International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Inside the Class of REGEX Languages

Markus L. Schmid
Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54296 Trier, Germany
M.Schmid@lboro.ac.uk

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

We study different possibilities of combining the concept of homomorphic replacement with regular expressions in order to investigate the class of languages given by extended regular expressions with backreferences (REGEX). It is shown in which regard existing and natural ways to do this fail to reach the expressive power of REGEX. Furthermore, the complexity of the membership problem for REGEX with a bounded number of backreferences is considered.

Keywords: Extended Regular Expressions, REGEX, Pattern Languages, Pattern Expressions, Homomorphic Replacement

1. Introduction

Since their introduction by Kleene in 1956 [12], *regular expressions* have not only constantly challenged researchers in formal language theory, they also attracted pioneers of applied computer science as, e. g., Thompson [15], who developed one of the first implementations of regular expressions, marking the beginning of a long and successful tradition of their practical application (see Friedl [9] for an overview). In order to suit practical requirements, regular expressions have undergone various modifications and extensions which lead to so-called *extended regular expressions with backreferences* (*REGEX* for short), nowadays a standard element of most text editors and programming languages (cf. Friedl [9]). The introduction of these new features of extended regular expressions has frequently not been guided by theoretically sound analyses and only recent studies have led to a deeper understanding of their properties (see, e. g., Câmpeanu et al. [5]).

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

For the REGEX $((_1\ \mathtt{a}^+\ )_1 \mid \mathtt{b})\, \mathtt{c}\, \backslash 1$, if we choose the option b in the alternation, then \1 points to a subexpression that has not been "initialised". Normally, such a backreference is then interpreted as the empty word.

Another particularity appears whenever a backreference points to a subexpression under a star, e. g., $s := ((_1\ \mathtt{a}^*\ )_1\, \mathtt{b}\, \backslash 1)^*\, \mathtt{c}\, \backslash 1$. One might expect s to define the set of all words of form $(\mathtt{a}^n\mathtt{b}\mathtt{a}^n)^m\mathtt{c}\mathtt{a}^n$, $n, m \geq 0$, but s really describes the set $\{\mathtt{a}^{n_1}\mathtt{b}\mathtt{a}^{n_1}\, \mathtt{a}^{n_2}\mathtt{b}\mathtt{a}^{n_2} \cdots \mathtt{a}^{n_m}\mathtt{b}\mathtt{a}^{n_m}\ \mathtt{c}\, \mathtt{a}^{n_m} \mid m \geq 1, n_i \geq 0, 1 \leq i \leq m\} \cup \{\mathtt{c}\}$. This is due to the fact that the star operation repeats a subexpression several times without imposing any dependencies between the single iterations. Consequently, in every iteration of the second star in s, the referenced subexpression $(_1\ \mathtt{a}^*\ )_1$ is treated as an individual instance and its scope is restricted to the current iteration. Only the factor that $(_1\ \mathtt{a}^*\ )_1$ matches in the very last iteration is then referenced by any backreference \1 outside the star. This behaviour is often called *late binding* of backreferences.

A suitable language theoretical approach to these backreferences is the concept of *homomorphic replacement*. For example, the REGEX r can also be given as a string $x\mathtt{b}x$, where the symbol x can be homomorphically replaced by words from $\{\mathtt{a}, \mathtt{b}\}^*$, i. e., both occurrences of x must be replaced by the same word. Numerous language generating devices can be found that use various kinds of homomorphic replacement. The most prominent example are probably the well-known L systems (see Kari et al. [11] for a survey), but also many types of grammars as, e. g., Wijngaarden grammars, macro grammars, Indian parallel grammars or deterministic iteration grammars, use homomorphic replacement as a central concept (cf. Albert and Wegner [2] and Bordihn et al. [4] and the references therein). Albert and Wegner [2] and Angluin [3] introduced H-systems and pattern languages, respectively, which both use homomorphic replacement in a more puristic way, without any grammar like mechanisms. More recent models like pattern expressions (Câmpeanu and Yu [6]), synchronized regular expressions (Della Penna et al. [13]) and EH-expressions (Bordihn et al. [4]) are mainly inspired directly by REGEX. While all these models have been introduced and analysed in the context of formal language theory, REGEX have mainly been formed by applications and especially cater for practical requirements. Hence, there is the need in formal language theory to catch up on these practical developments concerning REGEX and we can note that recent work is concerned with exactly that task (see, e. g., [5–8]).

The contribution of this paper is to investigate alternative possibilities to combine the two most elementary components of REGEX, i. e., regular expressions and homomorphic replacement, with the objective of reaching the expressive power of REGEX as close as possible, without exceeding it. Particularly challenging about REGEX is that due to the possible nesting of referenced subexpression the concepts of regular expressions and homomorphic replacement seem to be inherently entangled and there is no easy way to treat them separately. We illustrate this with the example $t := (_1\ \mathtt{a}^*\ )_1\, (_2\ (\mathtt{b}\, \backslash 1)^*\ )_2\, \backslash 2\, \backslash 1$. The language

$\mathcal{L}(t) := \{\mathtt{a}^n(\mathtt{ba}^n)^m(\mathtt{ba}^n)^m\mathtt{a}^n \mid n, m \geq 0\}$ cannot that easily be described in terms of a single string with a homomorphic replacement rule, e. g., by the string $xyyx$, where x can be replaced by words from $\{\mathtt{a}^n \mid n \geq 0\}$, and y by words of form $\{(\mathtt{ba}^n)^m \mid n, m \geq 0\}$, since then we can obtain words $\mathtt{a}^n(\mathtt{ba}^{n'})^m(\mathtt{ba}^{n'})^m\mathtt{a}^n$ with $n \neq n'$. In fact, two steps of homomorphic replacement seem necessary, i. e., we first replace y by words from $\{(\mathtt{b}z)^n \mid n \geq 0\}$ and after that we replace x and z by words from $\{\mathtt{a}^n \mid n \geq 0\}$, with the additional requirement that x and z are substituted by the same word. More intuitively speaking, the nesting of referenced subexpressions require *iterated* homomorphic replacement, but we also need to carry on information from one step of replacement to the next one.

The concept of homomorphic replacement is covered best by so-called *pattern languages* as introduced by Angluin [3]. A pattern is a string containing variables and terminal symbols and the corresponding pattern language is the set of all words that can be obtained from the pattern by homomorphically replacing the variables by terminal words. We combine Angluin's patterns with regular expressions by first adding the alternation and star operator to patterns and, furthermore, by letting their variables be typed by regular languages, i. e., the words variables are replaced with are from given regular sets. Then we iterate this step by using this new class of languages again as types for variables and so on. We also take a closer look at *pattern expressions*, which were introduced by Câmpeanu and Yu in [6]. In [6] many examples are provided that show how to translate a REGEX into an equivalent pattern expression and vice versa. It is also stated that this is possible in general, but a formal proof for this statement is not provided. In the present work we show that pattern expressions are in fact much weaker than REGEX and they describe a proper subset of the class of REGEX languages.

On the other hand, pattern expressions still describe an important and natural subclass of REGEX languages, that has been independently defined in terms of other models and, as shown in this work, also coincides with the class of languages resulting from the modification of patterns described above. We then refine the way of how pattern expressions define languages in order to accommodate the nesting of referenced subexpressions and we show that the thus obtained class of languages coincides with the class of languages given by REGEX that do not contain a referenced subexpression under a star.

Finally, we show that the membership problem for REGEX, which, in the unrestricted case, is NP-complete, can be solved in polynomial time provided that the number of different backreferences is restricted.

2. General Definitions

Let $\mathbb{N} := \{1, 2, 3, \ldots\}$ and let $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For an arbitrary alphabet A, a *word* (*over A*) is a finite sequence of symbols from A, and ε stands for the *empty word*. The notation A^+ denotes the set of all nonempty words over A, and $A^* := A^+ \cup \{\varepsilon\}$. For the *concatenation* of two words w_1, w_2 we write $w_1\, w_2$. We say that a word $v \in A^*$

is a *factor* of a word $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1\, v\, u_2$. The notation $|K|$ stands for the size of a set K or the length of a word K.

