all the variables in α_1 and α_2 using at most two marked blocks. If β_2, α_1 and α_2 are non-empty, then the marking sequence requires exactly two marked blocks. A consequence is the following simple characterization of 2-local patterns.

▶ **Lemma 16.** *A pattern $\alpha \in (X \cup \Sigma)^*$ is 2-local if and only if there exist homogeneously ordered patterns $\beta_1, \beta_2, \beta_3, \beta_4 \in (X \cup \Sigma)^*$ and a 2-local pair $(\alpha', \alpha'') \in ((X \cup \Sigma)^*)^2$ such that $\alpha = \beta_1 \alpha' \beta_2^R \beta_3 \alpha'' \beta_4^R$, where* $\mathrm{var}(\beta_1 \beta_2 \beta_3 \beta_4) \cap \mathrm{var}(\alpha' \alpha'') = \emptyset$, *and $|\alpha' \alpha''| < |\alpha|$.*

Nevertheless, the characterization still says little about the structure of 2-local patterns due to the fact that the structure of 2-local pairs is not discussed. A recursive characterization, conceptually similar to Lemma 9 is given for 2-local pairs below.

▶ **Lemma 17.** *Let $\alpha, \beta \in (X \cup \Sigma)^*$ be patterns such that* $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta) \neq \emptyset$. *Then (α, β) is a 2-local pair if and only if there exists a 2-local pair $\alpha', \beta' \in (X \cup \Sigma)^*$, homogeneously ordered patterns $\gamma_1, \gamma_2, \gamma_3, \gamma_4 \in (X \cup \Sigma)^*$ and a variable $x \in X$ such that either*

- $\alpha \in \gamma_1 [x]^{\mathfrak{b}} \alpha' [x]^{\mathfrak{b}} \beta' [x]^{\mathfrak{b}} \gamma_2^R$ *and* $\beta \in \gamma_3 [x]^{\mathfrak{b}} \gamma_4^R$, *or*
- $\alpha \in \gamma_1 [x]^{\mathfrak{b}} \gamma_2^R$ *and* $\beta \in \gamma_3 [x]^{\mathfrak{b}} \alpha' [x]^{\mathfrak{b}} \beta' [x]^{\mathfrak{b}} \gamma_4^R$.

where $\mathrm{var}(\gamma_1 \gamma_2 \gamma_3 \gamma_4), \{x\},$ *and* $\mathrm{var}(\alpha' \beta')$ *are pairwise disjoint.*

▶ **Lemma 18.** *Let $\alpha, \beta \in (X \cup \Sigma)^*$ be patterns with* $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta) = \emptyset$. *Then (α, β) is a 2-local pair if and only if both α and β are 2-local, and at most one of α, β is strictly 2-local.*

Using Lemmas 16, 17 and 18, it is possible to derive a dynamic programming algorithm for recognizing 2-local patterns. However, it is worth pointing out that such an algorithm runs in the same time as the one given by Theorem 7 and hence, even in the case of 2-locality, it is reasonable to expect that any improvements would require significant effort.

5 Strongly-Nested Patterns

In the current section, another class of patterns which satisfy certain locality-inspired structural constraints are considered, namely the *strongly-nested patterns*, which form a subclass of the nested patterns, and consequently the mildly entwined patterns defined in [25]. It is shown that the membership problem for the class of strongly-nested patterns can be solved in linear time, and that the matching problem can be solved in $O(mn^3)$ time, where m is the length of the pattern and n is the length of the word to be matched: a considerable improvement on the mildly entwined patterns in general, for which the state of the art algorithm requires $O(mn^6)$ time. It is then shown that being strongly-nested is orthogonal to k-locality for fixed k, and that the optimal k – the smallest such that a pattern is k-local – can be computed efficiently for this class. The definition of strongly-nested patterns is given formally as follows.

