A Polynomial Time Match Test for Large Classes of Extended Regular Expressions

Daniel Reidenbach and Markus L. Schmid *

Department of Computer Science, Loughborough University,
Loughborough, Leicestershire, LE11 3TU, United Kingdom
{D.Reidenbach,M.Schmid}@lboro.ac.uk

Abstract. In the present paper, we study the match test for extended regular expressions. We approach this NP-complete problem by introducing a novel variant of two-way multihead automata, which reveals that the complexity of the match test is determined by a hidden combinatorial property of extended regular expressions, and it shows that a restriction of the corresponding parameter leads to rich classes with a polynomial time match test. For presentational reasons, we use the concept of pattern languages in order to specify extended regular expressions. While this decision, formally, slightly narrows the scope of our results, an extension of our concepts and results to more general notions of extended regular expressions is straightforward.

1 Introduction

Regular expressions are compact and convenient devices that are widely used to specify regular languages, e. g., when searching for a pattern in a string. In order to overcome their limited expressive power while, at the same time, preserving their desirable compactness, their definition has undergone various modifications and extensions in the past decades. These amendments have led to several competing definitions, which are collectively referred to as *extended regular expressions* (or: *REGEX* for short). Hence, today's text editors and programming languages (such as Java and Perl) use individual notions of (extended) regular expressions, and they all provide so-called *REGEX engines* to conduct a *match test*, i. e., to compute the solution to the membership problem for any language given by a REGEX and an arbitrary string. While the introduction of new features of extended regular expressions have frequently not been guided by theoretically sound analyses, recent studies have led to a deeper understanding of their properties (see, e. g., Câmpeanu et al. [3]).

A common feature of extended regular expressions not to be found in the original definition is the option to postulate that each word covered by a specific REGEX must contain a variable substring at several recurrent positions (so-called *backreferences*). Thus, they can be used to specify a variety of non-regular languages (such as the language of all words w that satisfy $w = xx$ for arbitrary

* Corresponding author.

words x), and this has severe consequences on the complexity of their basic decision problems. In particular, their vital membership problem (i. e., in other words, the match test) is NP-complete (see Aho [1]). REGEX engines commonly use more or less sophisticated *backtracking* algorithms over extensions of NFA in order to perform the match test (see Friedl [5]), often leading even for rather small inputs to a practically unbearable runtime. Therefore, it is a worthwhile task to investigate alternative approaches to this important problem and to establish large classes of extended regular expressions with a polynomial-time match test.

It is the purpose of this paper to propose and study such an alternative method. In order to keep the technical details reasonably concise we do not directly use a particular REGEX definition, but we consider a well-established type of formal languages that, firstly, is defined in a similar yet simpler manner, secondly, is a proper subclass of the languages generated by REGEX and, thirdly, shows the same properties with regard to the membership problem: the *pattern languages* as introduced by Angluin [2]; our results on pattern languages can then directly be transferred to the corresponding class of REGEX. In this context, a *pattern* α is a finite string that consists of *variables* and *terminal symbols* (taken from a fixed alphabet Σ), and its language is the set of all words that can be derived from α when substituting arbitrary words over Σ for the variables. For example, the language L generated by the pattern $\alpha := x_1\mathtt{a}x_2\mathtt{b}x_1$ (with variables x_1, x_2 and terminal symbols $\mathtt{a}, \mathtt{b}$) consists of all words with an arbitrary prefix u, followed by the letter $\mathtt{a}$, an arbitrary word v, the letter $\mathtt{b}$ and a suffix that equals u. Thus, $w_1 := \mathtt{aaabbaa}$ is contained in L, whereas $w_2 := \mathtt{baaba}$ is not.