We use regular expression as they are commonly defined (see, e. g., Yu [16]). For the alternation operations we use the symbol "$|$" and in an alternation $(s \mid t)$, we call the subexpressions s and t *options*. For any regular expression r, $\mathcal{L}(r)$ denotes the language described by r and REG denotes the set of regular languages. Let Σ be a finite alphabet of *terminal symbols* and let $X := \{x_1, x_2, x_3, \ldots\}$ be a countably infinite set of *variables* with $\Sigma \cap X = \emptyset$. For any word $w \in (\Sigma \cup X)^*$, $\mathrm{var}(w)$ denotes the set of variables that occur in w.

3. Patterns with Regular Operators and Types

In this section, we combine the pattern languages mentioned in Section 1 with regular languages and regular expressions. Let $\mathrm{PAT} := \{\alpha \mid \alpha \in (\Sigma \cup X)^+\}$ and every $\alpha \in \mathrm{PAT}$ is called a *pattern*. We always assume that, for every $i \in \mathbb{N}$, $x_i \in \mathrm{var}(\alpha)$ implies $\{x_1, x_2, \ldots, x_{i-1}\} \subseteq \mathrm{var}(\alpha)$. For any alphabets A, B, a *morphism* is a function $h : A^* \to B^*$ that satisfies $h(vw) = h(v)h(w)$ for all $v, w \in A^*$. A morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ is called a *substitution* if $h(a) = a$ for every $a \in \Sigma$. For an arbitrary class of languages $\mathfrak{L}$ and a pattern α with $|\mathrm{var}(\alpha)| = m$, an *$\mathfrak{L}$-type for α* is a tuple $\mathcal{T} := (T_{x_1}, T_{x_2}, \ldots, T_{x_m})$, where, for every i, $1 \leq i \leq m$, $T_{x_i} \in \mathfrak{L}$ and T_{x_i} is called the *type language of (variable) x_i*. A substitution h *satisfies* $\mathcal{T}$ if and only if, for every i, $1 \leq i \leq m$, $h(x_i) \in T_{x_i}$.

Definition 1. *Let $\alpha \in \mathrm{PAT}$, let $\mathfrak{L}$ be a class of languages and let $\mathcal{T}$ be an $\mathfrak{L}$-type for α. The* $\mathcal{T}$-typed pattern language of α *is defined by $\mathcal{L}_{\mathcal{T}}(\alpha) := \{h(\alpha) \mid h$ is a substitution that satisfies $\mathcal{T}\}$. For any class of languages $\mathfrak{L}$, $\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT}) := \{\mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \in \mathrm{PAT}, \mathcal{T}$ is an $\mathfrak{L}$-type for $\alpha\}$ is the* class of $\mathfrak{L}$-typed pattern languages.

We note that $\{\Sigma^*\}$-typed and $\{\Sigma^+\}$-typed pattern languages correspond to the classes of E-pattern languages and NE-pattern languages, respectively, as defined by Angluin [3] and Shinohara [14]. It is easy to see that $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$ is contained in the class of REGEX languages. The substantial difference between these two classes is that the backreferences of a REGEX can refer to subexpressions that are not classical regular expressions, but REGEX. Hence, in order to describe larger classes of REGEX languages, the next step could be to type the variables of patterns with languages from $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$ instead of REG and then using the thus obtained languages again as type languages and so on. However, this approach leads to a dead end:

Proposition 2. *For any class of languages $\mathfrak{L}$, $\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT}) = \mathcal{L}_{\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT})}(\mathrm{PAT})$.*

The statement of the above proposition can be easily concluded from the observation that if a variable x of a pattern α is typed by a typed pattern language L, then we can as well substitute every occurrence of x in α by β, where β is the pattern that describes L.

Proposition 2 demonstrates that typed pattern languages are invariant with respect to iteratively typing the variables of the patterns, which suggests that in order to boost the expressive power of typed pattern languages, the regular aspect cannot completely be limited to the type languages of the variables. This observation brings us to the definition of $\mathrm{PAT}_{\mathrm{ro}} := \{\alpha \mid \alpha$ is a regular expression over $(\Sigma \cup X')$, where X' is a finite subset of $X\}$, the set of *patterns with regular operators.* In order to define the language given by a pattern with regular operators, we extend the definition of types to patterns with regular operators in the obvious way.

Definition 3. *Let $\alpha \in \mathrm{PAT}_{\mathrm{ro}}$ and let $\mathcal{T}$ be a type for α. The $\mathcal{T}$*-typed pattern language of α *is defined by $\mathcal{L}_{\mathcal{T}}(\alpha) := \bigcup_{\beta \in \mathcal{L}(\alpha)} \mathcal{L}_{\mathcal{T}}(\beta)$. For any class of languages $\mathfrak{L}$, we define $\mathcal{L}_{\mathfrak{L}}(\mathrm{PAT}_{\mathrm{ro}}) := \{\mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \in \mathrm{PAT}_{\mathrm{ro}}, \mathcal{T}$ is an $\mathfrak{L}$-type for $\alpha\}$.*

Patterns with regular operators are also used in the definition of pattern expressions (see [6] and Section 4) and have been called *regular patterns* in [4]. We can show that REG-typed patterns with regular operators are strictly more powerful than REG-typed patterns without regular operators

Proposition 4. $\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}_{\mathrm{ro}})$.

Proof. The inclusions follow by definition and we only have to show that they are proper. That $\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$ holds follows from the fact that all finite languages of cardinality at least two are in $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$ but not in $\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT})$. In order to show $\mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}_{\mathrm{ro}})$, we define $\alpha := (x_1\, \mathtt{c}\, x_1 \mid \varepsilon) \in \mathrm{PAT}_{\mathrm{ro}}$ and $\mathcal{T} := (\mathtt{a}^+)$. Clearly, $\mathcal{L}_{\mathcal{T}}(\alpha) = (\{\mathtt{a}^n\, \mathtt{c}\, \mathtt{a}^n \mid n \in \mathbb{N}\} \cup \{\varepsilon\}) \in \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}_{\mathrm{ro}})$. We now assume that there exists a pattern β and a REG-type $\mathcal{T}_r := (T_{x_1}, T_{x_2}, \ldots, T_{x_m})$ for β such that $\mathcal{L}_{\mathcal{T}_r}(\beta) = \mathcal{L}_{\mathcal{T}}(\alpha)$. Without loss of generality, we can assume that, for every i, $1 \leq i \leq m$, $T_{x_i} \neq \{\varepsilon\}$. Since $\varepsilon \in \mathcal{L}_{\mathcal{T}_r}(\beta)$, for every i, $1 \leq i \leq m$, $\varepsilon \in T_{x_i}$ and $\beta \in X^+$. Furthermore, if there exists an i, $1 \leq i \leq m$, such that T_{x_i} contains a non-empty word u without an occurrence of $\mathtt{c}$, then, by substituting every variable in β by ε expect x_i, which is substituted by u, we can obtain a non-empty word without an occurrence of $\mathtt{c}$, which is a contradiction. Thus, for every i, $1 \leq i \leq m$, and for every non-empty $u \in T_{x_i}$, there is exactly one occurrence of $\mathtt{c}$ in u. This implies that if there is more than one occurrence of a variable in β, then we can produce a word with at least two occurrences of $\mathtt{c}$. Thus, $\beta = x_1$ holds. Now, since we assume $\mathcal{L}_{\mathcal{T}_r}(\beta) = \mathcal{L}_{\mathcal{T}}(\alpha)$, it follows that $T_{x_1} = \mathcal{L}_{\mathcal{T}}(\alpha)$, which is a contradiction, since $\mathcal{L}_{\mathcal{T}}(\alpha)$ is not a regular language. Thus, $\mathcal{L}_{\mathcal{T}}(\alpha) \notin \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT})$. □

The invariance of typed patterns – represented by Proposition 2 – does not hold anymore with respect to patterns with regular operators. Before we formally prove this claim, we shall define an infinite hierarchy of classes of languages given by typed patterns with regular operators.

