# On Extended Regular Expressions

Benjamin Carle and Paliath Narendran

Dept. of Computer Science
University at Albany–SUNY
Albany, NY 12222

**Abstract.** In this paper we extend the work of Campeanu, Salomaa and Yu [1] on extended regular expressions featured in the Unix utility *egrep* and the popular scripting language *Perl*. We settle the open issue of closure under intersection and provide an improved pumping lemma that will show that a larger class of languages is not recognizable by extended regular expressions. We also investigate some questions regarding extended multi-pattern languages introduced by Nagy in [2].

## 1 Introduction

*Grep,* a well-known command line search utility, is used regularly on Unix and other operating systems to find matching lines in files or standard input. Grep uses regular expressions to match patterns, thereby allowing a user to quickly find important data in very large files or command output. *Egrep,* a variant of grep, uses extended regular expressions[1] to increase the set of languages recognizable by the utility. Because the set of languages recognized by egrep is larger than that of theoretical regular expressions, it is important to understand the expressive power of this utility.

In this paper we extend the pioneering work of [1]; we show that the family of languages recognizable by extended regular expressions is not closed under intersection, thereby settling an open problem. Furthermore, we introduce a different pumping lemma and use that lemma to show a class of languages that satisfy the pumping property of [1] but are not expressible by extended regular expressions. We also investigate some decidability and complexity issues.

We also consider the work of Nagy [2] which extends the multi-pattern languages (MPL) defined by Kari, Mateescu, Paun and Salomaa [3]. We show that the class Nagy defines, which we call *extended multi-pattern languages* (EMPL), is a strict subclass of the family of languages recognized by egrep. We also settle some questions that are left open in [2].

## 2 Definitions

The syntax of extended regular expressions as in *egrep* and *Perl* is defined in [1]. Standard regular expressions, as specified in formal language theory, are

[1] Note our definition of extended regular expressions includes backreferences unlike the definition given in some other sources.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 279–289, 2009.

extended using *backreferences.* The backreference $\backslash n$ stands for the string previously matched by the regular expression between the $n^{th}$ left parenthesis and the corresponding right parenthesis. As is well-known, this significantly increases expressive power; for instance, the expression `((aa)+a)\1*` specifies the language

$$\{\, a^i \mid i > 0 \text{ and } i \text{ is not a power of } 2\}$$

which is not even context-free. Similarly `(a+)(b+)\1\2` specifies

$$\{\, a^i b^j a^i b^j \mid i, j > 0\}$$

which is not context-free either.

For clarity, let us number left parentheses, starting with 1, from the left. Give the same numbers to the corresponding (matching) right parentheses.

$$\underset{1}{(}\ldots\underset{2}{(}\ldots\underset{2}{)}\ldots\underset{3}{(}\ldots\underset{3}{)}\ldots\underset{1}{)}$$

As in [1] we assume that any occurrence of a backreference `\m` in an extended regular expression is preceded by $\underset{m}{)}$.

Matching a string with an extended regular expression (eregex-matching) is often defined as follows (paraphrasing [4]):

1. If $a$ is a symbol in the alphabet, then $a$ matches $a$.
2. if $r$ matches a string $x$, then $\underset{i}{(}\, r \,\underset{i}{)}$ matches $x$ and the value $x$ is assigned to $\backslash i$.
3. $\backslash j$ matches the string that has been assigned to it.
4. if $r_1$ and $r_2$ are eregexes, then $r_1 \cup r_2$ matches any string matched by either $r_1$ or $r_2$.
5. if $r_1$ and $r_2$ are eregexes, then $r_1 r_2$ matches any string of the form $xy$ where $r_1$ matches $x$ and $r_2$ matches $y$.
6. if $r$ is an eregex, then $r^*$ matches any string of the form $x_1 \ldots x_n$, $n \geq 0$, where $r$ matches each $x_i$ $(1 \leq i \leq n)$.

A more precise definition of a match is given in [1] using ordered trees. We give below that definition too, with a slight modification. Positions in an ordered tree are denoted by sequences of positive integers, with the empty sequence denoting the root position. (See [5] for a formal definition.) Note that left-to-right lexicographic order $\prec_{lex}$ among positions corresponds to *pre-order* traversal.

