INSTITUT FÜR INFORMATIK

Efficient Computation of Descriptive Patterns

Henning Fernau, Florin Manea, Robert Mercaş,
Markus L. Schmid

Bericht Nr. 1405
May 8, 2014
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Efficient Computation of Descriptive Patterns

Henning Fernau, Florin Manea, Robert Mercaş, Markus L. Schmid

Bericht Nr. 1405
May 8, 2014
ISSN 2192-6247

e-mail: {flm,rgm}@informatik.uni-kiel.de, {Fernau, MSchmid}@uni-trier.de

Dieser Bericht ist als persönliche Mitteilung aufzufassen.

Efficient Computation of Descriptive Patterns

Henning Fernau[1], Florin Manea[2], Robert Mercaş[2], and Markus L. Schmid[1]

[1] Fachbereich IV – Abteilung Informatikwissenschaften, Universität Trier, D-54286 Trier, Germany, {Fernau, MSchmid}@uni-trier.de
[2] Kiel University, Department of Computer Science, D-24098 Kiel, Germany, {flm,rgm}@informatik.uni-kiel.de

Abstract. A pattern α is a word consisting of constants and variables and the pattern language $L(\alpha)$ (over an alphabet Σ) is the set of all words that can be obtained from α by uniformly replacing the variables with words over Σ. We investigate the problem of computing a pattern α that is descriptive of a given finite set $S \subseteq \Sigma^*$ of words, i. e., $S \subseteq L(\alpha)$ and there is no other pattern β with $S \subseteq L(\beta) \subset L(\alpha)$. A pattern α that is descriptive of a set S represents the structural commonalities of the words in S and, thus, can serve as a classifier with respect to this structure. Furthermore, (polynomial time) computability of descriptive patterns is sufficient for (polynomial time) inductive inference of pattern languages. We investigate the complexity of computing descriptive patterns and, for subclasses of patterns, we present efficient algorithms for computing them.

1 Introduction

The class of pattern languages was introduced by Angluin [1] as a formalism to describe similarities of words with respect to their repeating factors. For example, the words $w_1 := \mathtt{abbaabaa}$, $w_2 := \mathtt{baabbabaabba}$ and $w_3 := \mathtt{abaaaba}$ share the common feature of having a prefix that contains an occurrences of $\mathtt{ba}$ that is surrounded by exactly the same factor. These commonalities can be described by the pattern $\alpha = x_1\mathtt{ba}x_1x_2$, where x_1 and x_2 are variables that stand for arbitrary factors. The pattern language defined by α, denoted by $L(\alpha)$, is the set of all words that can be obtained from α by uniformly substituting the occurrences of variables x_1 and x_2 by some words. For example, the words w_1, w_2 and w_3 can be obtained from α by the substitutions $(x_1 \to \mathtt{ab}, x_2 \to \mathtt{aa})$, $(x_1 \to \mathtt{baab}, x_2 \to \mathtt{ba})$ and $(x_1 \to \mathtt{a}, x_2 \to \mathtt{aba})$, respectively, which shows $w_1, w_2, w_3 \in L(\alpha)$. In [23], Shinohara extends the concept of pattern languages to the case that variables can also be substituted by the empty word. Nowadays, these two variants are denoted by the expressions *nonerasing* (or NE-) pattern languages and *erasing* (or E-) pattern languages. In this work, we are mainly concerned with nonerasing pattern languages; for more informations about erasing pattern languages and the differences between E- and NE-pattern languages, the reader is referred to the surveys [15, 22].

The introduction of pattern languages was motivated by inductive inference from positive data, which was originally introduced by Gold in [7] and, informally

speaking, is the task of identifying the descriptor of a language by observing an enumeration of the words of the language. In [7], it is shown that every language class that contains all finite languages and at least one infinite language is not inferable from positive data, which renders inductive inference inappropriate for the classical language classes of the Chomsky hierarchy. However, Angluin brought new life to the field of inductive inference by characterising those language classes that are inferable from positive data in [2], and introducing the class of pattern languages in [1] as a prominent example of such a language class, thereby initiating a still active research field devoted to specific aspects of the learnability of pattern languages (see, e. g., Lange and Wiehagen [14], Wiehagen and Zeugmann [27], Reischuk and Zeugmann [20], Shinohara [24], Shinohara and Arimura [26], Mazadi et al. [16] and, for a survey, Shinohara and Arikawa [25]). The question whether erasing pattern languages can be learned in the model of inductive inference was open for two decades and has been answered in the negative by Reidenbach [17]. The class of pattern languages is also subject of ongoing studies in the context of formal language theory and many insights regarding their expressive power, decidability and complexity of decision problems and other language theoretical properties are known (for a survey, see [15, 22]).

In this paper, we are not primarily concerned with inductive inference of pattern languages, but with the problem of computing so-called *descriptive patterns*. A pattern α is said to be descriptive of a finite set S of words if $S \subseteq L(\alpha)$ and there is no other pattern β that describes S more accurately, i. e., $S \subseteq L(\beta) \subset L(\alpha)$. For example, the pattern $x_1\mathtt{baa}x_2x_3\mathtt{a}$ is descriptive of $S = \{w_1, w_2, w_3\}$, where the words w_i are as defined above, and the pattern $\alpha = x_1\mathtt{ba}x_1x_2$ introduced at the beginning of this section is not descriptive since $S \subseteq L(x_1\mathtt{ba}x_1x_2\mathtt{a}) \subset L(\alpha)$. Computing a descriptive pattern can be seen as the task of finding a good generalisation or approximation – in the form of a pattern – of the common structure of finitely many words, where the optimality of the approximation is described by the concept of descriptiveness. Descriptive patterns have already been introduced in Anguin's introductory paper [1] and, furthermore, as pointed out in [2], there is a strong connection between the computation of descriptive patterns and the inductive inference of pattern languages, since an algorithm for computing descriptive patterns can be easily transformed into an algorithm that infers pattern languages from positive data.[3]

From a practical point of view, a descriptive pattern α may serve as a representation of the structural commonalities of some set S of textual data (e. g., employee files, entries of a bibliographical database, etc.) and in order to check whether a new data element meets this common structure, it is sufficient to check whether it can be generated by α. Or, from a machine learning perspective, the set S is a training set from which the classifier α is obtained. The main obstacle of this application of descriptive patterns is that deciding on whether a given word can be generated by a given pattern, i. e., the membership problem

[3] The computation of descriptive patterns coincides with the so-called MINL calculation for the class of pattern languages (see Shinohara and Arikawa [25]).

of pattern languages, is $\mathcal{NP}$-complete [1]. Moreover, this particularly affects the complexity of computing descriptive patterns, since in order to do so, it seems necessary to perform the membership problem for pattern languages. In fact, it has been shown by Angluin in [1] that an algorithm computing a descriptive pattern of *maximal size* necessarily solves the membership problem and therefore, assuming $\mathcal{P} \neq \mathcal{NP}$, it cannot have a polynomial running time. Consequently, for practical applications, descriptive patterns have two disadvantages: computing them as well as using them as a tool for classification is both a computationally hard task.

A fruitful approach of how to deal with this computational intractability is provided by Shinohara [24], who defines subclasses of patterns for which the membership problem is solvable in polynomial time. The concept of descriptiveness can be easily restricted to an arbitrary subclass Π of patterns; more precisely, a pattern α is Π-descriptive if $\alpha \in \Pi$, $S \subseteq L(\alpha)$ and there is no other pattern $\beta \in \Pi$ with $S \subseteq L(\beta) \subset L(\alpha)$. Shinohara shows that the polynomial time decidability of the membership problem for his subclasses of patterns directly entails polynomial time computability of descriptive patterns for these classes (and therefore also polynomial time inference of the corresponding classes of pattern languages).

In this paper, we unify and further extend both Angluin's insights with respect to the hardness of computing descriptive patterns as well as Shinohara's work on their efficient computation as follows. We say that a class of patterns is *rich* if it contains the set of the most primitive patterns $\{x_1x_2 \dots x_k \mid k \in \mathbb{N}\}$ (note that these patterns correspond to the simple language class $\{\{w \mid |w| \geq k\} \mid k \in \mathbb{N}\}$). One of our main results is a meta-theorem which states that, under the assumption $\mathcal{P} \neq \mathcal{NP}$, for every rich class Π of patterns the problem of computing Π-descriptive patterns can be solved in polynomial time if and only if the question whether $\alpha \in \Pi$ can be decided in polynomial time and the membership problem for Π can be decided in polynomial time. It is intuitively clear that if Π is a rather small class of patterns, then a Π-descriptive pattern does not very accurately describe the structure of the words in the set S and, on the other hand, if Π is a large class of patterns, then computing Π-descriptive patterns and solving the membership problem is typically more difficult. With regard to this trade-off between accuracy and complexity, it can be observed that Shinohara's classes of patterns (see [24]) are fairly simple, i. e., the class of *regular patterns*, where every variable has only one occurrence (e. g., $x_1\mathtt{ab}x_2x_3\mathtt{a}x_4$), and the class of *non-cross* patterns, where the occurrences of variables are sorted by their index (e. g., $x_1\mathtt{a}x_1x_1x_2\mathtt{b}x_2x_3\mathtt{ab}x_3x_3$). On the other hand, computing descriptive patterns and solving the membership problem for these subclasses can be done quite efficiently.

While computing descriptive patterns for the classes of regular and non-cross patterns can be done quite efficiently, these classes have the disadvantage of being rather strongly restricted, which means that descriptive regular or non-cross patterns do not very accurately represent the common structure of the words in a set S. To this end, we also investigate the classes of patterns with a bounded

number of repeated variables (denoted by $\text{PAT}^{\text{r}}_{\text{var}\leq k}$) and the classes of patterns with a bounded *scope coincidence degree* (denoted by $\text{PAT}_{\text{scd}\leq k}$). The parameter k in this algorithm is the *scope coincidence degree*, introduced by Reidenbach and Schmid in [18]. Intuitively speaking, the scope coincidence degree measures the extent of disorder with respect to the variable occurrences of the patterns, e. g., the variable occurrences in a non-cross pattern are completely ordered; thus, non-cross patterns have a scope coincide degree of 1. While the existence of such a polynomial time algorithm follows from generalisations of known results, the focus of our work is an efficient implementation by sophisticated algorithmic techniques.Our respective main results are that, for a given finite set S of words, both $\text{PAT}^{\text{r}}_{\text{var}\leq k}$-descriptive patterns and $\text{PAT}_{\text{scd}\leq k}$-descriptive patterns can be computed in $\mathcal{O}(\frac{n^{2k}m^2}{((k-1)!)^2})$ time, where n is the total length of the words in the set S and m is the length of its shortest word.

We conclude this paper by pointing out in more detail some connections of our results and the polynomial time inductive inference of pattern languages.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, and Σ be a finite alphabet of *symbols*. A *word* (over Σ) is any sequence of symbols from Σ. For any word w over Σ, $|w|$ denotes its length and ε denotes the *empty word*, i. e., $|\varepsilon| = 0$. By Σ^+ we denote the set of all non-empty words over Σ and $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$. For the *concatenation* of two words w_1, w_2 we write w_1w_2. The concatenation of k words $w_1, w_2 \ldots, w_k$ is written $\Pi_{i=1,k}w_i$. If $w = w_i$, for every $1 \leq i \leq k$, this represents the kth power of w and is denoted by w^k. Furthermore, w is primitive if it cannot be expressed as a power ℓ of any other word, for $\ell > 1$.

Let $w \in \Sigma^*$ be a word. We say that $v \in \Sigma^*$ is a *factor* of w if $w = u_1vu_2$ for some $u_1, u_2 \in \Sigma^*$. If $u_1 = \varepsilon$, or $u_2 = \varepsilon$, then v is a *prefix*, or a *suffix*, respectively, of w. For any $v \in \Sigma^+$, by $|w|_v$ we denote the number of occurrences of v in w. If S is a set of words, by $\text{symb}(w)$ and $\text{symb}(S)$ we denote the smallest set B with $w \in B^*$ and $S \subseteq B^*$, resp. For each $1 \leq i \leq j \leq |w|$, let $w[i..j] = w[i]w[i+1]\cdots w[j]$, where $w[k]$ represents the letter on position k in w, for $1 \leq k \leq |w|$.

Let $\alpha \in (\Sigma \cup X)^*$ be a pattern. The *pattern language of* α (*over* Σ) is defined by $L_\Sigma(\alpha) = \{h(\alpha) \mid h : (\Sigma \cup X)^* \to \Sigma^* \text{ is a nonerasing terminal substitution}\}$. For an alphabet Σ, by Σ-PAT, we denote the set of patterns over the alphabet Σ, i. e., $\Sigma\text{-PAT} = (\Sigma \cup X)^+$; if Σ is clear from the context or negligible, then we also write PAT instead of Σ-PAT. For any alphabet Σ and any class of patterns $\Pi \subseteq \Sigma\text{-PAT}$, $\mathcal{L}_\Sigma(\Pi) = \{L_\Sigma(\alpha) \mid \alpha \in \Pi\}$ is the class of *Σ-Π-pattern languages*.

We say that a pattern α is in *canonical form* if and only if, for some $k \in \mathbb{N}$, $\text{var}(\alpha) = \{x_1, x_2, \ldots, x_k\}$ and, for every i, $1 \leq i \leq k-1$, the leftmost occurrence of x_i is to the left of the leftmost occurrence of x_{i+1}. Intuitively, we get a pattern in canonical form, by writing it from left to right and every time we need a new variable, we take the one with the smallest possible index that has not been used so far. We shall always assume that patterns are in canonical form.