Definition 5. *Let $\mathfrak{L}_{\mathrm{ro},0} := \mathrm{REG}$ and, for every $i \in \mathbb{N}$, we define $\mathfrak{L}_{\mathrm{ro},i} := \mathcal{L}_{\mathfrak{L}_{\mathrm{ro},i-1}}(\mathrm{PAT}_{\mathrm{ro}})$. Furthermore, we define $\mathfrak{L}_{\mathrm{ro},\infty} = \bigcup_{i=0}^{\infty} \mathfrak{L}_{\mathrm{ro},i}$.*

It follows by definition, that the classes $\mathfrak{L}_{\text{ro},i}$, $i \in \mathbb{N}_0$, form a hierarchy and we strongly conjecture that it is proper. However, here we only separate the first three levels of that hierarchy.

Theorem 6. $\mathfrak{L}_{\text{ro},0} \subset \mathfrak{L}_{\text{ro},1} \subset \mathfrak{L}_{\text{ro},2} \subseteq \mathfrak{L}_{\text{ro},3} \subseteq \mathfrak{L}_{\text{ro},4} \subseteq \ldots$.

Proof. The inclusions follow by definition and $\mathfrak{L}_{\text{ro},0} \subset \mathfrak{L}_{\text{ro},1}$ obviously holds. In order to prove $\mathfrak{L}_{\text{ro},1} \subset \mathfrak{L}_{\text{ro},2}$, we define $L := \{(\mathtt{a}^n\mathtt{c}\mathtt{a}^n)^m\mathtt{d}(\mathtt{a}^n\mathtt{c}\mathtt{a}^n)^m \mid n, m \in \mathbb{N}\}$ and first note that $\mathcal{L}_{(L_1)}(x_1\,\mathtt{d}\,x_1) = L$, where $L_1 := \mathcal{L}_{(\mathcal{L}(\mathtt{a}^+))}((x_1\,\mathtt{c}\,x_1)^+)$, which shows that $L \in \mathfrak{L}_{\text{ro},2}$. We now assume that $L \in \mathfrak{L}_{\text{ro},1}$ and show that this assumption leads to a contradiction. If $L \in \mathfrak{L}_{\text{ro},1}$, then there exists a pattern with regular operators α and a regular type $\mathcal{T} := (T_{x_1}, T_{x_2}, \ldots, T_{x_m})$ for α such that $\mathcal{L}_{\mathcal{T}}(\alpha) = L$. If $\mathcal{L}(\alpha)$ is finite, then there must exist at least one variable x such that there exists a word in T_x containing a factor $\mathtt{c}\,\mathtt{a}^{n'}\,\mathtt{c}\,\mathtt{a}^{n'}\,\mathtt{c}$, where n' is greater than the constant of the pumping lemma 4.2 on page 85 of [16] with respect to the regular language T_x. By applying the pumping lemma, we can show that in T_x there exists a word containing a factor $\mathtt{c}\,\mathtt{a}^{m}\,\mathtt{c}\,\mathtt{a}^{m'}\,\mathtt{c}$, $m \neq m'$, which is a contradiction. Next, we assume that $\mathcal{L}(\alpha)$ is infinite and, without loss of generality, we further assume that α does not contain any terminal symbols and, for every i, $1 \leq i \leq m$, $T_{x_i} \neq \{\varepsilon\}$. Since every word of L contains exactly one occurrence of $\mathtt{d}$, we can conclude that in α there are variables $y_1, y_2, \ldots, y_l$, $l \in \mathbb{N}$, such that, for every i, $1 \leq i \leq l$, T_{y_i} contains at least one word with exactly one occurrence of $\mathtt{d}$. Furthermore, for every $\beta \in \mathcal{L}(\alpha)$, there exists a j, $1 \leq j \leq l$, such that $\beta = \delta\, y_j\, \gamma$ and $\text{var}(\delta\,\gamma) \cap \{y_1, y_2, \ldots, y_l\} = \emptyset$. Since $\mathcal{L}(\alpha)$ is infinite, for some j, $1 \leq j \leq l$, there exists a word $\delta\, y_j\, \gamma$ in $\mathcal{L}(\alpha)$ such that $|\delta| > k$ or $|\gamma| > k$, where k is the pumping lemma constant with respect to the regular language $\mathcal{L}(\alpha)$. This implies that δ (or γ, respectively) can be arbitrarily pumped and, since every type language contains at least one non-empty word, this implies that there is a word in $\mathcal{L}_{\mathcal{T}}(\alpha)$ of form $u\,\mathtt{d}\,v$ with $|u| > |v|$ (or $|u| < |v|$, respectively), which is a contradiction. This shows that in fact $L \notin \mathfrak{L}_{\text{ro},1}$. $\square$

In the following section, we shall show that the class $\mathfrak{L}_{\text{ro},\infty}$ coincides with the class of languages that are defined by the already mentioned pattern expressions and we formally prove it to be a proper subset of the class of REGEX languages.

4. Pattern Expressions

We define pattern expressions as introduced by Câmpeanu and Yu [6], but we use a slightly different notation. A *pattern expression* is a tuple $(x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$, where, for every i, $1 \leq i \leq n$, $r_i \in \text{PAT}_{\text{ro}}$ and $\text{var}(r_i) \subseteq \{x_1, x_2, \ldots, x_{i-1}\}$. The set of all pattern expressions is denoted by PE. Next, we recall how pattern expressions describe formal languages as defined by Câmpeanu and Yu [6].

Definition 7. *Let* $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n) \in \text{PE}$. *We define* $L_{p,x_1} := \mathcal{L}(r_1)$ *and, for every* i, $2 \leq i \leq n$, $L_{p,x_i} := \mathcal{L}_{\mathcal{T}_i}(r_i)$, *where* $\mathcal{T}_i :=$

$(L_{p,x_1}, L_{p,x_2}, \ldots, L_{p,x_{i-1}})$ *is a type for* r_i. *The* language generated by p with respect to iterated substitution *is defined by* $\mathcal{L}_{\text{it}}(p) := L_{p,x_n}$ *and* $\mathcal{L}_{\text{it}}(\text{PE}) := \{\mathcal{L}_{\text{it}}(p) \mid p \in \text{PE}\}$.

We illustrate the above definition with the following example pattern expression: $q := (x_1 \to \mathtt{a}^*, x_2 \to x_1(\mathtt{c} \mid \mathtt{d})x_1, x_3 \to x_1\mathtt{c}x_2)$. By definition, $\mathcal{L}_{\text{it}}(q) = \{\mathtt{a}^k\mathtt{ca}^m u\mathtt{a}^m \mid k, m \in \mathbb{N}_0, u \in \{\mathtt{c}, \mathtt{d}\}\}$. We particularly note that Definition 7 implies that occurrences of the same variable in different elements of the pattern expression do not need to be substituted by the same word and we shall later see that this behaviour essentially limits the expressive power of pattern expressions.

As mentioned before, the class of languages described by pattern expressions with respect to iterated substitution coincides with the class $\mathfrak{L}_{\text{ro},\infty}$.

Theorem 8. $\mathfrak{L}_{\text{ro},\infty} = \mathcal{L}_{\text{it}}(\text{PE})$.