An ordered tree $T$ is a valid match-tree for $w$ and $\alpha$ if and only if:

1. The root of $T$ has the label $(w,\ \alpha)$.
2. For every node $u \in dom(T)$,
   (a) if $T(u) = (w,\ a)$ for some $a \in \Sigma$, then $u$ is a leaf node and $w = a$.
   (b) if $T(u) = (w,\ \beta_1\beta_2)$, then $u$ has two children labeled, respectively, by $(w_1,\ \beta_1)$ and $(w_2,\ \beta_2)$ where $w_1 w_2 = w$.
   (c) if $T(u) = (w,\ \beta_1|\beta_2)$, then $u$ has one child labeled by either $(w,\ \beta_1)$ or $(w,\ \beta_2)$.

(d) if $T(u) = (w,\ \beta^*)$, then either $u$ is a leaf node and $w = \lambda$ or $u$ has $k \geq 1$ children labeled by $(w_1,\ \beta), \ldots, (w_k,\ \beta)$ where each $w_i \in \Sigma^+$, and $w = w_1 \ldots w_k$.

(e) if $T(u) = (w,\ (\,{}_i\,\gamma\,{}_i))$, then it has one child labeled by $(w,\ \gamma)$.

(f) if $T(u) = (w,\ \backslash m)$, then $u$ is a leaf node, $(\,{}_m\,\beta\,{}_m)$ is a subexpression of $\alpha$, and there is a node $v$ to the left of $u$ such that $T(v) = (w,\ (\,{}_m\,\beta\,{}_m))$ and no node between $v$ and $u$ has $(\,{}_m\,\beta\,{}_m)$ in its label. In other words, $w$ is the string previously (in the left-to-right pre-order) matched by $(\,{}_m\,\beta\,{}_m)$.

The difference between this definition and the one in [1] is that unassigned backreferences are not set to the empty string $\lambda$ as default in our definition. Thus there is no valid match-tree for $b$ and `((aa)|\2b)`.

The language denoted by an extended regular expression $\alpha$ is defined as

$$\mathcal{L}(\alpha) = \{\, w \in \Sigma^* \mid (w,\ \alpha) \text{ is the label at the root of a valid match-tree } \}.$$

Let EREG be the family of languages defined by extended regular expressions. A language $L$ is an EREG language if and only if there is an extended regular expression $\alpha$ such that $L = \mathcal{L}(\alpha)$. In relation to the regular languages (REG), it can be seen that

$$REG \subset EREG.$$

## 3 The Results on EREG Languages

Campeanu, Salomaa and Yu [1] proved the following pumping lemma for EREG languages. To the best of our knowledge, this is the only pumping lemma of its kind.

**Lemma 1. (The CSY Pumping Lemma)** [1] *Let $\alpha$ be an extended regular expression. Then there is a constant $N > 0$ such that if $w \in \mathcal{L}(\alpha)$ and $|w| > N$, then there is a decomposition $w = x_0 y x_1 y \cdots y x_m$, for some $m \geq 1$, such that*

1. $|x_0 y| < N$,
2. $|y| \geq 1$, *and*
3. $x_0 y^j x_1 y^j \cdots y^j x_m \in \mathcal{L}(\alpha)$ *for all* $j > 0$.

**Lemma 2.** *The language*

$$\mathcal{S} \ = \ \{\, a^i b a^{i+1} b a^k \mid k = i(i+1)k' \text{ for some } k' > 0, i > 0 \,\}$$

*is not an EREG language.*

**Proof:** Assume $\mathcal{S}$ is expressed by an eregex and let $N$ be the constant given by the CSY pumping lemma. Consider $w = a^N b a^{N+1} b a^{N(N+1)}$. Then there is a decomposition $w = x_0 y x_1 y \ldots y x_m$ for some $m \geq 1$ and from the pumping

lemma $y = a^p$ for some $p \geq 1$. Since $|x_0 y| < N$ there must be at least one occurrence of $y$ in $a^N$. Assume there are $q \geq 1$ occurrences of $y$ in $a^N$. By (3) from the CSY pumping lemma there must also be $q$ occurrences of $y$ in $a^{N+1}$ as otherwise $x_0 y^2 x_1 y^2 \ldots y^2 x_m \notin \mathcal{S}$. Let $r$ be the number of occurrences of $y$ in $a^{N(N+1)}$ and note that $N(N+1) \geq rp \geq 0$. Now consider $x_0 y^2 x_1 y^2 \ldots y^2 x_m = a^{N+qp} b a^{N+1+qp} b a^{N(N+1)+rp} \in \mathcal{S}$. Then