In the definition of pattern languages, the option of using several occurrences of a variable exactly corresponds to the backreferences in extended regular expressions, and therefore the membership problem for pattern languages captures the essence of what is computationally complex in the match test for REGEX. Thus, it is not surprising that the membership problem for pattern languages is also known to be NP-complete (see Angluin [2] and Jiang et al. [10]). Furthermore, Ibarra et al. [9] point out that the membership problem for pattern languages is closely related to the solvability problem for certain Diophantine equations. More precisely, for any word w and for any pattern α with m terminal symbols and n different variables, w can only be contained in the language generated by α if there are numbers s_i (representing the lengths of the substitution words for the variables x_i) such that $|w| = m + \sum_{i=1}^{n} a_i s_i$ (where a_i is the number of occurrences of x_i in α and $|w|$ stands for the *length* of w). Thus, the membership test needs to implicitly solve this NP-complete problem, which is related to *Integer Linear Programming* problems (see the references in [9]) and the *Money-Changing Problem* (see Guy [6]). All these insights into the complexity of the membership problem do not depend on the question of whether the pattern contains any terminal symbols. Therefore, we can safely restrict our considerations to so-called *terminal-free* pattern languages (generated by patterns that consist of variables only); for this case, NP-completeness of the membership problem has indirectly been established by Ehrenfeucht and Rozenberg [4]. This

restriction again improves the accessibility of our technical concepts, without causing a loss of generality.

As stated above, these results on the complexity of the problem (and the fact that probabilistic solutions might often be deemed inappropriate for it) motivate the search for large subclasses with efficiently solvable membership problem and for suitable concepts realising the respective algorithms. Rather few such classes are known to date. They either restrict the number of *different* variables in the patterns to a fixed number k (see Angluin [2], Ibarra et al. [9]), which is an obvious option and leads to a time complexity of $\mathrm{O}(n^k)$, or they restrict the number of *occurrences* of each variable to 1 (see Shinohara [11]), which turns the resulting pattern languages into regular languages.

In the present paper, motivated by Shinohara's [12] *non-cross* pattern languages, we introduce major classes of pattern languages (and, hence, of extended regular expressions) with a polynomial-time membership problem that do not show any of the above limitations. Thus, the corresponding patterns can have any number of variables with any number of occurrences; instead, we consider a rather subtle parameter, namely the *distance* several occurrences of any variable x may have in a pattern (i. e., the maximum number of different variables separating any two consecutive occurrences of x). We call this parameter the *variable distance* vd of a pattern, and we demonstrate that, for the class of all patterns with $\mathrm{vd} \leq k$, the membership problem is solvable in time $\mathrm{O}(n^{k+4})$. Referring to the proximity between the subject of our paper and the solvability problem of the equation $|w| = m + \sum_{i=1}^{n} a_i s_i$ described above (which does not depend on the order of variables in the patterns, but merely on their numbers of occurrences), we consider this insight quite remarkable, and it is only possible since this solvability problem is *weakly* NP-complete (i. e. there exist pseudo-polynomial time algorithms). We also wish to point out that, in terms of our concept, Shinohara's non-cross patterns correspond to those patterns with $\mathrm{vd} = 0$.

We prove our main result by introducing the concept of a *Janus automaton*, which is a variant of a two-way two-head automaton (see Ibarra [7]), amended by the addition of a number of counters. Janus automata are algorithmic devices that are tailored to performing the match test for pattern languages, and we present a systematic way of constructing them. While an intuitive use of a Janus automaton assigns a distinct counter to each variable in the corresponding pattern α, we show that in our advanced construction the number of different counters can be limited by the variable distance of α. Since the number of counters is the main element determining the complexity of a Janus automaton, this yields our main result. An additional effect of the strictness of our approach is that we can easily discuss its quality in a formal manner, and we can show that, based on a natural assumption on how Janus automata operate, our method leads to an automaton with the smallest possible number of counters. Furthermore, it is straightforward to couple our Janus automata with ordinary finite automata in order to expand our results to more general classes of extended regular expressions, e. g., those containing terminal symbols or imposing regular restrictions to the sets of words variables can be substituted with.