Proof. We first prove $\mathcal{L}_{\text{it}}(\text{PE}) \subseteq \mathfrak{L}_{\text{ro},\infty}$. Let $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$ be a pattern expression and, for every i, $1 \leq i \leq n$, let the languages L_{p,x_i} be defined as in Definition 7. Next, we assume that for some i, $2 \leq i \leq n$, and for every j, $1 \leq j < i$, $L_{p,x_j} \in \mathfrak{L}_{\text{ro},j-1}$. This implies that $\mathcal{T} := (L_{p,x_1}, L_{p,x_2}, \ldots, L_{p,x_{i-1}})$ is an $\mathfrak{L}_{\text{ro},i-2}$-type for r_i. Thus, $\mathcal{L}_{\mathcal{T}}(r_i) = L_{p,x_i} \in \mathfrak{L}_{\text{ro},i-1}$. Since $L_{p,x_1} \in \mathfrak{L}_{\text{ro},0}$ obviously holds, we can conclude by induction that, for every i, $1 \leq i \leq n$, $L_{p,x_i} \in \mathfrak{L}_{\text{ro},i-1}$. In particular, $\mathcal{L}_{\text{it}}(p) \in \mathfrak{L}_{\text{ro},\infty}$; thus, $\mathcal{L}_{\text{it}}(\text{PE}) \subseteq \mathfrak{L}_{\text{ro},\infty}$ follows.

It remains to show $\mathfrak{L}_{\text{ro},\infty} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$. To this end, we first note that, for every pattern α with regular operators and for every $\mathcal{L}_{\text{it}}(\text{PE})$-type $\mathcal{T} := (T_{x_1}, T_{x_2}, \ldots, T_{x_n})$ for α, $\mathcal{L}_{\mathcal{T}}(\alpha) \in \mathcal{L}_{\text{it}}(\text{PE})$ holds. This is due to the fact that α and all the pattern expressions which describe the type languages T_{x_i}, $1 \leq i \leq n$, can be merged to a pattern expression p with $\mathcal{L}_{\text{it}}(p) = \mathcal{L}_{\mathcal{T}}(\alpha)$. Since $\mathfrak{L}_{\text{ro},0} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$, we can conclude by induction that $\mathfrak{L}_{\text{ro},\infty} \subseteq \mathcal{L}_{\text{it}}(\text{PE})$. □

In the following, we define an alternative way of how pattern expressions can describe languages, i. e., instead of substituting the variables by words in an iterative way, we substitute them uniformly.

Definition 9. *Let* $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n) \in \text{PE}$. *A word* $w \in \Sigma^*$ *is in the* language generated by p with respect to uniform substitution *(*$\mathcal{L}_{\text{uni}}(p)$*, for short) if and only if there exists a substitution* h *such that* $h(x_n) = w$ *and, for every* i*,* $1 \leq i \leq n$*, there exists an* $\alpha_i \in \mathcal{L}(r_i)$ *with* $h(x_i) = h(\alpha_i)$.

It can be verified that, by Definition 9, $\mathcal{L}_{\text{uni}}(q) = \{\mathtt{a}^n\mathtt{ca}^n u\mathtt{a}^n \mid n \in \mathbb{N}_0, u \in \{\mathtt{c}, \mathtt{d}\}\}$ holds, where q is the pattern expression defined above. For an arbitrary pattern expression $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$, the language $\mathcal{L}_{\text{uni}}(p)$ can also be defined in a more constructive way. We first choose a word $u \in \mathcal{L}(r_1)$ and, for all i, $1 \leq i \leq n$, if variable x_1 occurs in r_i, then we substitute all occurrences of x_1 in r_i by u. Then we delete the element $x_1 \to r_1$ from the pattern expression. If we repeat this step with respect to variables $x_2, x_3, \ldots, x_{n-1}$, then we obtain a pattern expression of form $(x_n \rightarrow r'_n)$, where r'_n is a regular expression over Σ. The language $\mathcal{L}_{\text{uni}}(p)$ is the union of the languages given by all these regular expressions. The language $\mathcal{L}_{\text{it}}(q)$ can be defined similarly. We first choose a word $u_1 \in \mathcal{L}(r_1)$ and then we substitute all occurrences of x_1 in r_2 by u_1. After that, we choose a *new* word $u_2 \in \mathcal{L}(r_1)$ and substitute all occurrences of x_1 in r_3 by u_2 and so on until there are no more occurrences of variable x_1 in q and then we delete the element $x_1 \rightarrow r_1$. Then this step is repeated with respect to $x_2, x_3, \ldots, x_{n-1}$.

From the above considerations, we can directly conclude the following.

Proposition 10. *Let $p := (x_1 \rightarrow r_1, x_2 \rightarrow r_2, \ldots, x_m \rightarrow r_m)$ be a pattern expression. Then $\mathcal{L}_{\text{uni}}(p) \subseteq \mathcal{L}_{\text{it}}(p)$ and if, for every i, j, $1 \leq i < j \leq m$, $\text{var}(r_i) \cap \text{var}(r_j) = \emptyset$, then also $\mathcal{L}_{\text{it}}(p) \subseteq \mathcal{L}_{\text{uni}}(p)$.*

The more interesting question is whether or not the iterative and the uniform way of defining pattern expression languages are substantially different in terms of expressive power. Intuitively, for any pattern expression p, it is not essential for the language $\mathcal{L}_{\text{it}}(p)$ that there exist occurrences of the same variable in different elements of p and it is possible to transform p into an equivalent pattern expression p', the elements of which have disjoint sets of variables and, thus, by Proposition 10, $\mathcal{L}_{\text{it}}(p) = \mathcal{L}_{\text{uni}}(p')$, which implies $\mathcal{L}_{\text{it}}(\text{PE}) \subseteq \mathcal{L}_{\text{uni}}(\text{PE})$. Hence, for the language generated by a pattern expression with respect to iterated substitution, the possibility of using the same variables in different elements of a pattern expression can be considered as mere syntactic sugar that keeps pattern expressions concise. The more difficult question whether or not pattern expressions, equipped with uniform substitution, can define languages that cannot be described by any pattern expression with respect to iterated substitution, is answered in the positive by the following lemma.

Lemma 11. *There exists a language $L \in \mathcal{L}_{\text{uni}}(\text{PE})$ with $L \notin \mathcal{L}_{\text{it}}(\text{PE})$.*

Proof. We define a pattern expression $p := (x_1 \rightarrow \mathtt{a}^+, x_2 \rightarrow (x_1\,\mathtt{c})^+, x_3 \rightarrow x_2\,\mathtt{b}\,x_1\,\mathtt{d}\,x_2)$ and note that $\mathcal{L}_{\text{uni}}(p) = \{(\mathtt{a}^n\,\mathtt{c})^m\,\mathtt{b}\,\mathtt{a}^n\,\mathtt{d}\,(\mathtt{a}^n\,\mathtt{c})^m \mid n, m \in \mathbb{N}\}$. In order to conclude the statement of the lemma, it remains to show that $\mathcal{L}_{\text{uni}}(p) \notin \mathcal{L}_{\text{it}}(\text{PE})$. To this end, we first prove the following claim:

Claim. Let q be a pattern expression. There exists a pattern expression $q' := (x_1 \rightarrow t'_1, x_2 \rightarrow t'_2, \ldots, x_{m'} \rightarrow t'_{m'})$, such that, for every i, $1 \leq i \leq m' - 1$, $\mathcal{L}(t'_i)$ is infinite and $\mathcal{L}_{\text{it}}(q) = \mathcal{L}_{\text{it}}(q')$.