$$k_2(N+qp)(N+1+qp) = N(N+1) + rp$$

for some $k_2$. Since $rp \leq N(N+1)$, $N(N+1) + rp \leq 2(N(N+1))$. Since $qp \geq 1$, $(N+qp)(N+1+qp) \geq (N+1)(N+2) > N(N+1)$. Thus $k_2(N+qp)(N+1+qp) \geq k_2(N+1)(N+2) > k_2 N(N+1)$. $k_2 N(N+1) < 2(N(N+1))$ is only true for $k_2 = 1$. Thus we have $(N+qp)(N+1+qp) = N(N+1) + rp$, so

$$rp = q^2 p^2 + qp(2N+1) \tag{1}$$

Now consider $x_0 y^3 x_1 y^3 \ldots y^3 x_m = a^{N+2qp} b a^{N+1+2qp} b a^{N(N+1)+2rp} \in \mathcal{S}$. Then it must be that

$$k_3(N+2qp)(N+1+2qp) = N(N+1) + 2rp$$

for some $k_3$. But note that $(N+2qp)(N+1+2qp) = N(N+1) + \mathbf{4q^2p^2} + 2qp(2N+1)$, whereas $N(N+1) + 2rp = N(N+1) + \mathbf{2q^2p^2} + 2qp(2N+1)$ by (1). Hence such a $k_3$ cannot exist. □

**Theorem 1.** *EREG languages are not closed under intersection.*

**Proof:** The language $\mathcal{S}$ of the previous lemma is the intersection of

$$\mathcal{L}(\texttt{(a+)b(\1a)b\1+}) \text{ and } \mathcal{L}(\texttt{(a+)b(\1a)b\2+}).$$ □

We can show, by a reduction from the membership problem for phrase structured grammars, that

**Theorem 2.** *The following problem is undecidable:*

Emptiness of Intersection of Extended Regular Expressions (EIERE):

*Instance: Two eregexes $\alpha$ and $\beta$.*
*Question: Is $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$ empty?*

**Proof:** The reduction is from the membership problem for phrase-structure grammars (MPSG), a known undecidable problem, as mentioned earlier. A phrase structure grammar is specified as $G = (V, \Sigma, P, S)$ where $V$ is a finite nonempty set called the *total vocabulary*, $\Sigma \subseteq V$ is a finite nonempty set called the *terminal alphabet*, $N = V - \Sigma$ is the nonterminal alphabet, $S \in N$ is the *start symbol* and $P$ is a finite set of rules (or productions) of the form $l \to r$ where $l \in V^* N V^*$ and $r \in V^*$. The membership problem MPSG is defined as follows:

Instance: Phrase-structure grammar $G = (V, \Sigma, P, S)$ and a string $w \in \Sigma^*$
Question: Is $w \in \mathcal{L}(G)$?

Given an instance of MPSG, we construct an instance of EIERE as follows: For each production $\mathbf{l_i} \to \mathbf{r_i} \in P$ let $\alpha_i = \#((\Sigma)*)\,\mathbf{l_i}\,((\Sigma)*)\#\backslash 1\,\mathbf{r_i}\,\backslash 3$, for some $\# \notin V$, for $1 \le i \le |P|$. Let $\alpha = (\alpha_1|\alpha_2|\ldots|\alpha_n)*$. Note that the backreferences will have to be renumbered, replacing each $\backslash j$ in $\alpha_i$ with $\backslash j'$ where $j' = 4(i-1)+j+1$.

So, $\mathcal{L}(\alpha)$ is the language of sequences of derivation steps (though not necessarily continuous).

$$\underbrace{\#w_1\#w_1'}_{\alpha_{i_1}}\underbrace{\#w_2\#w_2'}_{\alpha_{i_2}}\underbrace{\#w_3\#w_3'}_{\alpha_{i_3}}\ \ldots\ \underbrace{\#w_n\#w_n'}_{\alpha_{i_n}}$$

where each $w_i = x\mathbf{l}y$ and $w_i' = x\mathbf{r}y$ for $1 \le i \le n$ for some $x, y \in \Sigma*$ and some $\mathbf{l} \to \mathbf{r} \in P$.