In order to validate our claim that the variable distance is a crucial parameter contributing to the complexity of the match test, and to examine whether our work – besides its theoretical value – might have any practical relevance, some instructive tests have been performed.[1] They compare a very basic Java implementation of our Janus automata with the original REGEX engine included in Java. With regard to the former objective, the test results suggest that our novel notion of a variable distance is indeed a crucial (and, as briefly mentioned above, rather counter-intuitive) parameter affecting the complexity of the match test for both our Janus-based algorithm and the established backtracking method. Concerning the latter goal, we can observe that our non-optimised implementation, on average, considerably outperforms Java's REGEX engine. We therefore conclude that our approach might also be practically worthwhile.

2 Definitions

Let $\mathbb{N} := \{0, 1, 2, 3, \ldots\}$. For an arbitrary alphabet A, a *string* (*over* A) is a finite sequence of symbols from A, and ε stands for the *empty string*. The symbol A^+ denotes the set of all nonempty strings over A, and $A^* := A^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a string $v \in A^*$ is a *factor* of a string $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1 \cdot v \cdot u_2$. The notation $|K|$ stands for the size of a set K or the length of a string K; the term $|w|_a$ refers to the number of occurrences of the symbol a in the string w.

For any alphabets A, B, a *morphism* is a function $h : A^* \to B^*$ that satisfies $h(vw) = h(v)h(w)$ for all $v, w \in A^*$. Let Σ be a (finite) alphabet of so-called *terminal symbols* and X an infinite set of *variables* with $\Sigma \cap X = \emptyset$. We normally assume $X := \{x_1, x_2, x_3, \ldots\}$. A *pattern* is a nonempty string over $\Sigma \cup X$, a *terminal-free pattern* is a nonempty string over X and a *word* is a string over Σ. For any pattern α, we refer to the set of variables in α as $\mathrm{var}(\alpha)$. We shall often consider a terminal-free pattern in its variable factorisation, i. e. $\alpha = y_1 \cdot y_2 \cdot \ldots \cdot y_n$ with $y_i \in \{x_1, x_2, \ldots, x_m\}$, $1 \leq i \leq n$ and $m = |\mathrm{var}(\alpha)|$. A morphism $\sigma : (\Sigma \cup X)^* \to \Sigma^*$ is called a *substitution* if $\sigma(a) = a$ for every $a \in \Sigma$.

We define the *pattern language* of a terminal-free pattern α by $L_\Sigma(\alpha) := \{\sigma(\alpha) \mid \sigma : X^* \to \Sigma^* \text{ is a substitution}\}$. Note, that these languages, technically, are terminal-free E-pattern languages (see Jiang et al. [10]). We ignore the case where a variable occurs just once, as then $L_\Sigma(\alpha) = \Sigma^*$.

The problem to decide for a given pattern α and a given word $w \in \Sigma^*$ whether $w \in L_\Sigma(\alpha)$ is called the *membership problem*.

3 Janus Automata

In the present section we introduce a novel type of automata, the so-called Janus automata, that are tailored to solving the membership problem for pattern lan-

[1] Tests and source code are available at `http://www-staff.lboro.ac.uk/~coms10/`.

guages. To this end, we combine elements of two-way multihead finite automata (see, e. g., Ibarra [7]) and counter machines (see, e. g., Ibarra [8]).