A class $\Pi \subseteq \Sigma$-PAT of patterns is *natural* if, for a given pattern α, the question $\alpha \in \Pi$ is decidable and the pattern x_1 is contained in Π. A natural class Π of patterns is *rich*, if $\{x_1 x_2 \dots x_k \mid k \in \mathbb{N}\} \subseteq \Pi$. Next, we define some specific classes of patterns.

A pattern α is *regular* if, for every $x \in \mathrm{var}(\alpha)$, $|\alpha|_x = 1$, and the class of regular patterns is denoted by $\mathrm{PAT}_{\mathrm{reg}}$. For any $k \in \mathbb{N}$, a *k-variable* pattern is a pattern α that satisfies $|\mathrm{var}(\alpha)| \leq k$ and a pattern β with $|\{x \in \mathrm{var}(\beta) \mid |\beta|_x \geq 2\}| \leq k$ is a *k-repeated-variable* pattern. For every $k \in \mathbb{N}$, $\mathrm{PAT}_{\mathrm{var}\leq k}$ and $\mathrm{PAT}^{\mathsf{r}}_{\mathrm{var}\leq k}$ denotes the set of k-variable patterns and k-repeated-variable patterns, respectively. 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 some variables $y_1, y_2, \dots, y_k \in \mathrm{var}(\alpha)$ *coincide in α* if $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. The *scope coincidence degree* of α ($\mathrm{scd}(\alpha)$ for short) is the maximum number of variables in α such that their scopes coincide. For every $k \in \mathbb{N}$, let $\mathrm{PAT}_{\mathrm{scd}\leq k} = \{\alpha \in (\Sigma \cup X)^*, \mathrm{scd}(\alpha) \leq k\}$. 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 $\alpha_2 = x_1\, x_2\, x_1\, x_1\, x_2\, x_3\, x_2\, x_3\, x_3$. As can be easily verified, $\mathrm{sc}_{\alpha_1}(x_1) = \{1, 2, \dots, 7\}$, $\mathrm{sc}_{\alpha_1}(x_2) = \{2, 3, \dots, 8\}$, $\mathrm{sc}_{\alpha_1}(x_3) = \{4, 5, \dots, 9\}$ and therefore the scopes of all 3 variables coincide in α_1, i. e., $\mathrm{scd}(\alpha_1) = 3$. On the other hand, $\mathrm{sc}_{\alpha_2}(x_1) = \{1, 2, \dots, 4\}$, $\mathrm{sc}_{\alpha_2}(x_2) = \{2, 3, \dots, 7\}$, $\mathrm{sc}_{\alpha_2}(x_3) = \{6, 7, \dots, 9\}$. Hence the scopes of x_1 and x_3 do not coincide in α_2, and $\mathrm{scd}(\alpha_2) = 2$.

By definition, $\mathrm{PAT}_{\mathrm{reg}}$ and, for every $k \in \mathbb{N}$, $\mathrm{PAT}_{\mathrm{scd}\leq k}$ and $\mathrm{PAT}^{\mathsf{r}}_{\mathrm{var}\leq k}$ are rich classes of patterns, whereas $\mathrm{PAT}_{\mathrm{var}\leq k}$ is a natural class of patterns, but not a rich one. Moreover, $\mathrm{PAT}_{\mathrm{scd}\leq 1}$ coincides with the class of non-cross patterns (see [24]), which we denote by $\mathrm{PAT}_{\mathrm{nc}}$. If in a non-cross pattern α no constants occur between two occurrences of the same variable, then α is *strictly* non-cross; the class of strictly non-cross patterns is denoted by $\mathrm{PAT}_{\mathrm{snc}}$.

2.1 Preliminary Results

The binary relations $\sqsubseteq$ and $\equiv$ on PAT, introduced in [1], are defined in the following way. For every $\alpha, \beta \in \mathrm{PAT}$, $\alpha \sqsubseteq \beta$ if and only if there exists a non-erasing substitution h with $h(\beta) = \alpha$ and $\alpha \equiv \beta$ if and only if there exists a non-erasing *renaming of variables* h with $h(\beta) = \alpha$, i. e., h is a 1-uniform non-erasing substitution. For patterns that are in canonical form, $\alpha \equiv \beta$ if and only if $\alpha = \beta$; thus, since we made the assumption that patterns are always in canonical form, we write $\alpha = \beta$ instead of $\alpha \equiv \beta$.

Lemma 1 (Angluin [1]). *Let $\alpha, \beta \in \mathrm{PAT}$.*

- *$\sqsubseteq$ is transitive.*
- *If $\alpha \sqsubseteq \beta$, then $L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$.*
- *$\alpha = \beta$ if and only if $\alpha \sqsubseteq \beta$ and $\beta \sqsubseteq \alpha$.*
- *If $|\alpha| = |\beta|$ and $L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$, then $\alpha \sqsubseteq \beta$.*

Lemma 1 shows that $\alpha \sqsubseteq \beta$ is sufficient for $L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$ and the example $L_{\{\mathtt{a},\mathtt{b}\}}(\mathtt{a}x\mathtt{ba}xx\mathtt{b}) \subseteq L_{\{\mathtt{a},\mathtt{b}\}}(xxy)$ and $\mathtt{a}x\mathtt{ba}xx\mathtt{b} \not\sqsubseteq xxy$ from [1] points out that

$\alpha \sqsubseteq \beta$ is not a necessary for $L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$. However, for the special case that $|\alpha| = |\beta|$, $\alpha \sqsubseteq \beta$ is characteristic for $L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$. In particular, since patterns describing the same nonerasing pattern language must have the same length, this implies that $\alpha = \beta$ if and only if $L_\Sigma(\alpha) = L_\Sigma(\beta)$.

Next, we recall some important known results about the complexity of the *membership problem for Π-pattern languages*, i. e., the problem to decide for a given pattern $\alpha \in \Pi$ and a word w, whether or not $w \in L_\Sigma(\alpha)$, where $\Sigma := \mathrm{symb}(w) \cup \mathrm{term}(\alpha)$ and Π is some class of patterns.

Theorem 1 (Angluin [1]). *The membership problem for* PAT*-pattern languages is $\mathcal{NP}$-complete.*[4]

Theorem 2 (Reidenbach, Schmid [19]). *Let $k \in \mathbb{N}$. The membership problem for $\mathrm{PAT}_{\mathrm{scd}\leq k}$-pattern languages is solvable in polynomial time.*

Since $\mathrm{PAT}_{\mathrm{reg}} \subseteq \mathrm{PAT}_{\mathrm{scd}\leq 1}$ and, for every $k \in \mathbb{N}$, $\mathrm{PAT}_{\mathrm{var}\leq k} \subseteq \mathrm{PAT}^{\mathsf{r}}_{\mathrm{var}\leq k} \subseteq \mathrm{PAT}_{\mathrm{scd}\leq k+1}$, the statement of Theorem 2 also holds for the classes $\mathrm{PAT}_{\mathrm{reg}}$, $\mathrm{PAT}_{\mathrm{var}\leq k}$ and $\mathrm{PAT}^{\mathsf{r}}_{\mathrm{var}\leq k}$.

2.2 Descriptive Patterns

Let Σ, Σ' be alphabets with $\Sigma \subseteq \Sigma'$, let $S \subseteq \Sigma^*$ be finite and let $\Pi \subseteq \Sigma$-PAT. A pattern α is *Σ'-Π-descriptive of S* if $\alpha \in \Pi$, $S \subseteq L_{\Sigma'}(\alpha)$ and there does not exist a $\beta \in \Pi$ with $S \subseteq L_{\Sigma'}(\beta) \subset L_{\Sigma'}(\alpha)$. In the following, we call a finite set S of words over an alphabet Σ a *sample* (*over Σ*).

Example 1 (Angluin [1]). Let $\Sigma := \{\mathtt{a}, \mathtt{b}\}$ and $S := \{\mathtt{aabaa}, \mathtt{babab}, \mathtt{aabab}, \mathtt{babaa}\}$. The pattern $\alpha := x\mathtt{aba}y$ is Σ-PAT-descriptive of S. Furthermore, the pattern $\beta := xxy$ is also Σ-PAT-descriptive of S.

Example 2 (Freydenberger and Reidenbach [6]). Let $\Sigma := \{\mathtt{a}, \mathtt{b}\}$, let $S := \{\mathtt{ababa}, \mathtt{ababbababbab}, \mathtt{babab}\}$, let $\alpha := xyxyx$ and $\beta := x\mathtt{ab}y$. Both α and β are Σ-PAT-descriptive of S.

Example 3. Let $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $S := \{\mathtt{aabcaaa}, \mathtt{caacbca}, \mathtt{bbcccbbbc}\}$. Then $xy\mathtt{c}zx$ is Σ-$\mathrm{PAT}_{\mathrm{var}\leq 2}$-descriptive of S and $x\mathtt{c}x$ is Σ-$\mathrm{PAT}_{\mathrm{var}\leq 1}$-descriptive of S.

Our definition of Σ-Π-descriptiveness allows the situation that Σ contains symbols that do not occur in any word of the sample S. We shall now show that this is not necessary, i. e., if a pattern is Σ-Π-descriptive of a sample $S \subseteq \Sigma^*$, then the same pattern is also Σ'-Π-descriptive of S, for any alphabet Σ' with $\Sigma \subseteq \Sigma'$.

Proposition 1. *Let Σ and Σ' be finite alphabets with $\Sigma \subseteq \Sigma'$, let S be a finite sample over Σ and let $\Pi \subseteq \Sigma$-PAT. If a pattern α is Σ-Π-descriptive of S, then α is also Σ'-Π-descriptive of S.*

[4] Note that in [1] a stronger result is shown, i. e., the membership problem is $\mathcal{NP}$-complete even if the terminal alphabet Σ is a fixed binary alphabet.

Proof. We first note that, for any patterns $\gamma, \delta \in (\Sigma \cup X)^+$, $L_{\Sigma'}(\gamma) \subseteq L_{\Sigma'}(\delta)$ implies $L_\Sigma(\gamma) \subseteq L_\Sigma(\delta)$. This is due to the fact that, for every $w \in \Sigma^*$ and every $\pi \in \{\gamma, \delta\}$, $w \in L_\Sigma(\pi)$ if and only if $w \in L_{\Sigma'}(\pi)$.

In order to prove the statement of the lemma, we assume that α is Σ-PAT-descriptive of S. Since $L_\Sigma(\alpha) \subseteq L_{\Sigma'}(\alpha)$, this implies $S \subseteq L_{\Sigma'}(\alpha)$. Now we assume to the contrary that α is not Σ'-PAT-descriptive of S, which means that there exists a pattern β with $S \subseteq L_{\Sigma'}(\beta) \subset L_{\Sigma'}(\alpha)$. This implies that $\alpha \neq \beta$ and, with the observation from above, $L_\Sigma(\beta) \subseteq L_\Sigma(\alpha)$; thus, $S \subseteq L_\Sigma(\beta) \subset L_\Sigma(\alpha)$. Hence α is not Σ-PAT-descriptive of S, which is a contradiction. □

In the following, we say that a pattern is Π-descriptive of S if it is Σ-Π-descriptive for the smallest Σ with $S \subseteq \Sigma^*$.

We conclude this section by pointing out that descriptive patterns for *erasing* pattern languages as well as descriptive patterns for *infinite* samples have also been investigated (see Jiang et al. [9] and Freydenberger and Reidenbach [6]).

3 The Complexity of Computing Descriptive Patterns

In this section, we investigate the problem of computing a Π-descriptive pattern for a given sample S. First, by extending a result from [1], we show that this problem is solvable if and only if Π is a natural class of patterns. We then show that any algorithm that computes Π-descriptive patterns implicitly decides whether a given pattern is in Π and, more importantly, solves the membership problem for Π-pattern languages. This implies that if for Π one of these problems is computationally hard, then this hardness carries over to the problem of computing Π-descriptive patterns. In the remainder of the section, by slightly generalising an algorithm introduced by Shinohara in [24], we show that, for every richt class Π of patterns, a polynomial time algorithm for the membership problem for Π-pattern languages and a polynomial time algorithm deciding whether a given pattern is in Π can be combined to a polynomial time algorithm that computes Π-descriptive patterns. These results directly imply the meta-theorem already mentioned in Section 1.

Let $S \in \Sigma^*$ be a sample and let $m := \min\{|w| \mid w \in S\}$. By definition, for every pattern α, $S \subseteq L_\Sigma(\alpha)$ implies $|\alpha| \leq m$. Hence, an obvious approach to find a pattern that is descriptive of S is to enumerate all finitely many patterns α with $S \subseteq L_\Sigma(\alpha)$ and search all these patterns for one that is minimal with respect to the subset relation of the corresponding pattern languages. However, this approach cannot be carried out, since, for two given patterns α and β, the question whether $L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$ is undecidable (see [5]). In [1], it is shown that it is sufficient to only investigate the patterns α with $S \subseteq L_\Sigma(\alpha)$ of maximal size in order to find a pattern that is descriptive of S and this idea can be easily extended to natural classes $\Pi \subseteq$ PAT.