▶ **Definition 19.** A pattern $\alpha \in (X \cup \Sigma)^*$ is *strongly-nested* if, for every variable $x \in \mathrm{var}(\alpha)$ there exist $\alpha_1, \alpha_2, \alpha_3 \in X^*$ such that $\alpha \in \alpha_1[x]^{\mathfrak{b}}\alpha_2[x]^{\mathfrak{b}}\alpha_3$, where $\{x\}$, $\mathrm{var}(\alpha_1\alpha_3)$ and $\mathrm{var}(\alpha_2)$ are pairwise disjoint. The set of all strongly-nested patterns is denoted $\textsc{Pat}_{\mathrm{nest}}$.

An alternative, inductive definition for strongly-nested patterns is the following: Patterns β with $|\mathrm{var}(\beta)| = 1$ are basic strongly-nested patterns. If β_1, β_2 are variable-disjoint strongly-nested patterns, x is a variable not contained in β_1, and $\gamma_1, \gamma_2 \in (\Sigma \cup \{x\})^*$, then both $\beta_1\beta_2$ and $\gamma_1\beta_1\gamma_2$ are strongly-nested patterns. Essentially, the construction of $\gamma_1\beta_1\gamma_2$ corresponds to wrapping blocks of x around the skeleton of β_1, which suggests the idea of nesting.

The alternative definition of strongly-nested patterns suggests also a notion of *depth* of such a pattern. To begin with, we say that the $\mathrm{depth}(\beta) = 1$ if $|\mathrm{var}(\beta)| = 1$. Further, if β_1, β_2 are variable-disjoint strongly-nested patterns, then $\mathrm{depth}(\beta_1\beta_2) = \max(\mathrm{depth}(\beta_1), \mathrm{depth}(\beta_2))$. Finally, if x is a variable not contained in β_1, and $\gamma_1, \gamma_2 \in (\Sigma \cup \{x\})^*$ with $x \in \mathrm{var}(\gamma_1) \cap \mathrm{var}(\gamma_2)$, then $\mathrm{depth}(\gamma_1\beta_1\gamma_2) = 1 + \mathrm{depth}(\beta_1)$. It is a simple consequence that $\mathrm{scd}(\beta) = \mathrm{depth}(\beta)$, if β is a strongly-nested pattern.

▶ **Theorem 20.** *For a pattern $\beta \in (X \cup \Sigma)^*$, of length m, we can decide in $O(m)$ time whether $\beta \in \textsc{Pat}_{\mathrm{nest}}$.*

As for 1-local patterns, the theorem above follows easily, as we can use a stack to check that the variables which occur in more than one block in the skeleton of β form a correct strongly-nested structure (i.e., they do not interleave like $..x..y..x..y..$ and occur in at most two blocks). In fact, we can represent the strongly-nested structure of a pattern as follows: let $\gamma \in (X \cup \Sigma)^*$ be a strongly-nested pattern that starts with a variable. We associate to γ a binary tree T_γ defined as follows:

1. If $|\mathrm{var}(\gamma)| = 1$, then T_γ has a single node, labelled with γ.
2. If $\gamma = \gamma'\gamma''$ where γ' and γ'' are variable disjoint strongly-nested patterns, both starting with a variable, then the tree associated with γ consists of a node labelled with $\odot$ which has two children. The left child is the tree $T_{\gamma'}$, and the right child is the tree $T_{\gamma''}$. If there are multiple ways to write γ as the catenation of two variable disjoint strongly-nested patterns γ' and γ'', we choose the one where γ' has minimal length.

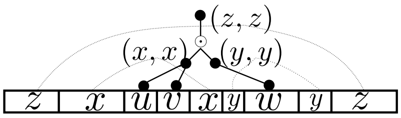

Figure 3 In the pattern, embedded in the z-blocks, two patterns are nested, which are again nested itself, namely the ones starting and ending in x and y respectively.

3. If $\gamma = \gamma'\gamma''\gamma'''$, where $\mathrm{var}(\gamma') = \mathrm{var}(\gamma''') = \{x\}$ and $x \notin \mathrm{var}(\gamma'')$ and γ', γ'', and γ''' start with a variable, then the tree T_γ consists of a node labelled with (γ', γ''') which has a single child: the tree $T_{\gamma''}$.