We now define $\beta$ to enforce derivation continuity. Consider $\beta_0 = \#((\Sigma)*)\,\#\,\backslash 1$ which matches strings of the form $\#w_i\#w_i$ for $w_i \in \Sigma*$. Let $\beta = \#S(\#((\Sigma)*)\,\#\,\backslash 2) * \#w$. Then $\mathcal{L}(\beta)$ contains all strings of the form

$$\#S\,\underbrace{\#w_1\#w_1}\underbrace{\#w_2\#w_2}\ \ldots\ \underbrace{\#w_n\#w_n}\#w \ \text{ for some } \ n \ge 0$$

where each underbraced segment matches $\beta_0$.

Thus $\mathcal{L}(\beta)$ is the language of all continuous steps. (Not necessarily derivation steps from $G$.)

Finally, if we take the intersection of the two languages, namely $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$, we get strings of the form

$$\#S\,\underbrace{\#w_1\#w_1}\underbrace{\#w_2\#w_2}\ \ldots\ \underbrace{\#w_{n-1}\#w_{n-1}}\#w_n$$

where $w_i = x\mathbf{l}y$ and $w_{i+1} = x\mathbf{r}y$ for each $1 \le i < n; x, y \in \Sigma*$; and $\mathbf{l} \to \mathbf{r} \in P$ and $\mathbf{S} \to \mathbf{w_1} \in P$, $w_n = w$. That is, we get sequences of continuous derivation steps beginning at $S$ and ending in $w$.

Therefore, $w \in \mathcal{L}(G)$ iff $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) \neq \phi$ □

**Lemma 3.** *Let $\alpha$ be an eregex. Then for any $k > 0$ there are positive integers $N(k)$ and $m$ such that if $w \in \mathcal{L}(\alpha)$ and $|w| > N(k)$ then $w$ has a decomposition $w = x_0yx_1y\cdots yx_{m'}$ ($m' < m$) such that*

1. *$|y| \ge k$, and*
2. *$x_0y^jx_1y^j\cdots y^jx_{m'} \in \mathcal{L}(\alpha)$ for all $j > 0$.*

**Proof:** Let $N(k) = |\alpha|2^tk$ where $t$ is the number of backreferences in $\alpha$. Then if $|w| > N$ there is a substring of $w$ of length $\ge k$ that matches a Kleene star in $\alpha$. (Each backreference can at most double the length of the word it matches.) Let $m = t + 1$.

Let $w = x_0yz$ where $y$ is the *rightmost* largest substring of $w$ that matches a Kleene star. Then clearly $|y| \ge k$. Let $t'$ equal the number of (direct or indirect) backreferences to any expression that contains this star. Let $m' = t' + 1$. Let $z = x_1yx_2yx_3\ldots x_{m'}$ where the multiple instances of $y$ correspond to these backreferences. Then $w = x_0yx_1yx_2y\ldots yx_{m'}$ and clearly $x_0y^jx_1y^jx_2y^j\ldots y^jx_{m'} \in \mathcal{L}(\alpha)$ for all $j \ge 1$. □

**Lemma 4.** *The language* $\mathcal{P} = \{ wcw^R \mid w \in \{a,b\}^* \}$ *satisfies the CSY pumping property. (*$w^R$ *stands for the reverse of* $w$*.)*

**Lemma 5.** *The language* $\mathcal{P} = \{ wcw^R \mid w \in \{a,b\}^* \}$ *is not an EREG language.*

**Proof:** Assume $\mathcal{P}$ is an EREG language. Let $k = 5$. Let $N(k)$ and $m$ be the constants given by Lemma 5. Consider $w = (abaabb)^{N(k)}c(bbaaba)^{N(k)}$. Then there is a decomposition $w = x_0yx_1y\cdots yx_{m'}$ for some $m' < m$. By Lemma 3, $|y| \geq 5$. ($k = 5$) Observe that $(abaabb)^{N(k)}$ and $(bbaaba)^{N(k)}$ do not share any common substrings of length $\geq 5$. Therefore, $y$ must occur to the left or the right of $c$, but not both. Consequently, $x_0y^2x_1y^2\ldots y^2x_{m'} \notin \mathcal{P}$. □

We now consider the Matching Problem for Extended Regular Expressions (MERE):

Instance: An eregex $\alpha$ and a string $w \in \Sigma^*$.
Question: Is ($w$,$\alpha$) the label at the root of a valid match tree?