Theorem 3. *Let $\Pi \subseteq$ PAT. There is an effective procedure that, for a given finite sample S, computes a pattern that is Π-descriptive of S if and only if Π is natural.*

Proof. In order to prove the *if* direction, we assume that Π is natural. Let Σ be the smallest alphabet with $S \subseteq \Sigma^*$. The procedure works as follows. We first compute the set Q of all patterns α that satisfy $\alpha \in \Pi$ and $S \subseteq L_\Sigma(\alpha)$ by enumerating all patterns $\alpha \in (\Sigma \cup X)^*$ up to length $m := \min\{|w| \mid w \in S\}$ and checking whether $\alpha \in \Pi$ and $S \subseteq L_\Sigma(\alpha)$. Deciding $\alpha \in \Pi$ can be done since Π is natural and $S \subseteq L_\Sigma(\alpha)$ is decidable since S is finite and the membership problem for Π-pattern languages is decidable. For any pattern α, if $|\alpha| > m$, then $S \nsubseteq L_\Sigma(\alpha)$; thus, the construction of Q from above is correct and Q is finite. Since Π contains x_1 and $S \subseteq L_\Sigma(x_1)$, Q is non-empty.

Now let $Q_{\max} \subseteq Q$ contain all elements of Q with maximum length. Next, we compute a pattern $\beta \in Q_{\max}$, which is minimal for the set $Q_{\max}$ with respect to $\sqsubseteq$, which can be done by computing the relation $\sqsubseteq$ for the whole set $Q_{\max}$. Now if β is not Π-descriptive of S, then there exists an $\alpha \in Q$ with $S \subseteq L_\Sigma(\alpha) \subset L_\Sigma(\beta)$. If $\alpha \in Q_{\max}$, then, since $|\alpha| = |\beta|$, $\alpha \sqsubseteq \beta$ is implied, which contradicts the fact that β is minimal with respect to $Q_{\max}$ and $\sqsubseteq$. On the other hand, if $\alpha \notin Q_{\max}$, then $|\alpha| < |\beta|$, which is a contradiction to $L_\Sigma(\alpha) \subset L_\Sigma(\beta)$. Thus, β is Π-descriptive of S.

In order to prove the *only if* direction, we assume that there is an effective procedure χ that, for a given sample S, computes a pattern that is Π-descriptive of S and we show that this implies that Π is natural. We first note that if $x_1 \notin \Pi$, then there exists no $\alpha \in \Pi$ with $\{\mathtt{a}, \mathtt{b}\} \in L_{\{\mathtt{a},\mathtt{b}\}}(\alpha)$, which leads to the contradiction that, on input $\{\mathtt{a}, \mathtt{b}\}$, χ does not compute a pattern that is Π-descriptive of $\{\mathtt{a}, \mathtt{b}\}$. Furthermore, for an arbitrary pattern α, $\alpha \in \Pi$ if and only if χ computes α on input sample $\{\alpha\}$. Hence, for every pattern α, the question $\alpha \in \Pi$ is decidable. This shows that Π is natural. □

The procedure of the proof of Theorem 3 computes a descriptive pattern with maximal length. This is due to the fact that for all patterns with maximal length that describe the sample S, the inclusion of their corresponding pattern languages is characterised by the relation $\sqsubseteq$, which is computable. Thus, a descriptive pattern can be found by an exhaustive search of all these patterns with maximal length. Angluin shows in [1] that, if $\mathcal{P} \neq \mathcal{NP}$, then it is not possible to compute descriptive patterns of maximum length in polynomial time:

Theorem 4 (Angluin [1]). *Let $\{\mathtt{a}, \mathtt{b}\} \subseteq \Sigma$. If $\mathcal{P} \neq \mathcal{NP}$, then there is no polynomial time algorithm that, for a given finite sample S over Σ, computes a pattern of maximum possible length that is* Pat*-descriptive of S.*

We wish to emphasise that the result of Theorem 4 holds for any fixed alphabet that is at least binary. We can prove a variant of Theorem 4 that is stronger in the sense that it does not need the maximality condition and the size of S can be restricted to 2, but weaker in the sense that the alphabet is considered a part of the input and is not fixed. More precisely, we can prove that, for any natural class Π of patterns, if it is possible to compute in polynomial time a Π-descriptive pattern (of any length) for a given sample of size 2 over some alphabet, then the membership problem for Π-pattern languages can be solved in polynomial time.

Lemma 2. *Let Π be a natural class of patterns. If there exists a polynomial time algorithm that, for a given sample S of size 2, computes a pattern that is Π-descriptive of S, then the membership problem for Π-pattern languages is decidable in polynomial time.*

Proof. Let $\Sigma := \{\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_k\}$ be some alphabet, let $\alpha \in ((\Sigma \cup X)^* \cap \Pi)$ and let $w \in \Sigma^*$. We define $\Sigma' := \Sigma \cup \mathrm{var}(\alpha)$ and prove the following claim.

Claim 1. $w \in L_\Sigma(\alpha)$ if and only if α is the only Π-descriptive pattern for $\{\alpha, w\}$.

Proof (Claim 1). In order to prove the *if* direction, we assume that α is Π-descriptive of $\{\alpha, w\}$, which directly implies $w \in L_\Sigma(\alpha)$.

Next, we prove the *only if* direction of the statement by contraposition. To this end, we assume that α is not Π-descriptive of $\{\alpha, w\}$ or α is not the only pattern that is Π-descriptive of $\{\alpha, w\}$. In any case, since Π is natural, this implies that there exists a pattern γ, with $\alpha \neq \gamma$, that is Π-descriptive of $\{\alpha, w\}$. Consequently, $\alpha \in L_{\Sigma'}(\gamma)$ and therefore $\alpha \sqsubseteq \gamma$ and $L_{\Sigma'}(\alpha) \subseteq L_{\Sigma'}(\gamma)$. Furthermore, since $\alpha \neq \gamma$, $L_{\Sigma'}(\alpha) \neq L_{\Sigma'}(\gamma)$ and, thus, we can conclude that $L_{\Sigma'}(\alpha) \subset L_{\Sigma'}(\gamma)$. Now if $w \in L_\Sigma(\alpha)$, then also $w \in L_{\Sigma'}(\alpha)$, which implies $\{\alpha, w\} \subseteq L_{\Sigma'}(\alpha)$. Together with the observation that $L_{\Sigma'}(\alpha) \subset L_{\Sigma'}(\gamma)$ from above, this leads to the contradiction that γ is not Π-descriptive of $\{\alpha, w\}$. Hence, if α is not Π-descriptive of $\{\alpha, w\}$ or α is not the only pattern that is Π-descriptive of $\{\alpha, w\}$, then $w \notin L_\Sigma(\alpha)$. (Claim 1) □

We now assume that there exists an algorithm χ, which, for a given sample S of size 2, computes in polynomial time a pattern that is Π-descriptive of S. Hence, in order to decide in polynomial time whether or not $w \in L_\Sigma(\alpha)$, it is sufficient to run χ on the input sample $\{\alpha, w\}$ in order to compute a Π-descriptive pattern γ of $\{\alpha, w\}$.With the claim from above, $\gamma = \alpha$ if and only if $w \in L_\Sigma(\alpha)$, which means that we can solve the membership problem for Π-pattern languages in polynomial time. □

The proof of Theorem 3 suggests that in order to compute Π-descriptive patterns, it is necessary to solve the membership problem for Π-pattern languages. Lemma 2 confirms this intuition by showing that (at least for the case where the cardinality of the sample alphabet is not fixed) independent of how an algorithm that computes descriptive patterns may work, it always implicitly solves the membership problem.

The next lemma shows a similar result, but with respect to the question whether a pattern α is a member of a class Π of patterns.

Lemma 3. *Let Π be a natural class of patterns. If there exists a polynomial time algorithm that, for a given sample S, computes a pattern that is Π-descriptive of S, then there exists a polynomial time algorithm that, for a given pattern α, decides whether $\alpha \in \Pi$.*

Proof. We assume that there exists a polynomial time algorithm χ that, for a given sample S, computes a pattern that is Π-descriptive of S. Now let α be

some pattern and let $\Sigma := \mathrm{symb}(\alpha)$. We can use χ in order to compute a pattern β that is Π-descriptive of $\{\alpha\}$. Obviously, $\{\alpha\} \subseteq L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$. If $\alpha = \beta$, then $\alpha \in \Pi$ and if $\alpha \neq \beta$, then $\{\alpha\} \subseteq L_\Sigma(\alpha) \subset L_\Sigma(\beta)$, which implies that α cannot be in Π, since then β is not Π-descriptive of $\{\alpha\}$. Thus, $\alpha \in \Pi$ if and only if $\alpha = \beta$ and therefore we can decide in polynomial time whether $\alpha \in \Pi$. □

Lemmas 2 and 3 show that for natural classes Π of patterns the computational hardness of answering the question whether α is in Π and of solving the membership problem for Π-pattern languages carries over to the problem of computing Π-descriptive patterns. In the following, we show that, at least for rich classes Π of patterns, polynomial time solvability of these problems carries over as well.

In [24], Shinohara presents an exponential time algorithm that computes Pat-descriptive patterns. Furthermore, in [24] it is shown that this algorithm can be restricted in such a way that it computes in polynomial time descriptive patterns for the classes of regular pattern languages and non-cross pattern languages. We restate Shinohara's algorithm in a slightly modified form, i. e., as a generic algorithm that computes Π-descriptive patterns for an arbitrary rich class Π of patterns (see Algorithm 1).

Algorithm 1: Π-DescPat

Input : A sample $S \in \Sigma^*$, a shortest word w of S.
Output: A Π-descriptive pattern

```
1  m := |w|, α_1 := x_1 x_2 ... x_m;
2  for i := 1 to m do
3  |  q := true, j := 1;
4  |  if α_i[x_i → w[i]] ∈ Π and S ⊆ L_Σ(α_i[x_i → w[i]]) then
5  |  |  α_{i+1} := α_i[x_i → w[i]] and q := false;
6  |  while j ≤ i and q do
7  |  |  if x_j ∈ var(α[1, i − 1]), α_i[x_i → x_j] ∈ Π and S ⊆ L_Σ(α_i[x_i → x_j]) then
8  |  |  |  α_{i+1} := α_i[x_i → x_j] and q := false;
9  |  |  else
10 |  |  |  j := j + 1;
11 return α_{m+1}
```

The algorithm Π-DescPat works as follow. We start with a pattern $\alpha := x_1 x_2 \dots x_m$, where m is the length of a shortest word w in the sample S. Then we move over α from left to right and at every position i, we try to refine α if possible. This refinement is done by first trying to replace x_i by the i^{th} symbol of w (Line 4) and then consecutively by all of the variables that already occur in the prefix $\alpha[1..i-1]$ (Line 7). As soon as one of these refinements yields a pattern that describes the sample S, we move on to the next position and if all refinements fail in this regard, then we keep variable x_i at position i (which means that x_i occurs in the final pattern that is computed). Furthermore, in

order to make sure that a Π-descriptive pattern is computed, a refinement of x_i is only carried out if it yields a pattern that is still in Π.

The following result establishes the correctness of Π-DescPat and states that its running time is polynomial if both $\alpha \in \Pi$ can be decided in polynomial time and the membership problem for Π-pattern languages can be solved in polynomial time.

Lemma 4. *Let Π be a rich class of patterns, let S be a sample and let α the output of Π-DescPat on input S. Then α is Π-descriptive of S. Furthermore, if deciding whether a pattern β is in Π can be done $\mathcal{O}(f(|\beta|))$ time and deciding, for $\beta \in \Pi$, whether $w \in L(\beta)$ can be done in $g(|w|, |\beta|)$ time, then Π-DescPat runs in $\mathcal{O}(|\alpha|^2(f(|\alpha|) + \sum_{w \in S} g(|w|, |\alpha|))$.*

Proof. We first prove that α is Π-descriptive of S. To this end, let $w \in S$, such that $|w| = \min\{|u| \mid u \in S\}$ and let $m = |w|$. For every i, $1 \leq i \leq m+1$, let α_i be the pattern at the beginning of the i^{th} iteration of the main loop of Π-DescPat; in particular, $\alpha_1 = x_1 x_2 \dots x_m$ and $\alpha_{m+1} = \alpha$.

Claim 1. $S \subseteq L_\Sigma(\alpha)$ and $\alpha \in \Pi$.

Proof (Claim 1). Following the algorithm's definition, for every i, $1 \leq i \leq m$, $S \subseteq L_\Sigma(\alpha_i)$ implies $S \subseteq L_\Sigma(\alpha_{i+1})$ and $\alpha_i \in \Pi$ implies $\alpha_{i+1} \in \Pi$. Since $S \subseteq L_\Sigma(\alpha_1)$ and $\alpha_1 \in \Pi$ is satisfied, $S \subseteq L_\Sigma(\alpha)$ and $\alpha \in \Pi$ follows. (Claim 1) □

For the sake of contradiction, we assume that α is not Π-descriptive of S, i. e., there exists a pattern $\beta \in \Pi$ with $S \subseteq L_\Sigma(\beta) \subset L_\Sigma(\alpha)$.

Claim 2. $|\alpha| = |\beta|$ and $\beta \sqsubseteq \alpha$.

Proof (Claim 2). If $|\alpha| < |\beta|$, then $w \notin L_\Sigma(\beta)$, and if $|\beta| < |\alpha|$, then $L_\Sigma(\beta) \nsubseteq L_\Sigma(\alpha)$. Thus, $|\alpha| = |\beta|$. Furthermore, since $L_\Sigma(\beta) \subset L_\Sigma(\alpha)$ and $|\alpha| = |\beta|$, Lemma 1 implies $\beta \sqsubseteq \alpha$. (Claim 2) □

Without loss of generality, we can assume that, for every i, $1 \leq i \leq m$, if $\beta[i] = x_j$ and $|\beta[1..i-1]|_{x_j} = 0$, then $i = j$ (note that α has this property, too). Since $L_\Sigma(\beta) \subset L_\Sigma(\alpha)$, $\alpha \neq \beta$ is implied; thus, there exists a p, $1 \leq p \leq |\alpha|$, with $\alpha[p] \neq \beta[p]$ and $\alpha[1..p-1] = \beta[1..p-1]$. As stated by Claim 2, $\beta \sqsubseteq \alpha$, which implies that $\alpha[p] = x_q$, for some $x_q \in \text{var}(\alpha)$ and $\beta \sqsubseteq \alpha[x_q \mapsto z]$, for some $z \in \text{var}(\beta) \cup \Sigma$. In particular, since $\alpha[1..q-1] x_q x_{q+1} \dots x_m = \alpha_q$, this also means that $\beta \sqsubseteq \alpha_q[x_q \mapsto z]$ and therefore $S \subseteq L_\Sigma(\alpha_q[x_q \mapsto z])$.