A *Janus automaton* (or JFA(k) for short) is a two-way 2-head automaton with k restricted counters, $k \in \mathbb{N}$. More precisely, it is a tuple $M := (k, Q, \Sigma, \delta, q_0, F)$, where Σ is an *input alphabet*, δ is a *transition function*, Q is a set of *states*, $F \subseteq Q$ is a set of *final states*, and $q_0 \in Q$ is the *initial state*. In each step of the computation the automaton M provides a distinct *counter bound* for each counter. The *counter values* can only be incremented or left unchanged and they count strictly modulo their counter bound, i. e. once a counter value has reached its counter bound, a further incrementation forces the counter to start at counter value 1 again. Depending on the current state, the currently scanned input symbols and on whether the counters have reached their bounds, the transition function determines the next state, the input head movements and the counter instructions. In addition to the counter instructions of incrementing and leaving the counter unchanged it is also possible to reset a counter. In this case, the counter value is set to 0 and a new counter bound is nondeterministically guessed. Furthermore, we require the first input head to be always positioned to the left of the second input head, so there are a well-defined *left* and *right head*. Therefore, we call this automata model a "Janus" automaton. Any string $\&w\$$, where $w \in \Sigma^*$ and the symbols $\&, \$$ (referred to as *left* and *right endmarker*, respectively) are not in Σ, is an *input* to M. Initially, the *input tape* stores some input, M is in state q_0, all counter bounds and counter values are 0 and both input heads scan $\&$. The word w is accepted by M if and only if it is possible for M to reach an accepting state by succesively applying the transition function. For any Janus automaton M let $L(M)$ denote the set of words accepted by M.

JFA(k) are nondeterministic automata, but their nondeterminism differs from that of common nondeterministic finite automata. The only nondeterministic step a Janus automaton is able to perform consists in guessing a new counter bound for some counter. Once a new counter bound is guessed, the previous one is lost. Apart from that, each transition, i. e. entering a new state, moving the input heads and giving instructions to the counters, is defined completely deterministically by δ.

The vital point of a JFA(k) computation is then, of course, that the automaton is only able to save exactly k (a constant number, not depending on the input word) different numbers at a time. For a JFA(k) M, the number k shall be the crucial number for the complexity of the *acceptance problem* (*for* M), i. e. to decide, for a given word w, whether $w \in L(M)$.

4 Janus Automata for Pattern Languages

In this chapter, we demonstrate how Janus automata can be used for recognising pattern languages. More precisely, for an arbitrary terminal-free pattern α, we construct a JFA(k) M satisfying $L(M) = L_\Sigma(\alpha)$. Before we move on to a formal analysis of this task, we discuss the problem of deciding whether $w \in L_\Sigma(\alpha)$ for given α and w, i. e. the membership problem, in an informal way.

Let $\alpha = y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern with $m := |\operatorname{var}(\alpha)|$, and let $w \in \Sigma^*$ be a word. The word w is an element of $L_\Sigma(\alpha)$ if and only if there exists a factorisation $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$ such that $u_j = u_{j'}$ for all $j, j' \in \{1, 2, \ldots, |\alpha|\}$ with $y_j = y_{j'}$. Thus, a way to solve the membership problem is to initially guess m numbers $\{l_1, l_2, \ldots, l_m\}$, then, if possible, to factorise $w = u_1 \cdot \ldots \cdot u_n$ such that $|u_j| = l_i$ for all j with $y_j = x_i$ and, finally, to check whether $u_j = u_{j'}$ is satisfied for all $j, j' \in \{1, 2, \ldots, |\alpha|\}$ with $y_j = y_{j'}$. A JFA(m) can perform this task by initially guessing m counter bounds, which can be interpreted as the lengths of the factors. The two input heads can be used to check if this factorisation has the above described properties. However, the number of counters that are then required directly depends on the number of variables, and the question arises if this is always necessary. The next step is to formalise and generalise the way of constructing a JFA(k) for arbitrary pattern languages.

Definition 1. *Let $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern, and let $n_i := |\alpha|_{x_i}$ for each $x_i \in \operatorname{var}(\alpha)$. The set $\operatorname{varpos}_i(\alpha)$ is the set of all positions j satisfying $y_j = x_i$. Let furthermore $\Gamma_i := ((l_1, r_1), (l_2, r_2), \ldots, (l_{n_i-1}, r_{n_i-1}))$ with $(l_j, r_j) \in varpos_i(\alpha)^2$ and $l_j < r_j$, $1 \leq j \leq n_i - 1$. The sequence Γ_i is a* matching order for x_i in α *if and only if the graph $(varpos_i(\alpha), \Gamma_i')$ is connected, where $\Gamma_i' := \{(l_1, r_1), (l_2, r_2), \ldots, (l_{n_i-1}, r_{n_i-1})\}$. The elements $m_j \in varpos_i(\alpha)^2$ of a matching order $(m_1, m_2, \ldots, m_k)$ are called* matching positions.