This has been shown to be NP-complete [4]. It turns out that the problem is NP-complete *even if the target alphabet is unary*:

**Theorem 3.** *The matching problem for extended regular expressions is NP-complete even when the target (subject) string is over a unary alphabet.*

**Proof:** Membership in NP follows from the earlier result. NP-hardness can be proved by a reduction from the vertex cover problem.
Vertex Cover (VC)

Instance: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Is there a $V' \subseteq V$ such that $|V'| \leq k$ and
$\forall (u, v) \in E : \; u \in V' \vee v \in V'$?

Given an instance of VC, construct an instance of MERE as follows: Define $n = |V|$ and $m = |E|$. Without loss of generality, assume the vertices are numbered from 2 to $n+1$, so $V = \{2,3,\ldots \mathrm{n{+}1}\}$ and $E \subseteq \{(i,j) \mid 2 \leq i \leq n+1, 2 \leq j \leq n+1\}$. (Note: The $n$ vertices are numbered from 2 to $n+1$ to account for the shifting of backreferences caused by the outer parenthesis of $\alpha_0$, defined below.) Let $\Sigma = \{a\}$. Let $w = a^{k+|E|} = a^{k+m}$.

Vertex Component: Construct $\alpha$ as follows:

Let $\alpha_0 = \underset{1}{(}\ \underset{2}{(}a\underset{2}{)}\ |\ \underset{3}{(}a\underset{3}{)}\ |\ \underset{4}{(}a\underset{4}{)}\ |\ \ldots\ |\ \underset{n}{(}a\underset{n}{)}\ |\ \underset{n+1}{(}\ a\ \underset{1}{)}\ )*$

That is, $\alpha_0$ is $n$ copies of $(a)$ connected by *or*, and then starred. Note that $\alpha_0$ can be constructed in $O(|V|)$ time.

Edge Component: Assume the edges are ordered from 1 to $m$: $e_t \in E$ for $1 \leq t \leq m$. For each $e_t = (i,j) \in E$, let $\alpha_t = (\backslash i \mid \backslash j)$. That is, each $\alpha_t$ represents the $t^{th}$ edge via backreferences, with the backreference incremented by one to account for the outer parenthesis in $\alpha_0$. Note that this can be done in $O(|E|)$ time.

Finally, let $\alpha = \alpha_0\alpha_1\alpha_2 \ldots \alpha_m$. Note that $\alpha$ can be constructed in $O(|V|+|E|)$ time. We now show that $\alpha$ matches $w$ iff $G$ has a vertex cover of size $\leq k$. Suppose $V' \subseteq V$ is a vertex cover for $G$ with $|V'| \leq k$. Then we can find a valid match tree for $(w, \alpha)$ as follows: We can safely assume that $|V'| = k$, since additional vertices from $V$ can always be added to make this true. Let us start by matching $k$ iterations of $\alpha_0$, one for each $v \in V'$. For each $v \in V' \subseteq \{2,3,\ldots,n+1\}$ match the $v^{th}$ option of the *or* in $\alpha_0$ on a different iteration. More specifically, if we let $V' = \{v_1, v_2, \ldots, v_k\} \subseteq \{2, 3, \ldots, n+1\}$ then on the $v_i^{th}$ iteration of $\alpha_0$ match the $v_i^{th}$ option of the *or* in $\alpha_0$, which will assign $a$ to $\backslash v_i$. Thus, $\alpha_0$ matches $a^k$. Recall that each $\alpha_t = (\backslash i \mid \backslash j)$ for $1 \leq t \leq m$ represents edge $e_t = (i, j)$. Since $V'$ is a vertex cover for $G$, at least one of $\{i, j\}$ is in $V'$. Therefore, at least one of $\{\backslash i, \backslash j\}$ is already defined in our match tree. If $\backslash i$ is defined, match it. Otherwise match $\backslash j$. Furthermore, any defined backreference $\backslash 2, \backslash 3, \ldots \backslash n+1$ can only match a single $a$, so $\alpha_1\alpha_2 \ldots \alpha_m$ matches $a^m$.

Therefore, $\alpha = \alpha_0\alpha_1\alpha_2 \ldots \alpha_m$ matches $a^k a^m = a^{k+m} = w$.