Since $x_q \in \text{var}(\alpha)$, position q is the first occurrence of x_q in α and since $\beta[q] = z \neq x_q$ and $\alpha[1..p-1] = \beta[1..p-1]$, it follows that $p = q$. If $z \in \text{var}(\beta)$, then $z \in \{x_1, x_2, \dots, x_{q-1}\}$. This is due to the fact that, by our assumption from above, the first occurrence of any variable x_j, $j \geq q+1$ is to the right of position q. If, on the other hand, $z \in \Sigma$, then clearly $z = w[q]$.

Consequently, in iteration q of the main loop, either Line 5 or 8 is executed, which means that in α there is no occurrence of variable x_q. Since this is clearly a contradiction, we conclude that α is in fact Π-descriptive of S.

It remains to prove that if, for any pattern β, the question $\beta \in \Pi$ and the membership problem for Π-pattern languages are decidable in polynomial time, then Π-DescPat is a polynomial time algorithm. To this end, note that the

for-loop has m iterations and the while-loop has at most m iterations. Therefore Lines 4 and 7 are executed $\mathcal{O}(m^2)$ times, and for each execution we have to check, for a pattern α_i, whether $\alpha_i \in \Pi$ and $S \subseteq L_\Sigma(\alpha_i)$. Hence, by first checking $\alpha_i \in \Pi$ in polynomial time and then checking $S \subseteq L_\Sigma(\alpha_i)$ only in the case that $\alpha_i \in \Pi$, Lines 4 and 7 can be executed in polynomial time. □

By combining Lemmas 2, 3 and 4, we obtain the following meta-theorem:

Theorem 5. *Let Π be a rich class of patterns. If $\mathcal{P} \neq \mathcal{NP}$, then there exists a polynomial time algorithm that, for a given sample S, computes a pattern that is Π-descriptive of S if and only if the question whether $\alpha \in \Pi$ can be decided in polynomial time and the membership problem for Π-pattern languages can be decided in polynomial time.*

Proof. If the question whether $\alpha \in \Pi$ can be decided in polynomial time and the membership problem for Π-pattern languages can be decided in polynomial time, then, by Lemma 4, Π-DescPat is a polynomial time algorithm that, for a given sample S, computes a pattern that is Π-descriptive of S. This proves the *if* direction of the statement.

In order to prove the *only if* direction, we assume that there exists a polynomial time algorithm that, for a given sample S, computes a pattern that is Π-descriptive of S. We can now conclude with Lemmas 2 and 3 that the question whether $\alpha \in \Pi$ can be decided in polynomial time and the membership problem for Π-pattern languages can be decided in polynomial time, respectively. □

The results presented in this section suggest that for the practical application of descriptive patterns (as a representation of the common structure of a sample or as a tool for inductive inference) it is reasonable to restrict ourselves to rich classes Π of patterns with a polynomial time membership problem. Furthermore, unless the membership problem for Π-pattern languages can be solved in sub-quadratic time, the running time of Π-DescPat is dominated by the running time of an algorithm that solves the membership problem for Π-pattern languages. Hence, a reasonable approach is to find a suitable rich class Π of patterns and then develop efficient algorithms for the membership problem for Π-pattern languages.

Shinohara pursued this approach in [24] with respect to the classes of regular and non-cross patterns, but an optimisation of the membership algorithms is not a focus of his work and, furthermore, the classes of regular and non-cross patterns are strongly restricted classes of patterns. In the next section, we present efficient and sophisticated algorithms for the membership problem of pattern languages with a bounded scope coincidence degree.

4 Algorithms for Computing Descriptive Patterns

The results of the previous section, especially Algorithm 1, show that solving efficiently the membership problem for rich classes of patterns leads to efficient

solutions for the problem of finding, for a sample set, a descriptive pattern in the respective classes. Accordingly, we propose now efficient algorithms for several rich classes of patterns. On one hand we look at classes where the number of repeated variables is bounded: first we consider the basic class of regular patterns, where no variable is repeated, and then investigate the class of patterns in which exactly k variables are repeated. On the other hand we look at classes with bounded scope coincidence degree. A basic class in this case is the one of non-cross patterns, where the upper bound of the scope coincidence degree is 1; we then move on and analyse the general case of patterns having the upper bound of the scope coincidence degree equal to k.

As an interesting feature, the algorithms solving the membership problem for patterns with bounded scope coincidence degree are based on a dynamic programming approach that identifies, for growing values of ℓ, all the prefixes of the words in the sample set that can be the image of a prefix of length ℓ of the input pattern. This approach can be combined to that in Π-DescPat: there, at each iteration of the cycle in Line 2, a prefix of the constructed pattern is already fixed and we try to extend it. Thus, if we know by our dynamic programming approach, from the previous step, which prefixes of the sample words are images of the respective prefix of the pattern, we can try to extend these ones, instead of trying to map the entire constructed pattern anew. Implementing this strategy leads to an improvement in the overall time complexity of the algorithm constructing a descriptive pattern for a sample set of words.

For the algorithms solving the membership problem in the case of patterns with a bounded number of repeated variables, such an approach does not seem to work. These algorithms have a greedy component so it seems hard to do more than just plug them in the parts of Algorithm 1 where we test the membership of the words of the sample set to the pattern languages defined by the constructed patterns. We can only improve the running time of Π-DescPat in the case of regular patterns, where we can safely skip the loop in Line 6.

Assumptions and Preliminary results. The algorithms proposed in this section are implemented on the RAM with logarithmic word size model. Moreover, for the sake of generality, we assume that whenever we are given as input a word w of length n, the symbols of w are in fact integers from $\{1, 2, \ldots, n\}$ (i.e., $\text{symb}(w) \subseteq \{1, \ldots, n\}$), and w is seen as a sequence of integers. This is a common assumption in the area of algorithmics on words (see, e.g., the discussion in [10]). Clearly, our reasoning holds canonically for constant alphabets, as well.

For a length n word w, over $\Sigma \subseteq \{1, 2, \ldots, n\}$, we can build in $\mathcal{O}(n)$ time the suffix tree and suffix array structures, as well as data structures allowing us to retrieve in constant time the length of the longest common prefix of any two suffixes $w[i..n]$ and $w[j..n]$ of w, denoted $LCP_w(i, j)$ (the subscript w is omitted when there is no danger of confusion). Such structures are called LCP data structures in the following. For details, see, e.g., [8, 10], and the references therein. Similarly, we can build structures allowing us to retrieve in constant

time the length of the longest common suffix of any two prefixes $w[1..i]$ and $w[1..j]$ of w, denoted $LCS_w(i,j)$.

As a general result, we will use that $\binom{n}{k} \in \mathcal{O}(\frac{n^k}{k!})$.

Algorithms. We now give the main results of this section. The results of the previous section, especially Algorithm 1, show that solving efficiently the membership problem for rich classes of patterns leads to efficient solutions for the problem of finding, for a sample set, a descriptive pattern in the respective classes. Accordingly, we propose now efficient algorithms for several rich classes of patterns. On one hand we look at classes where the number of repeated variables is bounded: first we consider the basic class of regular patterns, where no variable is repeated, and then investigate the class of patterns in which exactly k variables are repeated. On the other hand we look at classes with bounded scope coincidence degree. A basic class in this case is the one of non-cross patterns, where the upper bound of the scope coincidence degree is 1; we then move on and analyse the general case of patterns having the upper bound of the scope coincidence degree equal to k.

As an interesting feature, the algorithms solving the membership problem for patterns with bounded scope coincidence degree are based on a dynamic programming approach that identifies, for growing values of ℓ, all the prefixes of the words in the sample set that can be the image of a prefix of length ℓ of the input pattern. This approach can be combined to that in Π-DescPat: there, at each iteration of the cycle in Line 2, a prefix of the constructed pattern is already fixed and we try to extend it. Thus, if we know by our dynamic programming approach, from the previous step, which prefixes of the sample words are images of the respective prefix of the pattern, we can try to extend these ones, instead of trying to map the entire constructed pattern anew. Implementing this strategy leads to an improvement in the overall time complexity of the algorithm constructing a descriptive pattern for a sample set of words.

For the algorithms solving the membership problem in the case of patterns with a bounded number of repeated variables, such an approach does not seem to work. These algorithms have a greedy component so it seems hard to do more than just plug them in the parts of Algorithm 1 where we test the membership of the words of the sample set to the pattern languages defined by the constructed patterns. We can only improve the running time of Π-DescPat in the case of regular patterns, where we can safely skip the loop in Line 6.

Theorem 6. *The membership problem for* Pat_{reg}*-pattern languages is solvable in* $\mathcal{O}(n + |\alpha|)$ *time, where* n *is the length of the input word* w *and* α *the input pattern.*

Proof. Let us assume that $\alpha = w_0 \Pi_{i=1,m}(x_i w_i)$, where $x_1, \ldots, x_m$ are variables and $w_0, w_1, \ldots, w_m \in \Sigma^*$, and we want to decide whether a length n word w is in $L_\Sigma(\alpha)$. Using the Knuth-Morris-Pratt algorithm, we can build in $\mathcal{O}(|\alpha|)$ time data structures allowing us to locate the occurrence of all w_i, for $i \geq 0$, in w.

The main remark is that if $w[1..k_s]$ is the shortest prefix of w which can be the image of $w_0\Pi_{i=1,s}x_iw_i$, for every $1 \leq s \leq m$, then $k_s \geq k_{s-1}+1$ and $w[k_s-|w_s|+1..k_s]$ is the leftmost occurrence of w_s in $w[k_{s-1}+1..n]$.

To solve the membership problem for the language described by the pattern α, we first check if w starts with w_0. Then, we start with $s=1$ and $j=|w_0|$. While $s<n$, we read $w[j+2..n]$ from left to right until we find an occurrence of w_s, using the data structures constructed by the KMP algorithm for w_s. If no such occurrence is found, then the word is not in $L(\alpha)$. Otherwise, say that the occurrence of w_s ends on position j'; we update $j=j'$ and increase s by one unit and start looking for w_{s+1} (i.e., iterate the *while* cycle). In this way, we identify the shortest prefix $w[1..j']$ of w that can be an image of $w_0\Pi_{i=1,m-1}(x_iw_i)$. Now we check if $w[j'+2..n]$ has w_n as proper suffix, and, if yes, decide that w is in $L(\alpha)$.

The algorithm is correct and its overall time complexity is $\mathcal{O}(|w|+|\alpha|)$. □

To solve the problem of finding a pattern that is PAT_{reg}-descriptive for a set S of words we just run Π-DescPat using the above procedure to test the membership of a word to the set defined by a regular pattern. Obviously, the cycle in Line 6 of the algorithm is not applicable in the case of regular patterns. Therefore, we obtain the following result.

Theorem 7. *We can compute a pattern that is* PAT_{reg}*-descriptive of a given finite sample S in $\mathcal{O}(nm)$ time, where n is the total length of the words in S, while m is the length of its shortest word.*

We stress that for constant alphabets our results are similar to those in [24]. However, those results use string matching strategies based on finite automata, thus the $\mathcal{O}$-denotation used to express their complexity hides a factor depending on $|\Sigma|$. For our algorithms, this is not the case: the complexities we get for solving the membership problem or computing a descriptive pattern for a sample do not depend at all on $|\Sigma|$.

Lemma 5. *The membership problem for* $\text{PAT}^{\text{r}}_{\text{var}\leq 1}$*-pattern languages is solvable in $\mathcal{O}(n^2)$ time, where n is the length of the input word w.*

Proof. Consider a length n word $w \in \Sigma^*$ and a length m pattern α in which exactly one variable, denoted by x, occurs more than once (if no such variable exists, then the statement of the lemma follows from Theorem 6)

First, if $\alpha = w'x\beta$ with $w' \in \Sigma^*$, then $w \in L_\Sigma(\alpha)$ iff $w \in L_\Sigma(\alpha')$, where α' is a regular pattern obtained from α by substituting x by some factor $w[|w'|+1..|w'|+i]$, $1 \leq i \leq |w|-|w'|$. Hence, according to Theorem 6, $w \in L_\Sigma(\alpha)$ is decidable in time $\mathcal{O}(n^2)$. Obviously, the same holds in case $\alpha = \beta x w'$ with $w' \in \Sigma^*$.

Assume we are no longer in any of these cases, thus $\alpha = \beta x\gamma x\delta$, where β and δ contain at least one variable each and $|\beta|_x + |\delta|_x = 0$. Let $N = |\alpha|_x$. For the rest of the cases we factorise α as $\alpha = w_0\Pi_{i=1,m}(\beta_i w_i \gamma_i w_i')\beta_{m+1}w_{m+1}$, where $w_0 \in \Sigma^*$ and, for $j \geq 1$, β_j starts and ends with a variable and $|\beta_j|_x = 0$, w_j, $w_j' \in \Sigma^*$, and γ_j starts and ends with variable x and $\text{symb}(\gamma_j) = \Sigma \cup \{x\}$. For legibility we write $\alpha_j = w_0\Pi_{i=1,j}(\beta_i w_i \gamma_i w_i')$ for $1 \leq j \leq m$.