Proof (Claim). Let $q := (x_1 \rightarrow t_1, x_2 \rightarrow t_2, \ldots, x_m \rightarrow t_m)$ be a pattern expression. We assume that for some l, $1 \leq l \leq m - 1$, $\mathcal{L}(t_l) := \{\beta_1, \beta_2, \ldots, \beta_k\}$ is finite and, for every i, $1 \leq i < l$, $\mathcal{L}(t_i)$ is infinite. We transform q into a pattern expression $q'' := (x_1 \rightarrow t''_1, x_2 \rightarrow t''_2, \ldots, x_{l-1} \rightarrow t''_{l-1}, x_{l+1} \rightarrow t''_{l+1}, \ldots, x_m \rightarrow t''_m)$ in the following way. For every i, $l+1 \leq i \leq m$, we define $t''_i := (t_{i,1} \mid t_{i,2} \mid \ldots \mid t_{i,k})$, where, for every j, $1 \leq j \leq k$, $t_{i,j}$ is obtained from t_i by substituting every occurrence of

x_l by β_j and, for every i, $1 \leq i \leq l-1$, we define $t''_i := t_i$. It is straightforward to see that $\mathcal{L}_{\text{it}}(q) = \mathcal{L}_{\text{it}}(q'')$ and, by repeating this procedure, q can be transformed into $q' = (x_1 \to t'_1, x_2 \to t'_2, \ldots, x_{m'} \to t'_{m'})$, where, for every i, $1 \leq i \leq m'-1$, $\mathcal{L}(t'_i)$ is infinite and $\mathcal{L}_{\text{it}}(q) = \mathcal{L}_{\text{it}}(q')$. *q.e.d. (Claim)*

We now assume contrary to the statement of the lemma, that there exists a pattern expression $p' := (x_1 \to r_1, x_2 \to r_2, \ldots, x_m \to r_m)$ with $\mathcal{L}_{\text{it}}(p') = \mathcal{L}_{\text{uni}}(p)$, which shall lead to a contradiction. For every i, $1 \leq i \leq m$, let L_{p',x_i} be the language as defined in Definition 7. By the above claim, we can also assume that, for every i, $1 \leq i \leq m-1$, $\mathcal{L}(r_i)$ is infinite. Next, we note that, for every i, $1 \leq i \leq m$, if there is a word in L_{p',x_i} with an occurrence of $\mathtt{b}$ or $\mathtt{d}$, then, for every $w \in L_{p',x_i}$, $|w|_{\mathtt{b}} = 1$ ($|w|_{\mathtt{d}} = 1$, respectively) holds. This follows directly from the fact that, for every $w' \in \mathcal{L}_{\text{uni}}(p)$, $|w'|_{\mathtt{b}} = |w'|_{\mathtt{d}} = 1$ is satisfied.

Next, we assume that for some l, $1 \leq l \leq m$, L_{p',x_l} contains a word with an occurrence of $\mathtt{b}$ or $\mathtt{d}$ and $\mathcal{L}(r_l)$ is infinite. If L_{p',x_l} contains a word with an occurrence of $\mathtt{b}$, then, as pointed out above, all the words of L_{p',x_l} contain exactly one occurrence of $\mathtt{b}$, which implies that, for every $\beta \in \mathcal{L}(r_l)$, $\beta = \gamma\, z\, \gamma'$, where either $z = \mathtt{b}$ or $z = x_j$, $1 \leq j < l$, such that all the words of L_{p',x_j} contain exactly one occurrence of $\mathtt{b}$. Moreover, since $\mathcal{L}(r_l)$ is infinite, we can assume that $|\gamma|$ or $|\gamma'|$ exceeds the pumping lemma constant for the regular language $\mathcal{L}(r_l)$. Consequently, by applying the pumping lemma, we can produce a word $\widehat{\gamma}\, z\, \gamma' \in \mathcal{L}(r_l)$ with $|\gamma| < |\widehat{\gamma}|$ or a word $\gamma\, z\, \widehat{\gamma} \in \mathcal{L}(r_l)$ with $|\gamma'| < |\widehat{\gamma}|$, respectively. Since, without loss of generality, we can assume that, for every i, $1 \leq i \leq m$, $L_{p,x_i} \neq \{\varepsilon\}$, this directly implies that there exists a word $w \in \mathcal{L}_{\text{it}}(p')$ that is of form $w = u\,\mathtt{b}\,v$, where it is not satisfied that there exist $n, m \in \mathbb{N}$ with $|u| = (n+1)\,m$ and $|v| = (n+1)\,m + n + 1$, which is a contradiction. If L_{p',x_l} contains a word with an occurrence of $\mathtt{d}$ and $\mathcal{L}(r_l)$ is infinite, then we can obtain a contradiction in an analogous way. Consequently, if any L_{p',x_i}, $1 \leq i \leq m$, contains a word with an occurrence of $\mathtt{b}$ or $\mathtt{d}$, then $\mathcal{L}(r_i)$ is finite. This particularly implies that L_{p',x_m} is finite, since it contains words with $\mathtt{b}$ and $\mathtt{d}$, and, for every i, $1 \leq i \leq m-1$, since $\mathcal{L}(r_i)$ is infinite, L_{p',x_i} does not contain a word with an occurrence of $\mathtt{b}$ or $\mathtt{d}$. Hence, without loss of generality, we can assume that $r_m := (\beta_1 \mid \beta_2 \mid \ldots \mid \beta_k)$ with $\beta_i := \gamma_i\, \mathtt{b}\, \gamma'_i\, \mathtt{d}\, \gamma''_i \in \text{PAT}$, $1 \leq i \leq k$. Next, for every i, $1 \leq i \leq k$, and for every j, $1 \leq j \leq m-1$, we define $\widehat{L}_i := \mathcal{L}_{(L_{p',x_1}, \ldots, L_{p',x_{m-1}})}(\beta_i)$. Obviously, $\mathcal{L}_{\text{it}}(p') = \widehat{L}_1 \cup \widehat{L}_2 \cup \ldots \cup \widehat{L}_k$. This implies that there must exist at least one l, $1 \leq l \leq k$, such that, for every $n \in \mathbb{N}$, there exists a word $w \in \widehat{L}_l$ with $|w|_{\mathtt{c}} > n$ and a word $w' \in \widehat{L}_l$ with $w' = u\, \mathtt{b}\, \mathtt{a}^{n'}\, \mathtt{d}\, v$, for some n', $n < n'$. Thus, in γ_l there must occur a variable x_j such that the number of occurrences of $\mathtt{c}$ is unbounded in L_{p',x_j}. Moreover, we can assume that there is also an occurrence of x_j in γ''_l, since otherwise there would be a word in $\widehat{L}_l$ with a different number of occurrences of $\mathtt{c}$ to the left of $\mathtt{b}$ than to the right of $\mathtt{d}$. Similarly, in γ'_l there must occur a variable $x_{j'}$ such that $L_{p',x_{j'}}$ is an infinite unary language over $\{\mathtt{a}\}$, which particularly implies that $j \neq j'$. We note further that in L_{p',x_j}, there exists a word u with a factor $\mathtt{c}\,\mathtt{a}^n\,\mathtt{c}$, for some $n \in \mathbb{N}$. We can now obtain β'_l by substituting every occurrence of x_j in β_l by u. Next, we obtain β''_l from β'_l by substituting every occurrence of $x_{j'}$ by a word $\mathtt{a}^{n'}$ with $n < n'$. Next, for every i, $1 \leq i \leq m-1$, we substitute all occurrences of variable x_i in β''_l by some word from L_{p',x_i}. The thus constructed word is in $\widehat{L}_l$, but not in $\mathcal{L}_{\text{uni}}(p)$, since it contains both a factor $\mathtt{c}\,\mathtt{a}^n\,\mathtt{c}$ and $\mathtt{b}\,\mathtt{a}^{n''}\,\mathtt{d}$ with $n < n' \leq n''$. This is a contradiction. □

From Lemma 11, we can conclude that pattern expressions equipped with uniform substitution are more powerful than their iterative version.

Theorem 12. $\mathcal{L}_{\text{it}}(\text{PE}) \subset \mathcal{L}_{\text{uni}}(\text{PE})$.

We conclude this section by mentioning that by Bordihn et al. in [4], it has been shown that $\mathcal{H}^*(\text{REG}, \text{REG})$, a class of languages given by an iterated version of H-systems (see Albert and Wegner [2] and Bordihn et al. [4]), also coincides with $\mathcal{L}_{\text{it}}(\text{PE})$, which implies $\mathfrak{L}_{\text{ro},\infty} = \mathcal{L}_{\text{it}}(\text{PE}) = \mathcal{H}^*(\text{REG}, \text{REG}) \subset \mathcal{L}_{\text{uni}}(\text{PE})$.