▶ **Lemma 21.** *The tree T_γ can be constructed in linear time $O(|\gamma|)$.*

Using the tree structure defined in the previous lemma, we can solve the matching problem for strongly-nested patterns efficiently.

▶ **Theorem 22.** Match *for* $\mathrm{PAT}_{\mathrm{nest}}$ *can be decided in $O(mn^3)$ time, where m is the length of the input pattern $\beta \in (X \cup \Sigma)^*$ and n is the length of the input word $w \in \Sigma^*$.*

In this algorithm, we assign the variables of β following, again, a local approach: we first assign all the variables in a subtree of T_β, and only then move on to its sibling (if it has one). We start with the trees consisting of single nodes, and then move on to more complex trees. Generally, we consider the trees in increasing order of their depth. The key observation is that the way the variables occurring in a subtree are assigned does not influence in any way the assignment of the variables outside this subtree (thus, enforcing a locality flavour) because a subtree only shares variables with its own subtrees.

Finally, the k-locality of strongly-nested patterns is considered. In particular, while it can be expected that the problem of computing the optimal k such that a pattern is k-local is intractable, it is shown that for strongly-nested patterns this can be done in polynomial time, and thus that they are not among the (expected) hard cases. Moreover, as a by-product of the algorithm, logarithmic bounds on the strict k-locality of strongly-nested patterns are given relative to their length, providing a clear formal comparison between the two classes.

The key to the algorithm is that, if we consider a factor $x..x$, it is assured that all other variables occurring in the factor do not occur outside as well, and thus the factor may be treated, to an extent, as independent from the rest of the pattern. Since such factors are fundamental to the remaining exposition, they are defined formally below.

▶ **Definition 23.** For a strongly-nested pattern $\alpha \in (X \cup \Sigma)^*$ and each $x \in \mathrm{var}(\alpha)$, let α_x be the shortest factor with $\alpha = \beta\alpha_x\gamma$ and $x \notin \mathrm{var}(\beta\gamma)$.

In other words, α_x is the factor of α from the leftmost to the rightmost occurrence of x. The approach is based around dynamic programming on the factors α_x, starting with the shortest (just blocks of the same variable). Precisely how this may be achieved is presented in Lemma 25 which gives the main recursive combinatorial insight. Firstly, however, it is necessary to distinguish between two types of marking sequences: those at which the edges are marked 'early' – i.e., as soon as the maximum number of marked blocks is reached – and those at which the edges are marked 'late'. Patterns permitting optimal marking sequences of the former variety are less likely to introduce a higher number of marked blocks, since at least one of the marked blocks may be absorbed by an existing marked block to the left/right.

▶ **Definition 24.** Let $\alpha \in (X \cup \Sigma)^+$ be a non-empty strictly k-local pattern. Let x and y be the leftmost and rightmost variables of α. Then α is *border priority markable* (BPM) if there exists an optimal marking sequence for α (in the sense that no other marking sequence exists which requires strictly fewer marked blocks, or equivalently, that the maximum number of marked blocks required is k), such that whenever there are k distinct marked blocks, x and y are marked.

▶ **Lemma 25.** *Let $\alpha \in X^*$ be a (terminal-free) non-empty strongly-nested pattern and let $x \in \mathrm{var}(\alpha)$. Then either $\alpha_x \in \{x\}^+$, or there exist $y_1, y_2, \dots y_n \in \mathrm{var}(\alpha)$ such that $\alpha_x \in x^{\mathfrak{b}}\alpha_{y_1}\alpha_{y_2}\dots\alpha_{y_n}x^{\mathfrak{b}}$. Moreover, suppose that α_{y_i} is strictly k_i-local for $i \in [n]$ and let $\mu = \max_{i\in[n]}\{k_i\}$. Then:*