It is easy to see that each pattern α that fulfils the conditions in our hypothesis has such a factorisation which can be computed in $\mathcal{O}(|\alpha|)$ time.

We construct for the words w and $w\alpha$ the suffix arrays and LCP and LCS data structures. It is clear that for any constant factor v of α and position i of w we can now test in constant time whether v occurs or ends at position i in w.

Further, we define the $n \times (m+1)$ matrix $M[\cdot][\cdot]$ where $M[i][j] = \ell$ if and only if $w[i..\ell]$ is the shortest factor starting on position i that is in $L(\beta_j)$. We state that $M[\cdot][\cdot]$ can be computed in $\mathcal{O}(nm)$ time. Indeed, for every constant factors u of α we can locate in $\mathcal{O}(n)$ all its occurrences in w. Now, for such a constant factor u we can compute for each position i the value $d_{u,i} = \min\{j \geq i | u \text{ occurs at position } j \text{ in } w\}$. Clearly, this also takes linear time $\mathcal{O}(n)$ for each u. Now, for some $i \leq n$ and $j \leq m+1$, let $u_1, \ldots, u_s$ be the constant factors occurring in β_j. We compute first $g_1 = d_{u_1,i+1}$; then $g_h = d_{u_h,g_{h-1}+|u_{h-1}|+1}$ for $1 < h \leq s$. We set then $M[i][j] = g_s + 1$. The time needed to compute $M[i][j]$ is clearly $\mathcal{O}(|\beta_j|)$. Hence, to compute the elements of the array $M[i][\cdot]$ we need $\mathcal{O}(m)$ time, while the time spent for the whole M is $\mathcal{O}(nm) = \mathcal{O}(n^2)$.

The main idea in our algorithm is the following. If, for $j \leq m$ there exists an assignment for α that maps x to v and α_j to $w[1..p]$, then there exists such an assignment that maps x to v, α_j to $w[1..p]$, and α_{j-1} to the shortest prefix of w that can be the image of α_{j-1} when x maps to v. Therefore, to find the shortest prefix $w[1..p_j]$ of w that is the image of α_j in an assignment mapping x to some v it is enough to find the shortest prefix $w[1..p_{j-1}]$ of w that is the image of α_{j-1} when x is assigned to v, then find $p'_j = M[p_{j-1}+1][j]$ (i.e., identify the shortest prefix of w that is the image of $\alpha_{j-1}\beta_j$), and then find the next occurrence of the image of $w_j\gamma_j w'_j$ when x is assigned to v, and take p_j to be its ending position. Once we found the shortest prefix $w[1..p_m]$ of w that can be the image of α_m in an assignment that maps x to v, it is enough to check whether $M[p_m+1][m+1]$ is defined and if yes, and $M[p_m+1][m+1] = s$ whether $w[s+1..n]$ has w_{m+1} as a suffix. If this final check returns true then there exists an assignment that maps x to v and α to w.

Further, we explain how to implement the above idea efficiently.

For ℓ from 1 to n we check if there is an assignment maping α to w, where x is mapped to a factor of length ℓ. So, let us fix such an ℓ. In linear time, we can partition the suffix array of w in several *clusters* (i.e., blocks of consecutive positions that are not extendable to the left or right) such that the suffixes contained in one cluster are sharing a common prefix of length at least ℓ. It is important to note that no matter to what factor v of length ℓ the variable x is mapped, if the image of each γ_i under this mapping is indeed a factor of w, then they all occur as prefixes of some suffixes contained in the same cluster from the ones defined above, with at least N elements (exactly the cluster where the suffixes share a common prefix starting with v). Moreover, in linear time, by traversing once the suffix array, we order the suffixes in all clusters by their starting position. Now, not only that all the images of the patterns γ_i under the aforementioned mapping occur as prefixes of suffixes contained in the same cluster, but the images of the γ_i occur in the order of their appearance in α within the cluster.

So, once ℓ fixed, we also fix a cluster of at least N suffixes sharing a prefix of length at least ℓ. Now, we try to find an assignment of α to w in which x is mapped to v, the common prefix for the cluster. We start with $j = 1$, $p = 1$ and a token placed on the first element of the cluster. Let $s = M[p][j]$ (i.e., $w[p..s]$ is the shortest prefix of $w[p..n]$ that is an image of β_j); initially, $s = M[|w_0|][1]$, (i.e., $w[1..s]$ is the shortest prefix of w that is the image of $w_0\beta_1$). We now traverse the cluster, left to right, from the position pointed by the token and moving the token accordingly, until we find the first suffix starting after position $s+|w_j|+1$ and preceded by w_j (checked in $\mathcal{O}(1)$ time by an LCS query). Assume we found such a suffix $w[r..n]$. We want to check whether it starts with the image of $\gamma_j w'_j$, where $\gamma_j = \left(\Pi_{i=1,q_j}(xu_{i,j})\right) x$ for some q_j and constants $u_{i,j}$. The maximum h with $(\Pi_{i=1,h}(vu_{i,j}))\, v$ a prefix of $w[r..n]$ is found in time $\mathcal{O}(h)$ by LCP queries.

If $h = q_j$ and $\left(\Pi_{i=1,q_j}(vu_{i,j})\right) v$ is followed by w'_j, then we found the leftmost occurrence of $w_j\gamma_j w'_j$ in w to the right of s; if p' is the last position of this occurrence, then we identified the shortest prefix $w[1..p']$ of w that can be the image of α_j in an assignment that maps x to v. We take now $p = p' + 1$, increase j by one unit and restart the procedure if $j \leq m$.

If $h < q_j$, or $h = q_j$ and w'_j does not follow $(\Pi_{i=1,h}(vu_{i,j}))\, v$, then we keep traversing the cluster, and moving the token we look for a suffix that is preceded by w_i and shares a prefix with $w[r..n]$ at least as long as $(\Pi_{i=1,h}(vu_{i,j}))\, v$. As soon as we found one, say $w[r'..n]$, we find the maximum h' such that $(\Pi_{i=1,h'}(vu_{i,j}))\, v$ is its prefix. Finding h' takes at most $2(h' - h)$ LCP-queries. If $h' = q_j$ and $(\Pi_{i=1,h'}(vu_{i,j}))\, v$ is followed by w'_j, then we proceed as above; otherwise, we continue the traversal of the cluster with $w[r'..n]$ in the role of $w[r..n]$ and h' in the role of h. If we cannot find the prefix $w[1..p']$ of w that is the image of α_j in an assignment that maps x to v within the current cluster, then the assignment of x was wrong, and we try another cluster defined for the same length, or even another length.

If finally we have $j = m + 1$, then we found the shortest prefix $w[1..p_m]$ of w that is an image of α_m while mapping x to v, and we can proceed as previously described to decide whether there exists an assignment that maps α to w. The processing of a cluster clearly takes $\mathcal{O}(k + N) = \mathcal{O}(k)$, where $k \geq N$ is its size.

We repeat the above procedure for all clusters of size at least N. As the total number of elements in these clusters is upper bounded by $n - \ell$ for the fixed ℓ, it is clear that their total processing takes $\mathcal{O}(n)$ time.

Further, we have to consider all possible values for ℓ. The time spent for each of them is $\mathcal{O}(n)$, so, in total our algorithm needs $\mathcal{O}(n^2)$ time to decide whether there exists an assignment of the variables of α that maps this pattern to w. The correctness of our algorithm follows from the explanations given above. □

We use now the above lemma to solve the membership problem when we deal with patterns that have at most k variables that appear more than once.

Theorem 8. *The membership problem for* $\text{PAT}^{\text{r}}_{\text{var}\leq k}$*-pattern languages is solvable in* $\mathcal{O}\left(\frac{n^{2k}}{((k-1)!)^2}\right)$ *time, where* n *is the length of the input word* w*.*

Proof. Let x be the leftmost repeating variables and $\{x_1, x_2, \ldots, x_{k-1}\}$ the other repeating variables of α, indexed in the order they occur. For a fixed length ℓ of the image of x, we assign to each variable x_i a factor of w. This means choosing $2k-2$ numbers $\{i_1, i_2, \ldots, i_{2k-2}\}$ with $\ell < i_{2h-1} \leq i_{2h} < i_{2h+1}$, for $1 \leq h \leq k-1$ ($i_{2k-1} = n+1$ by convention); here, $w[i_{2h-1}..i_{2h}]$ is the image of x_h. There are less than $\frac{n^{2k-2}}{((k-1)!)^2}$ ways of choosing these numbers. For each we produce a pattern α' where just x is repeated. We know the length of the image of x, and as in the proof of Lemma 5 we check in $\mathcal{O}(n)$ time whether there is an assignment for x to a length ℓ word that maps α' to w. We iterate this process for all ℓ.

Clearly, this decides the existence of an assignment mapping α to w. The running time is bounded by $\mathcal{O}\left(\sum_{\ell=1,n}\left(n\frac{n^{2k-2}}{((k-1)!)^2}\right)\right) = \mathcal{O}\left(\frac{n^{2k}}{((k-1)!)^2}\right)$. □

Unfortunately, we were unable to do more than plug in this algorithm into Π-DescPat so as to find a descriptive pattern with at most k repeated variables for a sample.

Theorem 9. *There exists an algorithm that computes a pattern that is* $\text{Pat}^{\mathsf{r}}_{\text{var}\leq k}$*-descriptive of a given sample S in $\mathcal{O}\left(\frac{n^{2k}m^2}{((k-1)!)^2}\right)$ time, where n is the total length of the words in S and m is the length of its shortest word.*

We now consider the case of patterns that have a bounded scope coincidence degree. The following combinatorial results are well known (see, e. g., [3])

Lemma 6. *Let u_1, u_2, u_3 be primitive words, such that $|u_1| < |u_2| < |u_3|$ and u_i^2 are suffixes of a word v, for all $1 \leq i \leq 3$. Then $2|u_1| < |u_3|$.*

By the previous lemma, the number of primitively rooted squares occurring as suffixes of a given word can be bounded.

Lemma 7. *For a word v with $|v| = n$, we have that*

$$|\{u | u \text{ primitive }, u^2 \text{ is a suffix of } v\}| \leq 2\log n.$$

Identical results can be derived for the primitively rooted squares occurring as prefixes of a given word. Assume, in the following, that $w \in \Sigma^*$ is a word of length n. For each $i \leq n$ we define the set P_i as follows:

$$P_i = \{|u| \mid u \text{ is a primitive word such that } u^2 \text{ is a suffix of } w[1..i]\}.$$

Lemma 7 shows that $|P_i| \leq 2\log n$ for all $1 \leq i \leq n$.

Lemma 8. *Let $w \in \Sigma^*$ be a word of length n. We can compute in $\mathcal{O}(n \log n)$ time all the sets P_i with $i \in \{1, \ldots, n\}$.*

The following observation is straightforward.

Remark 1. For a suffix $w[i..j]$ of $w[1..j]$ the maximum integer ℓ such that $w[i..j]^\ell$ is also a suffix of $w[1..j]$ is $\ell = \left\lfloor \frac{LCS_w(j,i-1)}{j-i+1} \right\rfloor + 1$.

Assume that the sets P_i are stored with the elements ordered according to their lengths (this is implemented at construction time). Accordingly, each P_i is an array where $P_i[k]$, $k \leq 2\log n$, stores the length of the k^{th} element of the ordered set P_i. For simplicity, when the set of primitively rooted squares ending at position i in w has $\ell < 2\log n$ elements, only the first ℓ elements of P_i are defined.

Lemma 9. *If $P_i[k] = \ell$ and $w[i-\ell+1..i]^2$ is a suffix of $w[1..j]$, then $P_j[k] = \ell$.*

Proof. A word x^2 with x primitive and $|x| < \ell$ is a suffix of $w[1..i]$ if and only if it is a suffix of $w[i-\ell+1..i]^2$ if and only if it is a suffix of $w[1..j]$. □

In the following, the number of one-variable blocks in a pattern is the number of contiguous blocks of occurrences of the same variable. For instance, the number of one-variable blocks in $x_1x_2^2x_3ax_2^2x_3^2$ is 5. Clearly, if the number of one-variable blocks in a pattern α is m, then $\alpha = w_0\Pi_{i=1,m}(x_i^{k_i}w_i)$.

Theorem 10. *The membership problem for* PAT_{snc}*-pattern languages is solvable in $\mathcal{O}(nm\log n)$ time, where n is the length of the input word w and m is the number of one-variable blocks occurring in the pattern.*

Proof. For a strictly non-cross pattern $\alpha = w_0\Pi_{i=1,m}(x_i^{k_i}w_i)$ (where x_i's are variables and w_i's are (possibly empty) strings of constants), we want to decide whether $w \in L(\alpha)$. For $1 \leq j \leq m$, given an assignment $\{v_1, v_2, \ldots, v_j\}$ for the variables $\{x_1, x_2, \ldots, x_j\}$ of α, we denote $\alpha_j = w_0\Pi_{i=1,j}(v_i^{k_i}w_i)$.

We start by defining LCP data structures for the word $w\alpha$. Also, we construct the sets P_i, as in Lemma 8. Assume that the sets P_i are stored with the elements ordered according to their lengths (this is implemented at construction time). Accordingly, each P_i is an array where $P_i[k]$, $k \leq 2\log n$, stores the length of the k^{th} element of the ordered set P_i. Note that, for the simplicity of the exposure, we will say that a prefix t of $w[i..n]$ is in P_i if, in fact, $|t| \in P_i$. Obviously, these two ways are equivalent, but our implementation of the sets P_i is more efficient, as we just store in them numbers that can be represented in one memory word (lengths), instead of actual strings. Also, when the set of primitively rooted squares ending at position i in w has $\ell < 2\log n$ elements, only the first ℓ elements of P_i are defined.