Conversely, suppose $(w, \alpha) = (a^{k+m}, \alpha)$ is the root of a valid match tree. Then we can find a vertex cover $V' \subseteq V$ for $G$ with $|V'| \leq k$ as follows: To begin, let $V' = \phi$. Since there is a match for $\alpha$, each of $\alpha_1\alpha_2 \ldots \alpha_m$ must be defined. Thus, each $\alpha_t$ for $1 \leq t \leq m$ matches a single $a$. For each $\alpha_t = (\backslash i \mid \backslash j)$, if $\backslash i$ is matched, let $V' = V' \cup \{i\}$, else if $\backslash j$ is matched, let $V' = V' \cup \{j\}$. Recall that $\alpha_1\alpha_2 \ldots \alpha_m$ matches $a^m$. Thus, $V'$ is a vertex cover for $G$ since it contains one vertex from each edge (from each corresponding $\alpha_i$).

Then $\alpha_0$ must match the remaining $a^k$. Therefore, there can be at most $k$ unique backreferences defined, which means there can be at most $k$ distinct vertices in $V'$. Therefore, $|V'| \leq k$ and $G$ has a vertex cover of size $\leq k$.

Thus $\alpha$ matches $w$ iff $G$ has a vertex cover of size $\leq k$. Furthermore, the reduction can be done in $O(|V| + |E|) = O(|V|^2)$ time. Therefore, MERE is NP-complete. □

**Remark:** Notice that this proof crucially uses the (semantic) assumption that unassigned backreferences are not set to the empty string. The result can also be proved without using this 'feature'. However, the proof is a bit more complicated and we omit it here.

## 4 Extended Multi-Pattern Languages (EMPL)

Let $\Sigma$ be a finite set of terminals $\{a_1, \ldots, a_n\}$ and $V = \{x_1, x_2, \ldots\}$ be an infinite set of variables ($\Sigma \cap V = \emptyset$). Then a *pattern* is a non-null finite string over $\Sigma \cup V$. We use the terms *erasing* (E) and *non-erasing* (NE) pattern languages in the following sense. Let $H_{\Sigma,V}$ be the set of morphisms $h : (\Sigma, V)^* \to (\Sigma, V)^*$. The $E$ pattern language generated by a pattern $\pi$ is defined as

$$L_E(\pi) = \{w \in \Sigma^* \mid \exists h \in H_{\Sigma,V}((\forall a \in \Sigma : h(a) = a) \wedge w = h(\pi))\}$$

The $NE$ pattern language generated by a pattern $\pi$ is defined as

$$\{w \in \Sigma^* \mid \exists h \in H_{\Sigma,V}((\forall a \in \Sigma : h(a) = a) \wedge (\neg\exists v \in V : h(v) = \lambda) \wedge w = h(\pi))\}$$

and denoted as $L_{NE}(\pi)$.

Given a set of patterns $\{\pi_1, \pi_2, \ldots, \pi_n\}$, the *E-multi-pattern language* (MPL-E) they define is $\bigcup_{i=1}^{n} L_E(\pi_i)$. Similarly, the *NE-multi-pattern language* (MPL-NE) defined by $\{\pi_1, \pi_2, \ldots, \pi_n\}$ is $\bigcup_{i=1}^{n} L_{NE}(\pi_i)$.

This notion was extended by Nagy [2] to that of EMP expressions in the following way:

Let $\{\pi_1, \pi_2, \ldots, \pi_n\}$ be a set of patterns. Each pattern $\pi_i$ $(1 \leq i \leq n)$ is an EMP expression. If $\gamma$ and $\delta$ are EMP expressions then

$\gamma \vee \delta$ is also an EMP expression (using the operation union),
$\gamma \cdot \delta$ is also an EMP expression (using the operation concatenation),
$\gamma^*$ is also an EMP expression (using the operation Kleene star).

In other words, extended multi-pattern (EMP) expressions are obtained from the patterns $\pi_1, \ldots, \pi_n$ by using finitely many regular operators. The EMP expressions which can be obtained *without using union* $(\vee)$ are called *star-pattern* expressions (EMSP expressions).

The *erasing extended multi-pattern language* defined by an EMP expression can be obtained from the E pattern languages in the following way:

$L_E(\gamma \vee \delta) = L_E(\gamma) \cup L_E(\delta)$ (using the operation union),
$L_E(\gamma \cdot \delta) = L_E(\gamma) \cdot L_E(\delta)$ (using the operation concatenation),
$L_E(\gamma^*) = (L_E(\gamma))^*$ (using the operation Kleene star).

We then define EMPL-E (Extended Multi-Pattern Languages – Erasing) to be the family of erasing extended multi-pattern languages [2].