- *α_x is (strictly) μ-local if and only if there exists exactly one $i \in [n]$ such that $k_i = \mu$ with α_{y_i} is not BPM. Otherwise α_x is strictly $(\mu+1)$-local.*
- *α_x is BPM if and only if there exists exactly one $i \in [n]$ such that $k_i = \mu$, there exists at most one $j \in [n]$ such that $k_j = \mu - 1$ with α_{y_j} is not BPM, and α_{y_i} is BPM.*

▶ **Theorem 26.** *Given a strongly-nested pattern $\alpha \in (X \cup \Sigma)^*$, the smallest value k such that α is k-local can be computed in polynomial time.*

In addition to Theorem 26, it is also possible to infer the following bound on the strict k-locality of strongly-nested patterns from Lemma 25. Note the contrast to the general case for which a pattern of length n may be strictly $\frac{n}{2}$-local, as witnessed e.g., by $(xy)^{\frac{n}{2}}$.

▶ **Theorem 27.** *Let $\alpha \in (X \cup \Sigma)^*$ be a strongly-nested pattern, and let $k \in \mathbb{N}$ with $k > \log|\alpha|$. Then α is k-local. Moreover, for each $k \in \mathbb{N}$, there exist strongly-nested patterns of length $2^{k+1} + 2^{k-1} - 4$ which are strictly k-local.*

Finally, we propose the following extension of strongly-nested patterns. Let Π be a class of patterns for which we can decide in polynomial time both whether a pattern α is in Π and Match. We define *strongly-Π-nested patterns* as follows. Patterns $\beta \in \Pi$ are strongly-Π-nested patterns. If β_1, β_2 are variable-disjoint patterns, $\beta_1 \in \Pi$ and β_2 is a strongly-Π-nested pattern, $x \in X \setminus \mathrm{var}(\beta_1)$, $\gamma_1, \gamma_2 \in (\Sigma \cup \{x\})^*$, and $\beta_2'\beta_2'' = \beta_2$, then $\beta_2'\gamma_1\beta_1\gamma_2\beta_2''$ is a strongly-Π-nested pattern (i.e., we just shuffle $\gamma_1\beta_1\gamma_2$ inside β_2 to obtain $\beta_2'\gamma_1\beta_1\gamma_2\beta_2''$). Note that for $\gamma_1 = \gamma_2 = \beta_2' = \varepsilon$, we obtain that $\beta_1\beta_2$ is a strongly-Π-nested pattern.

It is not hard to see that one can decide whether a pattern is strongly-Π-nested in polynomial time. Essentially, to test whether β is such a pattern, we need to decide whether $\beta \in \Pi$ or there exist $1 \leq i < j \leq |\beta|$ such that $j - i + 1 < |\beta|$, $\beta[i..j]$ consists of a nested Π-pattern surrounded by two blocks of a variable x, and $\beta[1..i-1]\beta[j+1..|\beta|]$ is in Π. This can be clearly implemented in polynomial time. A similar strategy works for matching strongly-Π-nested patterns. Assume we want to match β to a word w, with $|\beta| = m$ and $|w| = n$. If $\beta \in \Pi$, then we just check if β matches w. Otherwise, for β there exist $1 \leq i < j \leq m$ such that $j - i + 1 < m$, $\beta[i..j]$ consists of a strongly-Π-nested pattern surrounded by two patterns containing only the variable x, and $\beta[1..i-1]\beta[j+1..n] \in \Pi$. Then we match first $\beta[i..j]$ to some factor $w[i'..j']$ of w, and then check if $\beta[1..i-1]\beta[j+1..m]$, which is in Π, can be matched to $w[1..i'-1]w[j'+1..n]$. Again, this clearly works in polynomial time.

6 Conclusions

We introduce the classes of k-local and strongly-nested patterns. We give polynomial time algorithms (assuming k is treated as constant) for the membership of these classes and for the matching problem, in both cases gaining a significant improvement compared to existing algorithms for more general classes (i.e., those given in [25]). We have also considered the structure of patterns belonging to these classes, giving characterizations for patterns which are 1- and 2-local, as well as an optimized algorithm for matching 1-local patterns. We leave two interesting open problems outstanding, namely finding lower bounds for the time needed to decide whether a pattern is k-local, and for matching k-local patterns.