We illustrate Definition 1 by the example pattern $\beta := x_1 \cdot x_2 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_2 \cdot x_3$. Possible matching orders for x_1, x_2 and x_3 in β are given by $((1, 3))$, $((2, 4), (4, 6))$ and $((5, 7))$, respectively. To obtain a matching order for a pattern α we simply combine matching orders for all $x \in \operatorname{var}(\alpha)$:

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

With respect to our example pattern β this means that any sequence of the matching positions in $\{(1, 3), (2, 4), (4, 6), (5, 7)\}$ is a complete matching order for β. As pointed out by the following lemma, the concept of a complete matching order can be used to solve the membership problem.

Lemma 1. *Let $\alpha = y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern and $((l_1, r_1), (l_2, r_2), \ldots, (l_k, r_k))$ a complete matching order for α. Let w be an arbitrary word in some factorisation $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$. If $u_{l_j} = u_{r_j}$ for each j with $1 \leq j \leq k$, then $u_j = u_{j'}$ for all $j, j' \in \{1, 2, \ldots, |\alpha|\}$ with $y_j = y_{j'}$.*

Let $\alpha = y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern and let w be an arbitrary word in some factorisation $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$. According to the previous lemma, we may interpret a complete matching order as a list of instructions specifying how the factors u_i, $1 \leq i \leq n$, can be compared in order to check if $u_j = u_{j'}$ for all $j, j' \in \{1, 2, \ldots, |\alpha|\}$ with $y_j = y_{j'}$, which is of course characteristic for $w \in$

$L_\Sigma(\alpha)$. With respect to the complete matching order $((4,6),(1,3),(2,4),(5,7))$ for the example pattern β, we apply Lemma 1 in the following way. If a word $w \in \Sigma^*$ can be factorised into $w = u_1 \cdot u_2 \cdot \ldots \cdot u_7$ such that $u_4 = u_6$, $u_1 = u_3$, $u_2 = u_4$ and $u_5 = u_7$ then $w \in L_\Sigma(\beta)$. These matching instructions given by a complete matching order can be carried out by using two pointers, or input heads, moving over the word w.

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

Definition 3. *Let $((l_1, r_1), (l_2, r_2), \ldots, (l_k, r_k))$ be a complete matching order for a terminal-free pattern α and let $l_0 := r_0 := 0$. For all j, j', $1 \leq j < j' \leq |\alpha|$ we define $g(j, j') := (j+1, j+2, \ldots, j'-1)$ and $g(j', j) := (j', j'-1, \ldots, j)$. For each i with $1 \leq i \leq k$ we define $D_i^\lambda := ((p_1, \lambda), (p_2, \lambda), \ldots, (p_{k_1}, \lambda))$ and $D_i^\rho := ((p'_1, \rho), (p'_2, \rho), \ldots, (p'_{k_2}, \rho))$, where $(p_1, p_2, \ldots, p_{k_1}) := g(l_{i-1}, l_i)$, $(p'_1, p'_2, \ldots, p'_{k_2}) := g(r_{i-1}, r_i)$ and λ, ρ are constant markers. Now let $D'_i := ((s_1, \mu_1), (s_2, \mu_2), \ldots, (s_{k_1+k_2}, \mu_{k_1+k_2}))$, with $s_j \in \{p_1, \ldots, p_{k_1}, p'_1, \ldots, p'_{k_2}\}$, $\mu_j \in \{\lambda, \rho\}$, $1 \leq j \leq k_1 + k_2$, be a tuple containing exactly the elements of D_i^λ and D_i^ρ such that the relative orders of the elements in D_i^λ and D_i^ρ are preserved. Furthermore, for each j, $1 \leq j \leq k_1 + k_2$, $q_j \leq q'_j$ needs to be satisfied, where $q_j := l_{i-1}$ if $\mu_{j'} = \rho$, $1 \leq j' \leq j$, and $q_j := \max\{j' \mid 1 \leq j' \leq j, \mu_{j'} = \lambda\}$ else, analogously, $q'_j := r_{i-1}$ if $\mu_{j'} = \lambda$, $1 \leq j' \leq j$, and $q'_j := \max\{j' \mid 1 \leq j' \leq j, \mu_{j'} = \rho\}$ else. Now we append the two elements (r_i, ρ), (l_i, λ) in exactly this order to the end of D'_i and obtain D_i. Finally, the tuple $(D_1, D_2, \ldots, D_k)$ is called a* Janus operating mode for α (derived from the complete matching order $((l_1, r_1), (l_2, r_2), \ldots, (l_k, r_k))$). *By $\overline{D_i}$, we denote the tuple D_i without the markers, i. e., if $D_i = ((p_1, \mu_1), \ldots, (p_n, \mu_n))$ with $\mu_j \in \{\lambda, \rho\}$, $1 \leq j \leq n$, then $\overline{D_i} := (p_1, p_2, \ldots, p_n)$.*