Our solution is based on the following claims.

Claim 1. For $k_p > 1$, $w[1..i] = \alpha_{p-1}v_p^{k_p}$ with $1 \leq i \leq p$ if and only if there exists $t \in P_i$ such that one of the following holds:

1. t^{k_p} is a suffix of $w[1..i]$ and, for $j = i - |t|k_p$ (so, $w[j+1..i] = t^{k_p}$), we have $w[1..j] = \alpha_{p-1}$; in this case $v_p = t$.
2. t^{2k_p} is a suffix of $w[1..i]$ and, for $j = i - |t|k_p$ (so, $w[j+1..i] = t^{k_p}$), we have $w[1..j] = \alpha_{p-1}(t^s)^{k_p}$ for some $s \geq 1$; in this case $v_p = t^{s+1}$.

Proof (Claim 1). The *if* part of the statement is immediate. So, let us assume that $w[1..i] = \alpha_{p-1}v_p^{k_p}$ with $k_p > 1$. Clearly, v_p^2 is a suffix of $w[1..i]$. If v_p is primitive then $v_p = t \in P_i$. Now, t^{k_p} is a suffix of $w[1..i]$ and, for $j = i - |t|k_p$, we

have $w[1..j] = \alpha_{p-1}$. When v_p is not primitive, we have $v_p = t^{s+1}$ for some $s \geq 1$ and $t \in P_i$. Clearly, for $j = i - |t|k_p$, we have $w[1..j] = \alpha_{p-1}(t^s)^{k_p}$.(Claim 1) □

Claim 2. $w[1..i] = \alpha_{p-1}v_p$ with $1 \leq i \leq p$ and $v_i \in \Sigma^+$ if and only if one of the following holds:

1. $w[1..i-1] = \alpha_{p-1}$; in this case $v_p = w[i]$.
2. $w[1..i-1] = \alpha_{p-1}v'_p$; in this case $v_p = v'_p w[i]$.

Proof (Claim 2). This claim follows immediately. (Claim 2) □

The two claims suggest a dynamic programming approach for solving the membership problem for $\mathrm{Pat}_{\mathrm{snc}}$. In particular, we define the array $M[\cdot][\cdot][\cdot]$, where $M[i][p][\ell] = 1$ for some $1 \leq i \leq n$, $1 \leq p \leq m$, and $1 \leq \ell \leq 2\log n$ if and only if $k_p > 1$, $P_i[\ell] = |t|$, and $w[1..i] = \alpha_{p-1}v_p^{k_p}$ where v_p is a power of t. Also, we define the array $M'[\cdot][\cdot]$, where $M'[i][p] = 1$ for $1 \leq i \leq n$ and $1 \leq p \leq m$ if and only if $k_p > 1$ and there exists ℓ such that $M[i][p][\ell] = 1$ or $k_p = 1$ and $w[1..i] = \alpha_{p-1}v_p$ with $v_p \neq \varepsilon$.

The values stored in M and M' can be computed inductively. In particular, Claims 1 and 2 show the following strategy of computing $M[i][p][\ell]$ and $M'[i][p]$.

For $p = 1$, if $k_1 > 1$, for $i \leq n$ and $\ell \leq 2\log n$, we set $M[i][1][\ell] = 1$ if and only if $P_i[\ell]$ is defined and $w[1..i] = w_0(w[i - P_i[\ell] + 1..i])^s$ where s is divisible by k_1. Thus, $M[i][1][\ell]$ can be computed in constant time: we just check whether $w[1..|w_0|] = w_0$ and $w[|w_0| + 1..i] = (w[i - P_i[\ell] + 1..i])^s$; the first check is done using an LCP query while the latter as in Remark 1. If at least one of the values $M[i][1][\ell] = 1$ or if $k_1 = 1$ and $w[1..|w_0|] = w_0$, then we also set $M'[i][1] = 1$.

Assume now that $p > 1$ and we have computed $M[\cdot][p-1][\cdot]$ and $M'[\cdot][p-1]$.

Further, if $k_p > 1$, we compute $M[i][p][\cdot]$ for all i from 1 to n, in increasing order, as suggested by Claim 1. Fix a value i, and assume we want to compute $M[i][p][\ell]$, for $\ell \leq 2\log n$; according to our strategy, we already computed all the values $M[j][p][\cdot]$, $M'[j][p]$, and $M'[j][p-1]$, for $j < i$. First, we check whether $P_i[\ell]$ is defined. If so, we let $t = w[i-P_i[\ell]+1..i]$. Then, we compute the maximum s such that t^s occurs as a suffix of $w[1..i]$, as in Remark 1. If $s \geq k_p$ we check, for $j = i-|t|k_p$ and $j' = j-|w_{p-1}|$, whether $w[j'+1..j] = w_{p-1}$ and $M'[j'][p-1] = 1$; that is, whether $w[1..j]$ can be written as α_{p-1}. If yes, we set $M[i][p][\ell] = 1$. If the check above fails and, moreover, $s \geq 2k_p$ we check, for the same j, whether $M[j][p][\ell] = 1$ (this was computed, as $j < i$). Since by Lemma 9, in our case, $P_j[\ell] = P_i[\ell]$ and $w[i - P_i[\ell] + 1..i] = w[j - P_i[\ell] + 1..j]$, this latest check verifies whether $w[1..j] = \alpha_{p-1}v'^{k_p}_p$ where v'_p is a power of t. If the check is positive, we set $M[i][p][\ell] = 1$. In all the other cases, following Claim 1, we set $M[i][p][\ell] = 0$. Also, when $k_p = 1$, we set $M[i][p][\ell] = 0$ for all i and ℓ.

If $k_p > 1$ and at least one of $M[i][p][\ell]$ was set to 1, then we also set $M'[i][p] = 1$. If $k_p = 1$, then $M[i][p][\ell] = 0$ for all i and ℓ, and by Claim 2 we set $M'[i][p] = 1$ if and only if for some non-empty words $v_1, \ldots, v_{p-1}$ one of the following holds:

- $M'[i-1][p] = 1$; that is, $w[1..i-1] = \alpha_{p-1}v'_p$ with $|v'_p| > 0$;
- $M'[i - |w_{p-1}| - 1][p-1] = 1$; that is, $w[1..i-1] = \alpha_{p-1}$.

Clearly, this inductive approach towards computing M and M' guarantees that when we compute $M[i][p][\ell]$ and $M'[i][p]$, we already computed $M[j][p][\cdot]$ and $M'[j][p]$, for $j < i$, as well as the values in $M[\cdot][p-1][\cdot]$ and in $M'[\cdot][p-1]$. Hence, computing $M[i][p][\ell]$ and $M'[i][p]$ requires only a constant number of checks, each requiring constant time. Therefore, using dynamic programming, we have a method to compute each of the elements of M and M' in constant time. Moreover, as $1 \leq i \leq n$, $1 \leq p \leq m$, and $1 \leq \ell \leq 2 \log n$, the time needed to compute all the elements of M and M' is $\mathcal{O}(nm \log n + nm) = \mathcal{O}(nm \log n)$.

Once these arrays computed, we decide that the instance of the membership problem defined by the input word w and the input pattern α can be answered positively if and only if w has the suffix w_m and $M'[n - |w_m|][m] = 1$. □

In the case of general non-cross patterns, we can solve the membership problem by a similar approach, however, with a slightly increased time complexity.

Theorem 11. *The membership problem for* PAT_{nc}*-pattern languages is solvable in* $\mathcal{O}(n^2 m)$ *time, where* n *is the length of the input word* w *and* m *is the number of one-variable blocks occurring in the pattern.*

Proof. Assume that $\alpha = w_0 \Pi_{i=1,m}(y_i^{k_i} w_i)$ is the non-cross pattern. Hence, for each $i > 1$, y_i may be equal to y_{i-1}, but if $y_i \neq y_{i-1}$ then $y_i \neq y_j$ for all $j < i$. We again define *LCP* data structures for the word $w\alpha$.

Following the same general ideas as in the previous proof, we define the $n \times m \times n$ matrix $M[\cdot][\cdot][\cdot]$ as follows: $M[i][p][\ell] = 1$ if $w[1..i] = w_0 \Pi_{i=1,p-1}(v_i^{k_i} w_i) v_p^{k_p}$ for some non-empty words $v_1, \ldots, v_p$ with $|v_p| = \ell$. We also define a matrix $M'[\cdot][\cdot]$ such that $M'[i][p] = 1$ if and only if there exists ℓ such that $M[i][p][\ell] = 1$. Again, the elements of M and M' can be computed by dynamic programming, considering the possible values of p in increasing order.

For $p = 1$, we set $M[i][1][\ell] = 1$ if and only if w_0 is a prefix of w and $i = |w_0| + \ell$; the elements of $M'[\cdot][1]$ are computed accordingly. Then, for $p > 1$, if $y_p \neq y_{p-1}$, we set $M[i][p][\ell] = 1$ if $w_{p-1}(w[i-\ell+1..i])^{k_p}$ is a suffix of $w[1..i]$ (tested with Remark 1 and *LCP* queries) and $M'[i'][p-1] = 1$ for $i' = i - (k_p \ell + |w_{p-1}|)$. If $y_p = y_{p-1}$, we set $M[i][p][\ell] = 1$ if $w_{p-1}(w[i - \ell + 1..i])^{k_p}$ is a suffix of $w[1..i]$ (tested with Remark 1 and *LCP* queries) and for $i' = i - (k_p \ell + |w_{p-1}|)$ we have $M[i'][p-1][\ell] = 1$ and $w[i' - \ell + 1..i'] = w[i - \ell + 1..i]$. Once the values of $M[i][p][\cdot]$ are computed, we can set the value of $M'[i][p]$.

This computation takes $\mathcal{O}(n^2 m)$ time. Once these arrays computed, we decide that the instance of the membership problem defined by the input w and α is answered positively if and only if w has the suffix w_m and $M'[n - |w_m|][m] = 1$. □

As a consequence of the previous theorem we can show the following result:

Theorem 12. *We can compute a pattern that is* PAT_{nc}*-descriptive (*PAT_{snc}*-descriptive) of a given sample* S *in* $\mathcal{O}(n^2 m)$ *time (*$\mathcal{O}(nm \log n)$ *time), where* n *is the total length of the words in* S*, while* m *is the length of its shortest word.*

Proof. Intuitively, when solving the membership problems for PAT_{nc} or PAT_{snc} patterns, we can use a more tedious approach than in the proofs of Theorems 10 and 11 and match, one by one, in increasing order with respect to the length, a prefix of the pattern to all possible prefixes of the input word, then read another letter of the pattern and use the previously computed information to obtain the matches of the new extended prefix. The essential difference in the case of computing a descriptive pattern for a sample is that, now, we do not know the pattern. However, when using Π-DescPat to construct descriptive patterns, at the i^{th} iteration of the cycle in Line 2, we know a prefix of length $i-1$ of the pattern. So, we could use the solutions for the membership problem to find all its matches to prefixes of the words in the sample. Then we guess the new letter of the prefix, instead of reading it from the input. For each guess, we try to get valid matches. When a guess allows us matching a new prefix of the pattern to valid prefixes of the input words, the guess becomes permanent part of the pattern. In the end, we have a descriptive pattern of the sample.

Let us assume that Π-DescPat computes the descriptive pattern α for the given sample in each case and w be the shortest word of S; obviously $|\alpha| = |w|$.

We start by explaining the case of PAT_{nc}, as the other case is similar. As explained, our idea is to incorporate the computation of (some variants of) the matrices M and M' into the main cycle of the Π-DescPat.

First we modify slightly the constructions of the matrices M and M' from the proof of the previous theorem. For each $v \in S$, with $|v| = n_v$, we define the $n_v \times m \times n_v$ matrices $M_v[\cdot][\cdot][\cdot]$ as follows: $M_v[i][p][\ell] = 1$ if $\alpha[1..p]$ can be mapped to $v[1..i]$ and ℓ is either $|v_x u|$ where xu is the suffix of $\alpha[1..p]$ such that x is a variable that was mapped to v_x and u a constant factor, or p if $\alpha[1..p]$ is constant. We also define the $n_v \times m$ matrix $M'_v[\cdot][\cdot]$ such that $M'_v[i][p] = 1$ if and only if there exists ℓ such that $M_v[i][p][\ell] = 1$. Just as before, when α is a fixed pattern, M_v and M'_v can be computed by dynamic programming, as follows.

For $p = 1$, if $\alpha[1] = v[1]$, we set $M_v[1][1][1] = 1$ and $M_v[i][1][\ell] = 0$ for all other i and ℓ; otherwise, if $\alpha[1]$ is a variable, we set $M_v[i][1][i] = 1$ for all i and $M_v[i][1][\ell] = 0$ for all other i and ℓ. The elements of $M'[\cdot][1]$ are computed accordingly. Then, for $p > 1$, if $\alpha[p] = x$ and x did not appear before, we set $M_v[i][p][\ell] = 1$ if $M'_v[i-\ell][p-1] = 1$. If x appeared before, then $\alpha[1..p] = u'xux$ with u constant and we set $M_v[i][p][\ell] = 1$ if for $i' = i - \ell$ we have $M_v[i'][p-1][\ell + |u|] = 1$ and $v[i' - (\ell + |u|) + 1..i' - |u|] = v[i - \ell + 1..i]$. Once the values of $M_v[i][p][\cdot]$ are computed, we can set the value of $M'_v[i][p]$.