The non-erasing extended multi-pattern language defined by an EMP expression can be obtained from the NE pattern languages in the following way:

$L_{NE}(\gamma \vee \delta) = L_{NE}(\gamma) \cup L_{NE}(\delta)$ (using the operation union),
$L_{NE}(\gamma \cdot \delta) = L_{NE}(\gamma) \cdot L_{NE}(\delta)$ (using the operation concatenation),
$L_{NE}(\gamma^*) = (L_{NE}(\gamma))^*$ (using the operation Kleene star).

Let EMPL-NE (Extended Multi-Pattern Languages – Non-Erasing) stand for the family of non-erasing extended multi-pattern languages.

It is not hard to see that EMPL-E (resp. EMPL-NE) is the *regular closure* [7,8] of the family of E pattern (resp. NE pattern) languages.

If $\gamma$ is an EMSP expression then $L_E(\gamma)$ is an erasing extended multi-star-pattern language. Let EMSPL-E be the family of erasing extended multi-star-pattern languages. Similarly, if $\gamma$ is an EMSP expression then $L_{NE}(\gamma)$ is a non-erasing extended multi-star-pattern language. Let EMSPL-NE be the family of non-erasing extended multi-star-pattern languages.

Finally, let EMPL = EMPL-E ∪ EMPL-NE, and likewise, let EMSPL = EMSPL-E ∪ EMSPL-NE. Now two questions arise:

Question 1: Does EMPL-E = EMPL-NE (= EMPL)?

Question 2: Does EMSPL-E = EMSPL-NE (= EMSPL)?

[2] Note that this is not the same as the family PL(REG, REG) defined in [6].

We answer Question 1 affirmatively and Question 2 negatively.

**Lemma 6.** *The language* $L = \{ xbx \mid x \in \{a, b\}^* \} \in$ *EMSPL-E.*

**Proof:** Let $\Sigma = \{a, b\}$ and $v \in V$. Then $\alpha = vbv$ is an EMSP expression and $L_E(\alpha) = L$. □

**Lemma 7.** *The language* $L = \{ xbx \mid x \in \{a, b\}^* \} \notin$ *EMSPL-NE.*

**Proof:** (by contradiction). Assume $L$ is an NE star-pattern language. Then there is an NE star-pattern expression $\alpha$ such that $L_{NE}(\alpha) = L$. $\alpha$ cannot contain the union operator since it is an NE star-pattern expression.

Clearly $\alpha$ cannot have a star as its outer-most operator. Otherwise, $\alpha$ would match $\lambda$, which is not in $L$. Define a language to be *non-trivial* if and only if it is neither empty nor the singleton set $\{\lambda\}$.

**Claim:** $L$ is not the concatenation of two non-trivial languages.

**Proof:** Assume the contrary and let $L = A \circ B$ with $A$ and $B$ non-trivial. Without loss of generality assume that $b \in B$. Hence every non-empty string in $A$ must be of the form $ubbu$ for some $u \in \{a, b\}^*$. Now consider the string $aba$ which belongs to $L$. $aba$ has to be in $B$ since no non-empty prefix of it can be in $A$. But then the word equation $ubbuaba =^? xbx$ has no solution. □

Since $\alpha$ cannot be a single pattern either, the result follows[3].

**Lemma 8.** *EMSPL-E* $\neq$ *EMSPL-NE.*

**Proof:** The language $L$ of Lemma 6 (and 7) is in EMSPL-E, but is not in EMSPL-NE. □

**Lemma 9.** *EMPL-NE = EMPL-E.*

**Proof:** This follows from the results of [3]. We omit the proof. □

**Lemma 10.** *Every EMPL language is semi-linear.*

**Proof-sketch:** Every language in MPL is semi-linear [3]. The family of semi-linear languages is closed under union, concatenation and star. □

**Lemma 11.** *The EREG language* $\mathcal{L}$`((a*)b(\1))` $= \{a^i b a^i \mid i \geq 0\}$ *is not an EMPL.*

**Proof-idea:** If any of the patterns used in the defining expression contains a variable, then it can be replaced with $bb$. □

**Theorem 4.** *EMPL* $\subset$ *EREG.*

[3] Thus without backreferences the analogous result to Theorem 17 of [2] does not hold.