We recall once again the example $\beta := x_1 \cdot x_2 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_2 \cdot x_3$. According to Definition 3 we consider the tuples D_i^λ and D_i^ρ with respect to the complete matching order $((4,6),(1,3),(2,4),(5,7))$ for β. We omit the markers λ and ρ for a better presentation. The tuples D_i^λ are given by $D_1^\lambda = (1,2,3)$, $D_2^\lambda = (4,3,2,1)$, $D_3^\lambda = ()$ and $D_4^\lambda = (3,4)$. The tuples D_i^ρ are given by $D_1^\rho = (1,2,\ldots,5)$, $D_2^\rho = (6,5,4,3)$, $D_3^\rho = ()$ and $D_4^\rho = (5,6)$. Therefore, $\Delta_\beta := (D_1, D_2, D_3, D_4)$ is a possible Janus operating mode for β derived from $((4,6),(1,3),(2,4),(5,7))$, where $D_1 = ((1,\rho), (1,\lambda), (2,\rho), (2,\lambda), (3,\rho), (3,\lambda), (4,\rho), (5,\rho), (6,\rho), (4,\lambda))$, $D_2 = ((4,\lambda), (3,\lambda), \ldots, (1,\lambda), (6,\rho), (5,\rho), \ldots, (3,\rho), (3,\rho), (1,\lambda))$, $D_3 = ((4,\rho), (2,\lambda))$, $D_4 = ((3,\lambda), (4,\lambda), (5,\rho), (6,\rho), (7,\rho), (5,\lambda))$.

We shall see that it is possible to transform a Janus operating mode for any pattern directly into a Janus automaton recognising the corresponding pattern language. As we are particularly interested in the number of counters a Janus automaton needs, we introduce an instrument to determine the quality of Janus operating modes with respect to the number of counters that are required to actually construct a Janus automaton.

Definition 4. *Let $(D_1, D_2, \ldots, D_k)$ be a Janus operating mode for a terminal-free pattern $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$. Let $D = (d'_1, d'_2, \ldots, d'_{k'})$ with $k' = \sum_{i=1}^{k} |\overline{D_i}|$ be the tuple obtained from concatenating all tuples $\overline{D_j}$, $1 \leq j \leq k$, in the order given by the Janus operating mode. For each i, $1 \leq i \leq k'$, let $s_i := |\{x \mid \exists\, j, j' \text{ with } 1 \leq j < i < j' \leq k', y_{d'_j} = y_{d'_{j'}} = x \neq y_{d'_i}\}|$. Finally let the* counter number of $(D_1, D_2, \ldots, D_k)$ *(denoted by $\mathrm{cn}(D_1, D_2, \ldots, D_k)$) be $\max\{s_i \mid 1 \leq i \leq k'\}$.*