Acknowledgements. The authors wish to thank the referees of the paper for their helpful remarks and suggestions, in particular for their insightful comments on patterns with bounded treewidth which have helped to place the classes in the present paper more precisely within the literature.

References

1 Amihood Amir and Igor Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.

2 Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

3 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37, 2012.

4 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.

5 Thomas Erlebach, Peter Rossmanith, Hans Stadtherr, Angelika Steger, and Thomas Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261:119–156, 2001.

6 Henning Fernau, Florin Manea, Robert Mercaş, and Markus L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proc. 32nd Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 302–315, 2015.

7 Henning Fernau, Florin Manea, Robert Mercaş, and Markus L. Schmid. Revisiting Shinohara's algorithm for computing descriptive patterns. *Theoretical Computer Science*, 2016. Accepted for publication. Preliminary version: `http://www.informatik.uni-trier.de/~schmid/preprints/journals/2017_TCS_preprint.pdf` .

8 Henning Fernau and Markus L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Information and Computation*, 242:287–305, 2015.

9 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. *Theory of Computing Systems*, 2015.

10 Dominik D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory of Computing Systems*, 53:159–193, 2013.

11 Dominik D. Freydenberger and Daniel Reidenbach. Inferring descriptive generalisations of formal languages. *Journal of Computer and System Sciences*, 79:622–639, 2013.

12 Jeffrey E. F. Friedl. *Mastering Regular Expressions.* O'Reilly, Sebastopol, CA, third edition, 2006.

13 Juhani Karhumäki, Wojciech Plandowski, and Filippo Mignosi. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.

14 Michael Kearns and Leonard Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In *Proceedings of the 2nd Annual Conference on Learning Theory, COLT*, pages 57–71, 1989.

15 Satoshi Kobayashi and Takashi Yokomori. On approximately identifying concept classes in the limit. In *Proceedings of the 6th International Conference on Algorithmic Learning Theory, ALT*, volume 997 of *LNCS*, pages 298–312, 1995.

16 Dmitry Kosolobov, Florin Manea, and Dirk Nowotka. Detecting one-variable patterns. *CoRR*, abs/1604.00054, 2016. to appear in SPIRE 2017.

17 Stefan Lange and Rolf Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.

18 M. Lothaire. *Combinatorics on Words.* Cambridge University Press, 1997.

19 M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, 2002.

20 Alexandru Mateescu and Arto Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique Théoretique et Applications*, 28:233–253, 1994.

21 Zeinab Mazadi, Ziyuan Gao, and Sandra Zilles. Distinguishing pattern languages with membership examples. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications, LATA*, volume 8370 of *LNCS*, pages 528–540, 2014.

22 Yen K. Ng and Takeshi Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397:150–165, 2008.

23 Sebastian Ordyniak and Alexandru Popa. A parameterized study of maximum generalized pattern matching problems. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation, IPEC*, 2014.

24 Daniel Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.

25 Daniel Reidenbach and Markus L. Schmid. Patterns with bounded treewidth. *Information and Computation*, 239:87–99, 2014.

26 Rüdiger Reischuk and Thomas Zeugmann. Learning one-variable pattern languages in linear average time. In *Proceedings of the 11th Annual Conference on Computational Learning Theory, COLT*, pages 198–208, 1998.

27 Markus L. Schmid. A note on the complexity of matching patterns with variables. *Information Processing Letters*, 113(19):729–733, 2013.

28 Takeshi Shinohara. Polynomial time inference of pattern languages and its application. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 191–209, 1982.

29 Takeshi Shinohara and Setsuo Arikawa. Pattern inference. In K.P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*, volume 961 of *LNAI*, pages 259–291, 1995.

30 Takeshi Shinohara and Hiroki Arimura. Inductive inference of unbounded unions of pattern languages from positive data. *Theoretical Computer Science*, 241:191–209, 2000.