For some $v \in S$, and fixed α, M_v and M'_v can be computed in $\mathcal{O}(n_v^2 m)$ time, so for all M_v and M'_v with $v \in S$ we need $\mathcal{O}(n^2 m)$ time, where $m = |\alpha|$. In the following we explain how to proceed when α is constructed by Π-DescPat.

Since when the *for* cycle of Π-DescPat is executed for $i = p$, $\alpha[1..p-1]$ was already found, for each $v \in S$, we already know all the values $M_v[i][p-1][\cdot]$ and $M'_v[i][p-1]$ for $v \in S$. Assume now that we execute the cycle for $i = p$, thus we have the values $M_v[\cdot][p-1][\cdot]$ and $M'_v[\cdot][p-1]$ for all $v \in S$, as well as $\alpha[1..p-1]$; clearly, this holds for $p = 1$. We first check if $\alpha[1..p-1]w[p]$ is a prefix of α. For this, we compute $M_v[i][p][\cdot]$ and $M'_v[i][p]$ under the assumption

that $\alpha[p]$ is $w[p]$. If we find for each v a value i_v such that $M'_v[i_v][p] = 1$ and $i_v \leq n_v - m + p$, then we conclude that $\alpha[p]$ should be $w[p]$ (as all $v \in S$ are in $L(\alpha[1..p-1]w[p]x_{p+1} \dots x_m)$, and the constructed pattern is non-cross), and execute the cycle for $i = p + 1$. Otherwise, we take $\alpha[p]$ to be the last variable occurring in $\alpha[1..p-1]$, say x, and try to compute $M_v[i][p][\cdot]$ and $M'_v[i][p]$ under the assumption that $\alpha[p]$ is indeed x. If we find for each v a value i_v such that $M'_v[i_v][p] = 1$ and $i_v \leq n_v - m + p$, then we conclude again that $\alpha[p]$ should indeed be x (as all $v \in S$ are in $L(\alpha[1..p-1]xx_{p+1} \dots x_m)$, and the constructed pattern is non-cross), and execute the cycle for $i = p + 1$. If this is not the case, then we take $\alpha[p] = x_p$, compute $M_v[i][p][\cdot]$ and $M'_v[i][p]$ under this assumption, and execute the cycle for $i = p + 1$. Clearly, when we move to the execution for $i = p + 1$ our initial assumption holds: we have all the values $M_v[\cdot][p][\cdot]$ and $M'_v[\cdot][p]$ for all $v \in S$, as well as $\alpha[1..p]$. In the end, after the *for* cycle was executed m times, we obtain a pattern α that is descriptive for the sample S.

In all cases, one iteration of the cycle in Line 2 for $i = p$ takes $\mathcal{O}(n^2)$ time (more precisely, $\mathcal{O}(\sum_{v \in S} n_v^2)$): we just compute all the values $M_v[\cdot][p]$ for three possible choices of $\alpha[p]$, and keep one of the variants. So the overall running time of the algorithm is $\mathcal{O}(n^2 m)$.

We discuss now the case of strictly non-cross patterns. We define data structures allowing us to answer *LCP* queries for the word $w\alpha$, and we construct the sets $P_{v,i}$, that correspond for each v to the sets P_i from Lemma 8, under the same conventions as in the proof of Theorem 10.

In this case, for each $v \in S$, with $|v| = n_v$, we define the array $M_v[\cdot][\cdot][\cdot]$, where $M_v[i][p][\ell] = 1$ for some $1 \leq i \leq n_v$, $1 \leq p \leq m$, and $1 \leq \ell \leq 2 \log n_v$ if and only if $\alpha[1..p]$ can be mapped to $v[1..i]$, $\alpha[1..p]$ ends with $x_p^{k_p}$ with $k_p > 1$, $P_{v,i}[\ell]$ encodes the suffix t of $v[1..i]$, and when mapping $\alpha[1..p]$ to $v[1..i]$ the variable x_p is mapped to a power of t. Also, we define the array $M'_v[\cdot][\cdot]$. We have $M'_v[i][p] = 1$ for some $1 \leq i \leq n_v$ and $1 \leq p \leq m$ if and only if one of the following conditions holds:

- there exists ℓ such that $M_v[i][p][\ell] = 1$, or
- $\alpha[1..p]$ ends with $v[i]$ and $\alpha[1..p-1]$ can be mapped to $v[1..i-1]$, or
- $\alpha[1..p]$ ends with a variable x that occurs first in α on position p and either $\alpha[1..p]$ can be mapped to $v[1..i-1]$ or $\alpha[1..p-1]$ can be mapped to $v[1..i-1]$.

Just like in the case of $\textsc{Pat}_{\text{nc}}$ patterns, the arrays $M_v[\cdot][p][\cdot]$ and $M'_v[\cdot][p]$ are computed in the iteration $i = p$ of the *for* loop of Π-DescPat. We now show how to compute these arrays in $\mathcal{O}(n_v \log n_v)$ time.

Let us assume that the loop in Line 2 of Π-DescPat is executed for $i = p$. Thus, for $p' < p$, we have all the values $M_v[i][p'][\cdot]$ and $M'_v[i][p']$ for all $v \in S$, as well as $\alpha[1..p-1]$; clearly, this holds for $p = 1$. In the execution of the cycle, we first try to see if $\alpha[1..p-1]w[p]$ is a prefix of α. For this, we try to compute $M_v[i][p][\cdot]$ and $M'_v[i][p]$ under the assumption that $\alpha[p]$ is indeed $w[p]$. Clearly, $M_v[i][p][\ell] = 0$, for all i and ℓ. On the other hand $M'_v[i][p] = 1$ if and only if $M'_v[i-1][p-1] = 1$ and $v[i] = w[p]$. If we can find for each v a value i_v such that $M'_v[i_v][p] = 1$ and $i_v \leq n_v - m + p$, then we conclude that $\alpha[p]$ should

indeed be $w[p]$ (as, we get that all $v \in S$ are in $L(\alpha[1..p-1]w[p]x_{p+1}\dots x_m)$, and the constructed pattern is clearly strictly non-cross), and move to the execution of the cycle for $i = p+1$. The entire analysis takes $\mathcal{O}(n_v)$ time in this case. Otherwise, we just take $\alpha[p]$ to be the last variable occurring in $\alpha[1..p-1]$, say $x = \alpha[p-1]$ and we try to compute $M_v[i][p][\cdot]$ and $M'_v[i][p]$ under the assumption that $\alpha[p]$ is indeed x. Take x^{k_p} to be the maximum power of x that is a suffix of $\alpha[1..p]$. Now, for each $i \leq n_v$ (considered in increasing order) we do the following processing. We take each ℓ such that $P_{v,i}[\ell]$ is defined, and let $i' = i - k_p P_{v,i}[\ell]$ and $t = v[i - P_{v,i}[\ell] + 1..i]$. Now, just like in Theorem 10, it is clear that $M_v[i][p][\ell] = 1$ if and only if $M'_v[i'][p-k_p] = 1$ or $M_v[i'][p][\ell] = 1$ and $v[i' - P_{v,i'}[\ell] + 1..i'] = v[i - P_{v,i}[\ell] + 1..i]$. Now, $M'_v[i][p]$ is set to 1 if there exists ℓ such that $M_v[i][p][\ell] = 1$. If we can find for each v a value i_v such that $M'_v[i_v][p] = 1$ and $i_v \leq n_v - m + p$, then we conclude again that $\alpha[p]$ should indeed be x (as, we get that all $v \in S$ are in $L(\alpha[1..p-1]xx_{p+1}\dots x_m)$, and the constructed pattern is clearly strictly non-cross), and move to the execution of the cycle for $i = p+1$. The total analysis in this case takes $\mathcal{O}(n_v \log n_v)$ time. If this case cannot be successfully completed, then we just take $\alpha[p] = x_p$ and compute $M_v[i][p][\cdot]$ and $M'_v[i][p]$ under this assumption, and move to the execution of the cycle for $i = p+1$. In this case, we only have to set $M'_v[i][p] = 1$ if and only if $M'_v[i-1][p] = 1$ or $M'_v[i-1][p-1] = 1$; this requires $\mathcal{O}(|v|)$ time. Clearly, whenever we move on to execute the loop for $i = p+1$ our initial assumption holds: we have all the values $M_v[i][p][\cdot]$ and $M'_v[i][p]$ for all $v \in S$, as well as $\alpha[1..p]$. When we finish executing the *for* cycle, we obtained a descriptive pattern for S.

In all cases, one iteration of the *for* loop takes $\mathcal{O}(n \log n)$ time (more precisely, $\mathcal{O}(\sum_{v \in S} n_v \log n_v)$. So the overall running time of the algorithm is $\mathcal{O}(mn \log n)$.

The correctness of the algorithm is straightforward, as it is, just a reorganisation of Π-DescPat with the membership queries executed more efficiently. □

We now move on to the general case of patterns with bounded scope coincidence degree. Generally, for a pattern α, we say that the variable x is active at position p of α if x occurs at least once both in $\alpha[1..p]$ and in $\alpha[p+1..|\alpha|]$. In a pattern α, with $\mathrm{scd}(\alpha) = k$, there are at most k active variables at each position.

Theorem 13. *The membership problem for* $\mathrm{PAT}_{\mathrm{scd} \leq k}$ *is solvable in* $\mathcal{O}\left(\frac{n^{2k}m}{((k-1)!)^2}\right)$ *time, where n is the length of the input word w and m is the number of one-variable blocks occurring in the pattern.*

Proof. This proof follows the same lines of the proof of Theorem 11. Let $\alpha = w_0 \Pi_{i=1,m}(y_i^{k_i} w_i)$ (where y_i's are variables and w_i's are strings of constants) be a pattern with $\mathrm{scd}(\alpha) \leq k$; we want to decide whether $w \in L_\Sigma(\alpha)$. For $1 \leq j \leq m$, we denote $\alpha_j = w_0 \Pi_{i=1,j}(y_i^{k_i} w_i)$ and $\ell_j = |\alpha_j|$. For all $j \leq m$, we produce a list of the active variables at position ℓ_j; this takes $\mathcal{O}(|\alpha|k)$ time. We also build LCP-data structures for $w\alpha$.

This time, we define the $n \times m$ matrix $M[\cdot][\cdot]$ such that $M[i][j]$ contains a representation of all the possible assignments of the active variables at position ℓ_j in

assignments mapping α_j to $w[1..i]$. Before stating how this matrix is computed, it is important to explain our representation.

First, let us note that if at most $k-1$ variables are active at position ℓ_j, then we will store them by the starting and ending positions of their images, in the order of their occurrence in the pattern; in this way, the positions we store are also ordered, as the images of the variables will also occur in the same order as the variables. In this way, we need to store a list of $2k-2$ ordered indices less than i, so we may have to store in $M[i][j]$ at most $\binom{i}{k-1}^2$ different lists.

If k variables are active at position ℓ_j then one of the active variables is y_j. Then, for y_j we do not need to store the ending position. Once we know the starting position of the block $y_j^{k_j}$, say i', we can get its ending position by noting that the image of α_j is $w[1..i]$ and this image ends with the image of $y_j^{k_j} w_j$. So, the image of y_j occurs between i' and $i' + \frac{i-|w_j|-i'+1}{k_j} - 1$. Therefore, when exactly k variables are active at position ℓ_j we only need to store $2k-1$ indices: the starting and ending positions of all the active variables except y_j, in order of their occurrence, and the starting position of $y_j^{k_j}$. This means that the $M[i][j]$ stores at most $\binom{i}{k-1}\binom{i}{k}$ distinct lists of $2k-1$ indices (see below why).

Now, we address the question of how to represent efficiently such collections of lists. Assume that we want to store a collections of lists, each containing p indices $i_1, \ldots, i_p$ between 1 and i, such that $i_{2h-1} \leq i_{2h}$, for $1 \leq h \leq \frac{p}{2}$, and $i_{2h} < i_{2h+1}$, for $1 \leq h \leq \frac{p-1}{2}$. If $p = 2k-1$ then there less than $\binom{i}{k}\binom{i}{k-1}$ such lists; if $p = 2k$ then there less than $\binom{i}{k}^2$ such lists. This holds because an upper bound on the number of such lists is obtained by considering all the possibilities of choosing the numbers on the even positions and the numbers on the odd positions independently. We construct a tree (called (i,p)-tree in the following) with $p+1$ levels $0, 1, \ldots, p$. On the level 0 we have the root labeled with 0. Then, we take all the possible lists of p elements that fulfil the above restriction, in the natural order on $\mathbb{N}^p$, and insert them, one by one, in such a tree as paths starting from the root (so, 0 and then the actual elements of the list), keeping the children of each node ordered. The time needed to construct this tree is upper bounded by $\mathcal{O}(p\binom{i}{k}^2)$ if $p = 2k$ or by $\mathcal{O}(p\binom{i}{k}\binom{i}{k-1})$ if $p = 2k-1$.

Clearly, to store a collection I of lists containing p indices between 1 and i we can use such an (i,p)-tree and we just mark in it the leaf corresponding to each list in I (so, we will have an (i,p)-tree with some marked leaves instead of the collections I). With this representation, we can test whether a list is in the collection in $\mathcal{O}(p)$ time, we can insert or delete a list in $\mathcal{O}(p)$ time, and, by keeping a linked list of the leaves, we can traverse the marked leaves in $\mathcal{O}(\binom{i}{p}^2)$ time. Given a leaf, we can retrieve the list that defines it by following the path from that leaf to the root, in $\mathcal{O}(p)$ time (provided that we store also child to father links in our tree). For simplicity, the root of a tree is said to be also marked if at least one of the leaves is marked.