In the following section, we compare $\mathcal{L}_{\text{uni}}(\text{PE})$ to the class of REGEX languages.

5. REGEX

A REGEX is a regular expression, the subexpressions of which can be numbered by adding an integer index to the parentheses delimiting the subexpression (i. e., $(_n \ \ldots \ )_n$, $n \in \mathbb{N}$). This is done in such a way that there are no two different subexpressions with the same number. The subexpression that is numbered by $n \in \mathbb{N}$, which is called the n^{th} *referenced subexpression*, can be followed by arbitrarily many *backreferences* to that subexpression, denoted by $\backslash n$. A formal definition of the language described by a REGEX can be found in [5]. Here, we stick to the more informal definition which has already been briefly outlined in Section 1 and that we now recall in a bit more detail. Let r be an arbitrary REGEX. The language described by r, which is denoted by $\mathcal{L}(r)$, is defined in a similar way as for classical regular expressions. The only aspect that needs to be explained in more detail is the role of referenced subexpressions and backreferences. A word w is in $\mathcal{L}(r)$ if and only if we can obtain it from r in the following way. We move over r from left to right and produce a word as it is done for a classical regular expression. When we encounter the i^{th} referenced subexpression, then we store the factor u_i that is matched to it and from now on we treat every occurrence of $\backslash i$ as u_i. When we encounter the i^{th} referenced subexpression for a second time, which is possible since the i^{th} referenced subexpression may occur under a star, then we overwrite u_i with the possible new factor that is now matched to the i^{th} referenced subexpression. This entails the late binding of backreferences, which has been described in Section 1. If a backreference $\backslash i$ occurs and there is no factor u_i stored that has been matched to the i^{th} referenced subexpression, then $\backslash i$ is interpreted as the empty word.

We also define an alternative way of how a REGEX describes a language, that shall be useful for our proofs. The *language with necessarily initialised subexpres-*

sions of a REGEX r, denoted by $\mathcal{L}_{\text{nis}}(r)$, is defined in a similar way as $\mathcal{L}(r)$ above, but if a backreference $\backslash i$ occurs and there is currently no factor u_i stored that has been matched to the i^{th} referenced subexpression, then instead of treating $\backslash i$ as the empty word, we interpret it as the i^{th} referenced subexpression, we store the factor u_i that is matched to it and from now on every occurrence of $\backslash i$ is treated as u_i. For example, let $r := ((_1\, \mathtt{a}^*\,)_1 \mid \varepsilon)\, \mathtt{b} \backslash 1\, \mathtt{b} \backslash 1$. Then $\mathcal{L}(r) := \{\mathtt{a}^n\mathtt{b}\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}_0\}$ and $\mathcal{L}_{\text{nis}}(r) := \mathcal{L}(r) \cup \{\mathtt{b}\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}_0\}$.

We can note that the late binding of backreferences as well as non-initialised referenced subexpressions is caused by referenced subexpression under a star or in an alternation. Next, we define REGEX that are restricted in this regard. A REGEX is *alternation confined* if and only if all backreferences occur in the same option of the same alternation as the corresponding referenced subexpression. A REGEX is *star-free initialised* if and only if no referenced subexpression occurs under a star. Let REGEX_{ac} and $\text{REGEX}_{\text{sfi}}$ be the sets of REGEX that are alternation confined and star-free initialised, respectively, and let $\text{REGEX}_{\text{sfi,ac}} := \text{REGEX}_{\text{ac}} \cap \text{REGEX}_{\text{sfi}}$. We can show that the condition of being alternation confined does not impose a restriction on the expressive power of star-free initialised REGEX.

Lemma 13.
$\mathcal{L}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}(\text{REGEX}_{\text{sfi,ac}}) = \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$.

Proof. Since $\text{REGEX}_{\text{sfi,ac}} \subseteq \text{REGEX}_{\text{sfi}}$, $\mathcal{L}(\text{REGEX}_{\text{sfi,ac}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi}})$ and $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}})$ trivially holds. Moreover, for every $r \in \text{REGEX}_{\text{sfi,ac}}$, $\mathcal{L}_{\text{nis}}(r) = \mathcal{L}(r)$ holds, since in this case it is impossible that, while matching r to some word, a backreference occurs that points to a referenced subexpression that has not been initialised. Thus, $\mathcal{L}(\text{REGEX}_{\text{sfi,ac}}) = \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$. In order to conclude the proof, it is sufficient to show that $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi,ac}})$ and $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$.

We first prove that $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi,ac}})$. To this end, let r be a star-free initialised REGEX that is not alternation confined, which implies that r contains an alternation $(r_1 \mid r_2)$, where in r_1 or r_2 there occurs a referenced subexpression that is referenced outside of r_1 or r_2, respectively. We now obtain t_1 from r by substituting $(r_1 \mid r_2)$ by r_1 and by deleting all backreferences that point to a referenced subexpression in r_2. In a similar way, we obtain t_2 from r by substituting $(r_1 \mid r_2)$ by r_2 and by deleting all backreferences that point to a referenced subexpression in r_1. Next, we transform t_1 and t_2 in t'_1 and t'_2 by renaming all referenced subexpressions and their corresponding backreferences such that in t'_1 and t'_2 there are no referenced subexpressions that are numbered by the same number. We note that $r' := (t'_1 \mid t'_2)$ is a star-free initialised REGEX that satisfies $\mathcal{L}(r) = \mathcal{L}(r')$. By repeating the above construction, we can transform r into a REGEX r'' that is star-free initialised, alternation confined and which satisfies $\mathcal{L}(r'') = \mathcal{L}(r)$. Thus, $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi,ac}})$ is implied.

Next, we prove $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$. Again, let r be a star-free initialised REGEX that is not alternation confined, which implies that $r := r_1\,(r_2 \mid r_3)\,r_4$, where in r_2 or r_3 there occurs a referenced subexpression that is referenced outside of r_2 or r_3, respectively. Without loss of generality, we assume that $(_1\ t_1\ )_1, (_2\ t_2\ )_2, \ldots, (_k\ t_k\ )_k$ are exactly the referenced subexpressions in r_3 and we assume them to be ordered with respect to their nesting, i. e., for every i, j, $1 \leq i < j \leq k$, t_i does not occur in t_j. We now obtain s'_1 from $r_1\,r_2\,r_4$ in the following way. We first substitute the leftmost occurrence of $\backslash 1$ by $(_1\ t_1\ )_1$. Next, if there does not already exist an occurrence of $(_2\ t_2\ )_2$ (which might be the case if $(_2\ t_2\ )_2$ is contained in t_1), then we substitute the leftmost occurrence of $\backslash 2$ by $(_2\ t_2\ )_2$. This step is then repeated with respect to the referenced subexpressions $(_3\ t_3\ )_3, \ldots, (_k\ t_k\ )_k$. We observe that, for every i, $1 \leq i \leq k$, there is at most one occurrence of $(_i\ t_i\ )_i$ in s'_1 and if there exists a backreference $\backslash i$, then it occurs to the right of $(_i\ t_i\ )_i$. This implies that s'_1 is a valid REGEX. Next, we transform $r_1\,r_3\,r_4$ into s'_2 in the same way, just with respect to the referenced subexpressions in r_2. Finally, s_1 and s_2 are obtained from s'_1 and s'_2, respectively, by renaming all referenced subexpressions and the corresponding backreferences in such a way that s_1 and s_2 do not have any referenced subexpressions labeled by the same number. We define $r' := (s_1 \mid s_2)$ and note that r' is a valid star-free initialised REGEX with $\mathcal{L}_{\text{nis}}(r') = \mathcal{L}_{\text{nis}}(r)$. By successively applying the above transformation now on s and t and so on, r can be transformed into a star-free initialised REGEX r'' that is also alternation confined and $\mathcal{L}_{\text{nis}}(r'') = \mathcal{L}_{\text{nis}}(r)$. This proves $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi,ac}})$. □

In the following, we take a closer look at the task of transforming a pattern expression p into a REGEX r, such that $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}(r)$. Although, this is possible in general, a few difficulties arise, that have already been pointed out by Câmpeanu and Yu in [6] (with respect to $\mathcal{L}_{\text{it}}(p)$).