With regard to our example β, it can be easily verified that $\mathrm{cn}(\Delta_\beta) = 2$. The counter number of a Janus operating mode of a pattern α is an upper bound for the number of counters needed by a Janus automaton recognising $L_\Sigma(\alpha)$:

Theorem 1. *Let α be a terminal-free pattern and $(D_1, D_2, \ldots, D_k)$ be an arbitrary Janus operating mode for α. There exists a $\mathrm{JFA}(\mathrm{cn}(D_1, \ldots, D_k) + 1)$ M satisfying $L(M) = L_\Sigma(\alpha)$.*

Hence, the task of finding an optimal Janus automaton for a pattern is equivalent to finding an optimal Janus operating mode for this pattern. We shall investigate this problem in the subsequent section.

5 Patterns with Restricted Variable Distance

We now introduce a certain combinatorial property of terminal-free patterns, the so-called variable distance. The variable distance of a terminal-free pattern is the maximum number of different variables separating any two consecutive occurrences of a variable:

Definition 5. *The* variable distance *of a terminal-free pattern α is the smallest number $k \geq 0$ such that, for each $x \in \mathrm{var}(\alpha)$, every factorisation $\alpha = \beta \cdot x \cdot \gamma \cdot x \cdot \delta$ with $\beta, \gamma, \delta \in X^*$ and $|\gamma|_x = 0$ satisfies $|\mathrm{var}(\gamma)| \leq k$. We denote the variable distance of a terminal-free pattern α by $\mathrm{vd}(\alpha)$.*

Obviously, $\mathrm{vd}(\alpha) \leq \mathrm{var}(\alpha) - 1$ for all terminal-free patterns α. To illustrate the concept of the variable distance, we consider the slightly more involved pattern $\alpha := x_1 \cdot x_2 \cdot x_1 \cdot x_3 \cdot x_2 \cdot x_2 \cdot x_2 \cdot x_4 \cdot x_4 \cdot x_5 \cdot x_5 \cdot x_3$. In α, there are no variables between occurrences of variables x_4 or x_5 and one occurrence of x_2 between the two occurrences of x_1. Furthermore, the variables x_1 and x_3 occur between occurrences of x_2 and the variables x_2, x_4 and x_5 occur between the two occurrences of x_3. Thus, the variable distance of this pattern is 3.

The following vital result demonstrates the relevance of the variable distance, which is a lower bound for the counter number of Janus operating modes.

Theorem 2. *Let $(D_1, D_2, \ldots, D_k)$ be an arbitrary Janus operating mode for a terminal-free pattern α. Then* $\mathrm{cn}(D_1, \ldots, D_k) \geq \mathrm{vd}(\alpha)$.

In order to define a Janus operating mode satisfying $\mathrm{cn}(D_1, \ldots, D_k) = \mathrm{vd}(\alpha)$, we now consider a particular matching order:

Definition 6. *Let $\alpha := y_1 \cdot y_2 \cdot \ldots \cdot y_n$ be a terminal-free pattern with $p := |\operatorname{var}(\alpha)|$. For each $x_i \in \operatorname{var}(\alpha)$, let $varpos_i(\alpha) := \{j_{i,1}, j_{i,2}, \ldots, j_{i,n_i}\}$ with $n_i := |\alpha|_{x_i}$, $j_{i,l} < j_{i,l+1}$, $1 \leq l \leq n_i - 1$. Let $(m_1, m_2, \ldots, m_k)$, $k = \sum_{i=1}^{p} n_i - 1$, be an enumeration of the set $\{(j_{i,l}, j_{i,l+1}) \mid 1 \leq i \leq p, 1 \leq l \leq n_i - 1\}$ such that, for every i', $1 \leq i' < k$, the left element of the pair $m_{i'}$ is smaller than the left element of $m_{i'+1}$. We call $(m_1, m_2, \ldots, m_k)$ the* canonical matching order for α.