So, coming back to our matrix, each $M[i][j]$ is initialised as an empty (i,s)-tree, where $s = 2p$ when there are $p \leq k-1$ active variables at position ℓ_j in α

or $s = 2k - 1$ when there are exactly k active variables at position ℓ_j in α. We just have to explain how $M[i][j]$ is computed efficiently.

First, $M[i][1]$ is obtained in $\mathcal{O}(n^3)$ as follows. For each i, we try all possibilities of choosing the image of y_1 as a factor of $w[|w_0|..i]$. Each of them is saved in the corresponding $(i, 2)$ tree. The leaves and the root are marked accordingly.

Now, we move on to computing $M[i][j]$, assuming that we already computed $M[\cdot][j-1]$. Basically, for $i' \leq n$, looking at $M[i'][j-1]$ we retrieve the assignments of the variables that are active at position ℓ_{j-1}. Now, if y_j is one of them, we just have to check whether the image of $(y_j)^{k_j} w_j$ occurs at position $i' + 1$ (which is done in constant time by LCP-queries). If yes, and there are k active variables, then the positions storing the image of the last variable in our lists may have changed; hence, we update the list in $\mathcal{O}(k)$ (that is, delete the starting and, unless $y_j = y_{j-1}$, ending position that denoted the position of y_j, and then append to the list the starting and ending positions of y_{j-1} as well as i' as the starting position of $(y_j)^{k_j} w_j$, as explained before). If the list contained less than k variables, we just leave it as it is. In both cases, the new list is inserted in the tree $M[i][j]$, where i is the ending position of the image of $(x_j)^{k_j} w_j$. We just have to deal with the case when y_j was not one of the active variables. In this case, it is a new variable, that becomes active. As a first step, we assume that the image of y_j is $w[i'+1]$ and save this in a tree $C[i'+1][j]$ (these trees are auxiliary, and are empty before $M[\cdot][j]$ are computed); basically, at this step we did not check if w_j can follow the image of y_j nor did we find all possible assignments for y_j. After we finish this process for all values of i', we continue. Now, for $i'' < n$, if $C[i''][j]$ is non-empty, then we insert all of its assignments into $C[i''+1][j]$, just saving (if there are less than k active variables at position ℓ_j) a new ending position for y_j as $i'' + 1$. Basically, at this point $C[i''][j]$ contains all ways of assigning values to the variables active at position ℓ_j such that $\alpha_{j-1} y_j$ is mapped to $w[1..i'']$. Then, for each $i'' \leq n$ and each list in $C[i''][j]$ we check whether $y_j^{k_j - 1} w_j$ is a prefix of $w[i''..n]$, and, if yes, insert the list in the tree $M[i][j]$, where i is the ending position of $y_j^{k_j - 1} w_j$. The usage of the trees $C[\cdot][j]$ is justified as they store first assignments of the variables $y_1, \ldots, y_j$ that map only $w_0 \Pi_{i=1,j-1}(y_i^{k_i} w_i) y_j$ to prefixes of w, then we look for alignments of $w_0 \Pi_{i=1,j-1}(y_i^{k_i} w_i) y_j^{k_j}$ with prefixes of w, and finally, in M, we store assignments of the variables that map α_j to prefixes of w.

The total time needed to construct the trees stored in the matrix M is clearly upper bounded by $\mathcal{O}(nmk\binom{n}{k-1}\binom{n}{k}) = \mathcal{O}(\frac{n^{2k}m}{((k-1)!)^2})$.

We conclude that $w \in L(\alpha)$ if and only if $M[n][m]$ is not empty. □

Similar to the case of PAT_{snc} and PAT_{nc} we show the following result.

Theorem 14. *We can compute a* $\text{PAT}_{\text{scd}\leq k}$*-descriptive pattern for a given sample* S *in* $\mathcal{O}(\frac{n^{2k}m^2}{((k-1)!)^2})$ *time, where* n *is the total length of the words in* S *and* m *is the length of its shortest word.*

The only major difference, that has to be pointed out, is that now we can choose the way to prolongate the constructed pattern in at most $p + 1$ ways, when

executing the cycle in Line 2 of Algorithm 1 for $i = p$. Thus, the additional m factor in the overall complexity of the algorithm.

5 Polynomial Time Inference of Pattern Languages

In this section, we exhibit in more detail the connections between computing descriptive patterns and inductive inference of pattern languages. To this end, we shall first briefly define the basic concepts of inductive inference (for a more detailed reference see, e. g., [25]). Let $\mathcal{L}$ be a language class with descriptors $\mathcal{D}$, i. e., $\mathcal{L} = \{L(D) \mid D \in \mathcal{D}\}$. An *inference machine $\mathcal{I}$ for $\mathcal{L}$* is an algorithm that receives as input a *positive representation* of some $L \in \mathcal{L}$, i. e., a sequence of words $w_1, w_2, \ldots$ from L such that for every $w \in L$ there is an $i \in \mathbb{N}$ with $w = w_i$, and after each new word a hypothesis $D \in \mathcal{D}$ is produced. If, for every positive representation of some $L \in \mathcal{L}$ as input, $\mathcal{I}$ produces a hypothesis $D \in \mathcal{D}$ with $L(D) = L$ after finite steps and does not change this hypothesis anymore, then *$\mathcal{I}$ infers $\mathcal{L}$ from positive data*. We say that *$\mathcal{I}$ infers $\mathcal{L}$ from positive data in polynomial time* if the step of producing a new hypothesis after receiving the next word only requires time polynomial in the sum of the length of the words received so far. An inference machine is called *consistent* if every produced hypothesis describes a language that contains all words received so far.

A language class $\mathcal{L}$ has *finite thickness* if, for every word w, there are at most finitely many $L \in \mathcal{L}$ with $w \in L$. Furthermore, a *minl-algorithm for $\mathcal{L}$* is an algorithm that, for a given sample S, computes a $D \in \mathcal{D}$ such that $S \subseteq L(D)$ and there exists no $L' \in \mathcal{L}$ with $S \subseteq L' \subset L(D)$. Obviously, an algorithm computing Π-descriptive patterns is a minl-algorithm for the class of Π-pattern languages. It has been shown in [2] that if a language class $\mathcal{L}$ has finite thickness and there exists a minl-algorithm for $\mathcal{L}$, then the following simple procedure describes an inference machine for $\mathcal{L}$: for every new word that is not described by the current hypothesis, we compute a new hypothesis by running the minl-algorithm on all words received so far as input. Moreover, if the minl-algorithm has a polynomial running time, then this inference machine infers $\mathcal{L}$ in polynomial time. It can be easily verified that, for every class Π of patterns, the class of Π-pattern languages has finite thickness. Consequently, we obtain the following corollary of Theorem 5:

Corollary 1. *Let Π be a rich class of patterns such that the question whether $\alpha \in \Pi$ can be decided in polynomial time and the membership problem for Π-pattern languages can be decided in polynomial time. Then the class of Π-pattern languages is polynomial time inferable from positive data.*

Polynomial time inference of pattern languages was first investigated with respect to the classes $\Pi_{\mathrm{var}\leq k}$, $k \in \mathbb{N}$, of k-variable patterns. This is most likely due to the fact that the membership problem for these classes can obviously be solved in polynomial time. More precisely, Angluin shows in [1] that 1-variable pattern languages can be inferred in polynomial time, but, as shown in [12], her approach cannot be easily extended to the classes $\Pi_{\mathrm{var}\leq k}$, $k \geq 2$. Improvements

of this result can be found in [4,20] and in [11,21] the problem of learning $\Pi_{\mathrm{var}\leq k}$-pattern languages is investigated with respect to different learning models. In the context of this paper, it is particularly worth noting that the classes $\Pi_{\mathrm{var}\leq k}$, $k \in \mathbb{N}$, are not rich classes, which means that a polynomial time inference machine for $\Pi_{\mathrm{var}\leq k}$-pattern languages cannot be derived from our Corollary 1. In fact, to the knowledge of the authors, it is still an open question whether or not $\Pi_{\mathrm{var}\leq k}$-descriptive patterns can be computed in polynomial time.

Lange and Wiehagen present in [14] an *inconsistent* polynomial time inference machine for Pat-pattern languages. By using this inference machine, Lange shows in [13] that $\Pi_{\mathrm{var}\leq k}$-pattern languages are also *consistently* polynomial time inferable. However, the inference machine presented in [13] may produce overly general hypotheses from time to time, namely, the hypothesis x_1. Alternatively, since the class $\Pi^{\mathrm{r}}_{\mathrm{var}\leq k}$ of all patterns with at most k *repeated* variables is rich and the corresponding membership problem can be solved in polynomial time, we can construct a polynomial time inference machine based on the algorithm $\Pi^{\mathrm{r}}_{\mathrm{var}\leq k}$-DescPat in the way described above. Since $\Pi_{\mathrm{var}\leq k} \subseteq \Pi^{\mathrm{r}}_{\mathrm{var}\leq k}$, this inference machine infers all k-variable pattern languages in polynomial time, but, in the strict sense, it is not an inference machine for the class of $\Pi_{\mathrm{var}\leq k}$-pattern languages, since it produces patterns from $\Pi^{\mathrm{r}}_{\mathrm{var}\leq k}$ as hypothesis. However, this inference machine can be easily transformed into one for $\Pi_{\mathrm{var}\leq k}$-pattern languages by using the same idea of [13], i. e., whenever the hypothesis is not in $\Pi_{\mathrm{var}\leq k}$, we simply output x_1 as hypothesis. Hence, we present an alternative proof of the result given in [13].

We conclude that, since $\Pi_{\mathrm{var}\leq k} \subseteq \Pi^{\mathrm{r}}_{\mathrm{var}\leq k} \subseteq$ Pat, the inference machines of [14] and [13] as well as the one based on $\Pi^{\mathrm{r}}_{\mathrm{var}\leq k}$-DescPat sketched above can all be used in order to infer in polynomial time $\Pi_{\mathrm{var}\leq k}$-pattern languages. However, we claim that in a practical scenario the one based on $\Pi^{\mathrm{r}}_{\mathrm{var}\leq k}$-DescPat is to be preferred, since it always produces hypotheses patterns for which the membership problem can be solved in polynomial time and that are at least as descriptive as $\Pi_{\mathrm{var}\leq k}$-descriptive patterns, which is not the case for the inference machines presented in [14] and [13].

References

1. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
2. D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
3. M. Crochemore and W. Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995.
4. T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger, and T. Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261:119–156, 2001.
5. D. D. Freydenberger and D. Reidenbach. Bad news on decision problems for patterns. *Information and Computation*, 208:83–96, 2010.
6. D.D. Freydenberger and D. Reidenbach. Existence and nonexistence of descriptive patterns. *Theoretical Computer Science*, 411:3274–3286, 2010.

7. E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
8. D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.
9. T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Decision problems for patterns. *Journal of Computer and System Sciences*, 50:53–63, 1995.
10. J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, 2006.
11. M. Kearns and L. Pitt. A polynomial-time algorithm for learning k–variable pattern languages from examples. In *Proc. 2nd Annual Conference on Learning Theory, COLT 1989*, pages 57–71, 1989.
12. K.-I. Ko and C.-M. Hua. A note on the two-variable pattern-finding problem. *Journal of Computer and System Sciences*, 34:75–86, 1987.
13. S. Lange. A note on polynominal-time inference of k-variable pattern languages. In *Proc. 1st International Workshop on Nonmonotonic and Inductive Logic, NIL 1990*, volume 543 of *LNCS*, pages 178–183, 1991.
14. S. Lange and R. Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.
15. A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 230–242. Springer, 1997.
16. Z. Mazadi, Z. Gao, and S. Zilles. Distinguishing pattern languages with membership examples. In *Proc. 8th International Conference on Language and Automata Theory and Applications, LATA 2014*, volume 8370 of *LNCS*, pages 528–540, 2014.
17. D. Reidenbach. A non-learnable class of E-pattern languages. *Theoretical Computer Science*, 350:91–102, 2006.
18. D. Reidenbach and M.L.Schmid. Patterns with bounded treewidth. *Information and Computation*. To appear.
19. D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. In *Proc. 6th Int. Conf. on Language and Automata Theory and Applications, LATA 2012*, volume 7183 of *LNCS*, pages 468–479, 2012.
20. R. Reischuk and T. Zeugmann. Learning one-variable pattern languages in linear average time. In *Proc. 11th Annual Conference on Computational Learning Theory, COLT 1998*, pages 198–208, 1998.
21. P. Rossmanith and T. Zeugmann. Stochastic finite learning of the pattern languages. *Machine Learning*, 44:67–91, 2001.
22. K. Salomaa. Patterns. In C. Martin-Vide, V. Mitrana, and G. Păun, editors, *Formal Languages and Applications*, number 148 in Studies in Fuzziness and Soft Computing, pages 367–379. Springer, 2004.
23. 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.
24. 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.
25. T. Shinohara and S. Arikawa. Pattern inference. In K.P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*, volume 961 of *Lecture Notes in Artificial Intelligence*, pages 259–291. Springer, Berlin, 1995.
26. T. Shinohara and H. Arimura. Inductive inference of unbounded unions of pattern languages from positive data. *Theoretical Computer Science*, 241:191–209, 2000.

27. R. Wiehagen and T. Zeugmann. Ignoring data may be the only way to learn efficiently. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:131–144, 1994.