The natural way to transform a pattern expression into an equivalent REGEX is to successively substitute the occurrences of variables by referenced subexpressions and appropriate backreferences. For example, we could simply transform $q := (x_1 \to (\mathtt{a} \mid \mathtt{b})^*, x_2 \to x_1^*\,\mathtt{c}\,x_1\,\mathtt{d}\,x_1)$ into $r_q := (_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1^*\,\mathtt{c}\,\backslash 1\,\mathtt{d}\,\backslash 1$, but then we obtain an incorrect REGEX. This is due to the fact that the referenced subexpression is under a star. To avoid this, we can first rewrite q to $q' := (x_1 \to (\mathtt{a} \mid \mathtt{b})^*, x_2 \to (x_1\,x_1^* \mid \varepsilon)\,\mathtt{c}\,x_1\,\mathtt{d}\,x_1)$, which leads to $r_{q'} := ((_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1\,(\backslash 1)^* \mid \varepsilon)\,\mathtt{c}\,\backslash 1\,\mathtt{d}\,\backslash 1$. Now we encounter a different problem: in $\mathcal{L}(r_{q'})$ the only word that starts with $\mathtt{c}$ is $\mathtt{cd}$, which is not the case for $\mathcal{L}_{\text{uni}}(q')$. However, we note that the language with necessarily initialised subexpressions of $r_{q'}$ is exactly what we want, since $\mathcal{L}_{\text{nis}}(r_{q'}) = \mathcal{L}_{\text{uni}}(q)$. Hence, we can transform any pattern expression p to a REGEX r_p that is star-free initialised and satisfies $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}_{\text{nis}}(r_p)$.

Lemma 14. *For every pattern expression p, there exists a star-free initialised* REGEX *r with $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}_{\text{nis}}(r)$.*

Proof. Let $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_m \to r_m)$ be an arbitrary pattern expression. First, for every i, $1 \leq i \leq m$, if r_i contains a subexpression $(q)^*$, where $(q)^*$ is not under a star and q contains the leftmost occurrence of a variable, we substitute $(q)^*$ by $(q\,(q)^* \mid \varepsilon)$ and we repeat this step until we obtain a pattern expression $p' := (x_1 \to r'_1, x_2 \to r'_2, \dots, x_m \to r'_m)$, where, for every i, $1 \leq i \leq m$, the leftmost occurrence of any variable in r'_i does not occur under a star and $\mathcal{L}_{\text{uni}}(p') = \mathcal{L}_{\text{uni}}(p)$.

Next, we construct a REGEX t with $\mathcal{L}_{\text{nis}}(t) = \mathcal{L}_{\text{uni}}(p')$ in the following way. First, we transform r'_m into t_{m-1} by substituting the leftmost occurrence of x_{m-1} by $(_{m-1}\; r'_{m-1}\;)_{m-1}$ and all other occurrences of x_{m-1} by $\backslash m-1$. In a next step, we obtain t_{m-2} from t_{m-1} by substituting the leftmost occurrence of x_{m-2} by $(_{m-2}\; r'_{m-2}\;)_{m-2}$ and all other occurrences of x_{m-2} by $\backslash m-2$. This procedure is now repeated until we obtain the valid REGEX t_1. Since, for every i, $2 \leq i \leq m$, the leftmost occurrence of any variable in r'_i does not occur under a star, we can conclude that t_1 is star-free initialised. For the sake of convenience, we shall call t_1 simply t. It remains to show that $\mathcal{L}_{\text{nis}}(t) = \mathcal{L}_{\text{uni}}(p')$ holds.

To this end, we assume that, for some i, $1 \leq i \leq m-2$, and for every j, $1 \leq j \leq i$, the set of words that can be matched to the j^{th} referenced subexpression in t corresponds to the language described by the pattern expression $(x_1 \to r'_1, x_2 \to r'_2, \dots, x_j \to r'_j)$. We note that by the definition of t, this implies that also the $(i+1)^{\text{th}}$ referenced subexpression in t corresponds to the language described by the pattern expression $(x_1 \to r'_1, x_2 \to r'_2, \dots, x_{i+1} \to r'_{i+1})$. Since the set of words that can be matched to the first referenced subexpression in t clearly corresponds to the language described by the pattern expression $(x_1 \to r'_1)$ it follows by induction that $\mathcal{L}_{\text{nis}}(t) = \mathcal{L}_{\text{uni}}(p')$. $\square$

We recall that, by Lemma 13, $\mathcal{L}_{\text{nis}}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}(\text{REGEX}_{\text{sfi}})$ holds. Thus, Lemmas 13 and 14 imply that every pattern expression can be transformed into an equivalent star-free initialised REGEX. For example, the pattern expression q introduced on page 12 can be transformed into the REGEX $t_q := ((_1\; (\mathtt{a} \mid \mathtt{b})^*\;)_1\, (\backslash 1)^*\, \mathtt{c}\, \backslash 1\, \mathtt{d}\, \backslash 1 \mid \mathtt{c}\, (_2\; (\mathtt{a} \mid \mathtt{b})^*\;)_2\, \mathtt{d}\, \backslash 2)$, which finally satisfies $\mathcal{L}_{\text{uni}}(q) = \mathcal{L}(t_q)$.

Theorem 15. *$\mathcal{L}_{\text{uni}}(\text{PE}) \subseteq \mathcal{L}(\text{REGEX}_{\text{sfi}})$.*

In the remainder of this section, we show that every star-free initialised REGEX r can be transformed into a pattern expression which is equivalent with respect to uniform substitution. However, as pointed out by the following example, this cannot be done directly if r is not alternation confined. The natural way of transforming the REGEX $r := ((_1\; (\mathtt{a} \mid \mathtt{b})^*\;)_1 \mid (_2\; \mathtt{c}^*\;)_2)\, (\backslash 1)^*\, \backslash 2$ into a pattern expression would lead to $p_r := (x_1 \to (\mathtt{a} \mid \mathtt{b}), x_2 \to \mathtt{c}^*, x_3 \to (x_1 \mid x_2)\, (x_1)^*\, x_2)$. We can observe, that every word in $\mathcal{L}(r)$ which starts with c does not contain any occurrence of a or b, whereas this is not the case for $\mathcal{L}_{\text{uni}}(p_r)$. So in order to transform star-free initialised REGEX into equivalent pattern expressions, we can again apply Lemma 13, which states that we can transform every star-free initialised REGEX into an equivalent one that is also alternation confined.

Theorem 16. $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{uni}}(\text{PE})$.

Proof. Let t be a star-free initialised REGEX. By Lemma 13, the REGEX t can be transformed into a star-free initialised and alternation confined REGEX r with $\mathcal{L}(t) = \mathcal{L}(r)$. Without loss of generality, we assume that there are k referenced subexpressions in r and that these are ordered with respect to their nesting, i. e., for every i, j, $1 \leq i < j \leq k$, the j^{th} referenced subexpression does not occur inside the i^{th} referenced subexpression. Next, we transform r into the tuple $(x_1 \to r_1, z \to s_1)$, where r_1 is the first referenced subexpression of r and s_1 is obtained from r by substituting $(_1\ r_1\ )_1$ and all occurrences of $\backslash 1$ by x_1. Next, we transform $(x_1 \to r_1, z \to s_1)$ into $(x_1 \to r_1, x_2 \to r_2, z \to s_2)$, where r_2 is the second referenced subexpression of r and s_2 is obtained from s_1 by substituting $(_2\ r_2\ )_2$ and all occurrences of $\backslash 2$ by x_2. This step is repeated until we obtain a pattern expression $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_k \to r_k, z \to s_k)$.