**Proof-idea:** EREG $\not\subseteq$ EMPL follows from the previous lemma. EMPL $\subseteq$ EREG can be shown as follows: Given an EMP expression $\gamma$ over terminal alphabet $\Sigma = \{a_1, \ldots, a_n\}$, we can construct an equivalent EREG pattern $\alpha$. Let $\{\pi_1, \pi_2, \ldots, \pi_n\}$ be the set of patterns that occur within $\gamma$. For each pattern $\pi_i$ $(1 \leq i \leq n)$, replace the first occurrence of a variable within $\pi_i$ with $((a_1|a_2|\ldots|a_n)*)$, where $k$ is the index of the outer left parenthesis we are adding. Replace each subsequent occurrence of the same variable with $\backslash k$. □

**Lemma 12.** *The following problem is undecidable:*

Instance: Two sets of patterns $P_1$ and $P_2$.
Question: Is $(L_E(P_1))^* \subseteq (L_E(P_2))^*$?

**Proof idea:** The problem of deciding, given two patterns $\alpha$ and $\beta$, whether $L_E(\alpha) \subseteq L_E(\beta)$ is known to be undecidable [9]. Let # be a new symbol, not present in the alphabet $\Sigma$ of $\alpha$ and $\beta$. Let $\Omega = \Sigma \cup \{\#\}$. Now form the sets of patterns

$$\Gamma = \{\#\alpha\#\} \text{ and } \Delta = \{\#\beta\#,\ \#x_1\#x_2\#\}.$$

**Claim 1:** $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ over $\Omega$ if and only if $L_E(\alpha) \subseteq L_E(\beta)$ over $\Sigma$.

**Claim 2:** $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ if and only if $(L_E(\Gamma \cup \Delta))^* = (L_E(\Delta))^*$.

Thus the equivalence problem for EMSPL-E is undecidable. □

The same technique will work for EMSPL-NE, except that $\Delta$ will have to be defined a little differently, as $\{\#\beta\#,\ \#x_1\#x_2\#,\ \#\#x_2\#,\ \#x_1\#\#,\ \#\#\#\}$. It can also be shown that

**Lemma 13.** *For every EMP of the form $\alpha^*$ (i.e., with star as the outermost operator), there is an EMSP $\gamma$ such that $L_{NE}(\gamma) = L_E(\alpha^*)$.*

**Proof-sketch:** Since the families EMPL-E and EMPL-NE are the same, there must be an EMP $\beta$ such that $L_{NE}(\beta) = L_E(\alpha)$. It can be shown (see e.g., [10]) that an expression equivalent to $\beta^*$ that does not use $\vee$ can be found. □

**Theorem 5.** *The equivalence problem for EMSPL-NE is undecidable.*

This settles an open problem given in [2].

**Acknowledgements.** We thank Colin Scheriff for work on an earlier version of the paper [11] and in particular on the proof of Lemma 3. We also thank the referees for their insightful comments.

## References

1. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. Int. J. Found. Comput. Sci. 14, 1007–1018 (2003)
2. Nagy, B.: On the language equivalence of NE star-patterns. Inf. Process. Lett. 95, 396–400 (2005)

3. Kari, L., Mateescu, A., Paun, G., Salomaa, A.: Multi-pattern languages. Theor. Comput. Sci. 141, 253–268 (1995)
4. Aho, A.V.: Algorithms for finding patterns in strings. In: Handbook of Theoretical Computer Science. Algorithms and Complexity (A), vol. A, pp. 255–300. MIT Press, Cambridge (1990)
5. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Harper & Row (1986)
6. Dumitrescu, S., Păun, G., Salomaa, A.: Languages associated to finite and infinite patterns. Rev. Roum. Math. Pures Appl. 41, 607–625 (1996)
7. Bertsch, E., Nederhof, M.J.: Regular closure of deterministic languages. SIAM J. Comput. 29, 81–102 (1999)
8. Kutrib, M., Malcher, A.: Finite turns and the regular closure of linear context-free languages. Discrete Applied Mathematics 155, 2152–2164 (2007)
9. Jiang, T., Salomaa, A., Salomaa, K., Yu, S.: Decision problems for patterns. J. Comput. Syst. Sci. 50, 53–63 (1995)
10. Nagy, B.: A normal form for regular expressions. In: Eighth International Conference on Developments in Language Theory (DLT 2004), Auckland, New Zealand (2004), `www.cs.auckland.ac.nz/CDMTCS/researchreports/252dlt04.pdf`
11. Carle, B., Narendran, P., Scheriff, C.: On extended regular expressions. In: Twenty-first International Workshop on Unification (UNIF 2007), Paris, France (June 2007)