Proposition 1. *Let α be a terminal-free pattern. The canonical matching order for α is a complete matching order.*

For instance, the canonical matching order for the example pattern β introduced in Section 4 is $((1,3),(2,4),(4,6),(5,7))$. We proceed with the definition of a Janus operating mode that is derived from the canonical matching order. It is vital for the correctness of our results, that we first move the left head and then the right head. This is easily possible if for two consecutive matching positions $(l', r'), (l, r)$, $l < r'$. If this condition is not satisfied, then the left head may pass the right one, which conflicts with the definition of Janus operating modes. Therefore, in this case, we move the left head and right head alternately.

Definition 7. *Let $(m_1, m_2, \ldots, m_k)$ be the canonical matching order for a terminal-free pattern α. For any $m_i := (j_1, j_2)$ and $m_{i-1} := (j_1', j_2')$, $2 \leq i \leq k$, let $(p_1, p_2, \ldots, p_{k_1}) := g(j_1', j_1)$ and $(p_1', p_2', \ldots, p_{k_2}') := g(j_2', j_2)$, where g is the function introduced in Definition 3. If $j_1 \leq j_2'$, then we define*

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

If, on the other hand, $j_2' < j_1$, we define D_i in three parts

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

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

If we derive a Janus operating mode from the canonical matching order for β as described in Definition 7 we obtain the canonical Janus operating mode $(((1,\rho),(2,\rho),(3,\rho),(1,\lambda)),((4,\rho),(2,\lambda)),((3,\lambda),(5,\rho),(6,\rho),(4,\lambda)),((7,\rho),(5,\lambda)))$. This canonical Janus operating mode has a counter number of 1, so its counter number is smaller than the counter number of the example Janus operating mode Δ_β given in Section 4 and, furthermore, equals the variable distance of β. With Theorem 2 we conclude that the canonical Janus operating mode for β is optimal. The next lemma shows that this holds for every pattern and, together with Theorem 1, we deduce our first main result, namely that for arbitrary patterns α, there exists a $\mathrm{JFA}(\mathrm{vd}(\alpha)+1)$ exactly accepting $L_\Sigma(\alpha)$.

Lemma 2. *Let α be a terminal-free pattern and let $(D_1, D_2, \ldots, D_k)$ be the canonical Janus operating mode for α. Then* $\mathrm{cn}(D_1, \ldots, D_k) = \mathrm{vd}(\alpha)$.

Theorem 3. *Let α be a terminal-free pattern. There exists a* $\mathrm{JFA}(\mathrm{vd}(\alpha) + 1)$ *M such that $L(M) = L_\Sigma(\alpha)$.*

The Janus automaton obtained from the canonical Janus operating mode for a pattern α is called the *canonical Janus automaton*. Theorem 3 shows the optimality of the canonical automaton. However, this optimality is subject to a vital assumption: we assume that the automaton needs to know the length of a factor in order to move an input head over this factor.

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

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

As mentioned in the introduction, this main result also holds for more general classes of extended regular expressions. We anticipate, though, that the necessary amendments to our definitions involve some technical hassle.

References

[1] A. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, pages 255–300. MIT Press, 1990.

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

[3] C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.

[4] A. Ehrenfeucht and G. Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9:86–88, 1979.

[5] J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly, Sebastopol, CA, third edition, 2006.

[6] R. K. Guy. The money changing problem. In *Unsolved Problems in Number Theory*, chapter C7, pages 171–173. Springer, New York, third edition, 2004.

[7] O. Ibarra. On two-way multihead automata. *Journal of Computer and System Sciences*, 7:28–36, 1973.

[8] O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.

[9] O. Ibarra, T.-C. Pong, and S. Sohn. A note on parsing pattern languages. *Pattern Recognition Letters*, 16:179–182, 1995.

[10] T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *International Journal of Computer Mathematics*, 50:147–163, 1994.

[11] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *Proc. RIMS Symposia, Kyoto*, volume 147 of *LNCS*, pages 115–127, 1982.

[12] T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proc. 7th IBM MFCS*, pages 191–209, 1982.