The tuples $q_i := (x_1 \to r_1, x_2 \to r_2, \ldots, x_i \to r_i, z \to s_i)$, $1 \leq i \leq k-1$, constructed in this procedure are no pattern expressions, since the element s_i is not a pattern with regular operators, but a REGEX. However, it is straightforward to interpret these q_i in a similar way as pattern expressions, i. e., $\mathcal{L}_{\text{uni}}(q_i)$ is the union of all $\mathcal{L}(s')$, where s' is a REGEX (without variables), that can be obtained from q_i by first substituting all occurrences of variable x_1 in q_i by a word $u \in \mathcal{L}(r_1)$ and so on. In particular, for every i, $1 \leq i \leq k-2$, $\mathcal{L}_{\text{uni}}(q_i) = \mathcal{L}_{\text{uni}}(q_{i+1})$ and, furthermore, $\mathcal{L}(r) = \mathcal{L}_{\text{uni}}(q_1)$ and $\mathcal{L}_{\text{uni}}(q_{k-1}) = \mathcal{L}_{\text{uni}}(p)$. We note, however, that this only holds since r is alternation confined. Consequently, $\mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}_{\text{uni}}(\text{PE})$ is implied which concludes the proof. $\square$

Theorems 15 and 16 show that pattern expression languages with respect to uniform substitution coincide with star-free initialised REGEX languages.

Corollary 17. $\mathcal{L}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}_{\text{uni}}(\text{PE})$.

In Sections 3 and 4 and in the present section, we have investigated several proper subclasses of the class of REGEX languages and their mutual relations, which can be summarised as follows: $\mathcal{L}_{\{\Sigma^*\}}(\text{PAT}) \subset \mathcal{L}_{\text{REG}}(\text{PAT}) \subset \mathfrak{L}_{\text{ro},1} \subset \mathfrak{L}_{\text{ro},2} \subseteq \mathfrak{L}_{\text{ro},3} \subseteq \ldots \subseteq \mathfrak{L}_{\text{ro},\infty} = \mathcal{H}^*(\text{REG}, \text{REG}) = \mathcal{L}_{\text{it}}(\text{PE}) \subset \mathcal{L}_{\text{uni}}(\text{PE}) = \mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX})$.

6. REGEX with a Bounded Number of Backreferences

It is a well known fact that the membership problem for REGEX languages is NP-complete (cf. Aho [1] and Angluin [3]). Furthermore, Aho states that for a REGEX with k referenced subexpressions, it can be solved in time that is exponential only in k: for all possible tuples $(u_1, u_2, \ldots, u_k)$ of factors of the input word w, we try to match r to w in such a way that, for every i, $1 \leq i \leq k$, the i^{th} referenced subexpression is matched to u_i. This procedure can be carried out in time $\text{O}(|w|^{2k})$, but, unfortunately, it is incorrect, since it ignores the possibility that the referenced subexpressions under a star (and their backreferences) can be matched to a different factor in every individual iteration of the star. On the other hand, if we first iterate every expression under a star that contains a referenced subexpression an arbitrary number of times, then, due to the late binding of backreferences, we introduce arbitrarily many new referenced subexpressions and backreferences, so there is an arbitrary number of factors to keep track of.

In the following, we answer the question of whether the membership problem for REGEX can be solved in time that is exponential only in the number of referenced subexpressions in the positive, which yields the polynomial time solvability of the membership problem for REGEX with a bounded number of backreferences.

Theorem 18. *Let $k \in \mathbb{N}$. The membership problem for* REGEX *with at most k referenced subexpressions can be solved in polynomial time.*

Proof. Let r be a REGEX with k referenced subexpressions. We shall show that r can be transformed into a nondeterministic multi-head automaton M_r (see Holzer et al. [10] for a survey) with $(3k + 2)$ heads, $\mathrm{O}(|r|)$ states and M_r accepts exactly $\mathcal{L}(r)$. Since this transformation can be carried out in polynomial time and since the acceptance problem for nondeterministic multi-head automata can be solved in polynomial time with respect to the number of input heads and the number of states, the statement of the theorem follows. We shall now construct M_r.

First we note that a nondeterministic two-way multi-head automaton can use two input heads in order to implement a counter that can store numbers between 0 and the current input length (the details are left to the reader). The automaton M_r uses $2k$ of its $3k + 2$ input heads in order to implement k individual such counters. One input head is the *main head*, another one is the *auxiliary head* and the remaining k heads are enumerated from 1 to k. Initially, all input heads are located at the left endmarker. The finite state control contains a special pointer, referred to as the *r-pointer*, that is initially located at the left end of r. In the computation of M_r, the main head is moved over the input from left to right, checking whether or not the input word satisfies r, just as it is done by a classical nondeterministic finite state automaton that accepts the language given by a classical regular expression. Simultaneously, the r-pointer is moved over r. When the r-pointer enters the referenced subexpression i, then we move head i to the position of the main head, we start counting every following step of the main head on counter i and we stop counting as soon as the r-pointer has left the referenced subexpression i. This means that we store the length of the factor that has been matched to the referenced subexpression i in counter i, whereas head i scans now the position where this factor starts. Now if the r-pointer encounters a backreference $\backslash i$, it is checked whether or not at the positions scanned by the main head and head i the same factor occurs with the length stored by counter i. It is also possible that $\backslash i$ is encountered without having visited the referenced subexpression i. In this case, counter i stores 0, which means that $\backslash i$ is treated as the empty word. If the r-pointer encounters the referenced subexpression i for a second time, which is possible since it can occur under a star, then counter i and head i are simply reset and then the referenced subexpression i is handled in exactly the same way as before. This ensures that in different iterations of a star every referenced subexpression is treated individually and only the factor that is matched to it in the very last iteration is stored for future backreferences. It can be easily verified that a word is in $\mathcal{L}(r)$ if and only if it is possible that M_r accepts that word. Since the finite state control only needs to keep track of the position of the r-pointer, $\mathrm{O}(|r|)$ states are sufficient. □

References

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

[2] J. Albert and L. Wegner. Languages with homomorphic replacements. *Theoretical Computer Science*, 16:291–305, 1981.

[3] D. Angluin. Finding patterns common to a set of strings. In *Proc. 11th Annual ACM Symposium on Theory of Computing*, pages 130–141, 1979.

[4] H. Bordihn, J. Dassow, and M. Holzer. Extending regular expressions with homomorphic replacement. *RAIRO Theoretical Informatics and Applications*, 44:229–255, 2010.

[5] 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.

[6] C. Câmpeanu and S. Yu. Pattern expressions and pattern automata. *Information Processing Letters*, 92:267–274, 2004.

[7] B. Carle and P. Narendran. On extended regular expressions. In *Proc. LATA 2009*, volume 5457 of *LNCS*, pages 279–289, 2009.

[8] D. D. Freydenberger. Extended regular expressions: Succinctness and decidability. In *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011*, volume 9 of *LIPIcs*, pages 507–518, 2011.

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

[10] M. Holzer, M. Kutrib, and A. Malcher. Complexity of multi-head finite automata: Origins and directions. *Theoretical Computer Science*, 412:83–96, 2011.

[11] L. Kari, G. Rozenberg, and A. Salomaa. L systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 5, pages 253–328. Springer, 1997.

[12] S.C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, volume 34 of *Annals of Mathematics Studies*, pages 3–41. Princeton University Press, 1956.

[13] G. Della Penna, B. Intrigila, E. Tronci, and M. Venturini Zilli. Synchronized regular expressions. *Acta Informatica*, 39:31–70, 2003.

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

[15] K. Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11, 1968.

[16] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 2, pages 41–110. Springer, 1997.