**CS–2006–20**

# On Pattern Expression Languages

*Cezar Câmpeanu and Nicolae Santean*

*Technical Report 20*

David R. Cheriton School of Computer Science
University of Waterloo

*2006*

# On Pattern Expression Languages

Cezar Câmpeanu[1] and Nicolae Sântean[2]

[1] Department of Computer Science and Information Technology, University of Prince Edward Island
[2] Computer Science Department, University of Waterloo

**Abstract.** In this paper we show that the family of pattern expression languages is closed under the intersection with regular languages. Since this family is not closed under complement but is closed under reverse, a natural question arises, that is, whether particular languages such as those containing words of type $ww^R$ are pattern expression languages or not. We give a proof for a negative answer to this question, and we provide several examples of languages which can not be specified by pattern expressions.

## 1 Introduction

Regular expressions are used in many practical applications, e.g., Perl, Awk, Python, egrep, vi, and emacs. It is known that practical regular expressions are different from their theoretical counterparts. Practical regular expressions [6] are often called "regex". Regex were developed under the influence of theoretical regular expressions; and in some implementations (e.g. in Lex, [10]) they bare a strong similarity to them. However, regex are quite different in many other environments, and in most implementations regex can express language families larger than that of regular languages. For example, Perl regex [6] can express $L_1 = \{a^n ba^n \mid n \geq 0\}$ and $L_2 = \{ww \mid w \in \{a, b\}^*\}$. Anyway, Perl regex and pattern expressions cannot express the language $L_3 = \{a^n b^n \mid n \geq 0\}$, $L_4 = \{ww^R \mid w \in \{a, b\}^*\}$ (as will be shown in this paper), or $L_5 = \{\ (abc)^n (cba)^n \mid n \geq 0\}$. It is relatively easy to show that a language can be expressed by a regex. For example, $L_1$ can be expressed by the Perl regex $(a*)b\backslash 1$ and $L_2$ by $((a|b)*)\backslash 1$. Unlike language $L_3$, which is known that it cannot be generated in Perl (and it has been proven that is neither an extended regex language in [2], nor a pattern expression language in [3]), very little is known about $L_4$. There has been a long-standing controversy, on whether $L_4$ can or can not be generated in Perl or by pattern expressions. Some people believe the positive, although they cannot give a Perl regex for it, whereas some others believe the opposite, yet they cannot provide a rigorous argument to support their claims. For the latter, the difficulty consists in the fact that both pumping lemmas for extended regex and pattern expression fail to give a contradiction in the case of $L_4$. Consequently, the present result stating that the mirror language ($L_4$) is not a pattern expression language is expected to raise the interest of the research community as well as that of practitioners, due to its practical implications. In this paper we will mainly focus on new results for pattern expressions, rather than for extended regex. One reason for this is that they are more versatile, and there is strong evidence that the two models

are equivalent: although no formal proof is available yet, in [3] was proposed a method for converting a pattern expression into an extended regex and vice-versa.
Related to our study on pattern expressions (PE) we mention [1], where was considered the addition of a reverse operation to extend the power of pattern languages, or [5], where was proposed the use of a mirror operation to increase the power of multi-pattern languages. Other variations on multi-pattern languages, or similar constructs can be found in [8, 14, 5, 9], or more recently, in [9, 12], where one can find a comprehensive survey on the topic. The formalism and most of the results present in this research stream are developed under the influence of parallel communication grammar systems and other similar devices. It would be interesting and rather challenging to analyze the relationship between the formalism proposed and developed in [2, 3] (and used throughout this paper) and those employed in the past. For example, in this paper we have used pattern automata introduced in [3], which bear similarities with parallel communicating finite automata systems, mentioned in [11]. It is our belief that the two models are not equivalent, matter which we plan to address in the near future. Another parallel can be drawn between PE languages and certain families of languages studied in the past. Despite their proximity, we could not identify a past model equivalent to pattern expressions, and we believe that none of the results present in this paper can be stated equivalently in the other frameworks. One explanation of this status quo, of having several models sharing common ideas and yet being rather different, is that due to their particularities (e.g., the use of recursive definitions and iterating mechanisms), small model changes may have a huge impact on the behavior of the model. A conceptual difference between the PE model and the other models, as well as a justification for its study beside its inherent novelties, is that pattern expressions were inspired by pragmatic software applications and were influenced by the formalism of expressions (specifications) and automata (acceptors), whereas the previous work originated in the study of grammars (generative devices) and had only a purely theoretical justification. For illustration, let us emphasize some differences between the PE model and other models which although are apparently similar, their language families differ:

- In multi-patterns, variables are replaced with words given by a regular or a context free language, while in pattern expressions variables are replaced with words from a pattern expression language, in a recursive manner – thus there may be stages where substitutions are done with words in a context-sensitive language.

- For multi-patterns there is no order for substituting variables (all substitutions are done in one step), whereas for pattern expressions the substitutions are done in a predefined order and in a finite number of steps.

- Despite their names, iterated patterns (model introduced in [8]) do not contain a Kleene operator (the word “iteration” refers to substitutions) in contrast with pattern

expressions.

- The model that seems to be the closest to PE is the so-called iterative multi-patterns where the patterns are given by a language generated by a regular grammar. However, their differences become obvious when their families of languages are related to those of the Chomsky hierarchy.

- In some sense, one can view the idea behind pattern expressions as a combination of ideas used in multi-patterns and iterated patterns. Yet, we are not sure whether even combining these models in some way one can built a model equivalent with PE.

Finally, we should mention that one of the motivations of the present paper, i.e., whether $L_4$ or other similar languages belong to PE, has not been addressed in the past despite being such a natural matter. One reason for this may be that the problem has turned out to be either trivial for some models, or too complex for others. It is in our hope that our results will inspire studies on, and solutions to this and other problems remained open for the other related models.

## 2 Notations and Definitions

In this section we provide some basic notions and notations used throughout the paper. For definitions of formal language theory and automata, not covered here, we refer the reader to [7], [15], or [17]. We also give a formal definition for pattern expressions, along with several illustrative examples. In order to keep the paper as self contained as possible, we provide an informal definition and examples for extended regex as well.

An alphabet $\Sigma$ is a finite non-empty set. A word over $\Sigma$ is an element of the free monoid $\Sigma^*$, that is, a finite string of symbols in $\Sigma$. For $\Gamma \subseteq \Sigma$ let $|\cdot|_\Gamma \colon \Sigma \longrightarrow \mathbb{N}$ be defined as follows: $|a|_\Gamma = 1$ if $a \in \Gamma$, and $|a|_\Gamma = 0$ if $a \notin \Gamma$ (i.e., the characteristic function for $\Gamma$). We extend the function $|\cdot|_\Gamma$ to a monoid homomorphism from $(\Sigma^*, \cdot)$ to $(\mathbb{N}, +)$, thus $|w|_\Gamma$ will denote the number of occurrences of symbols of $\Gamma$ in the word $w$. When $\Gamma = \Sigma$ we omit the subscript and $|w|$ becomes the length of the word $w$. When $\Gamma = \{a\}$, that is, a singleton, we use the notation $|w|_a$ to express $|w|_{\{a\}}$. The word with no letters (the empty word) is denoted by $\lambda$ and $|\lambda|_\Gamma = 0$, for any $\Gamma \subseteq \Sigma^*$. Similarly, $|\ w\ |_\emptyset = 0$ for all $w \in \Sigma^*$.

A regular expression over $\Sigma$ is described recursively as follows:

1. the empty set and any letter $a$ of the alphabet $\Sigma$ are regular expressions denoting the languages $\emptyset$ and $\{a\}$, respectively;
2. if $\alpha$ and $\beta$ are regular expressions, then $\alpha\beta$, $\alpha + \beta$, $\alpha^*$, and $(\alpha)$ are regular expressions denoting the languages $L(\alpha)L(\beta)$, $L(\alpha) \cup L(\beta)$, $L(\alpha)^*$, and $L(\alpha)$, respectively;
3. any regular expression is obtained by applying the above rules a finite number of times.

In other words, a regular expression over $\Sigma$ is the set of all well-formed parenthesized infix formulae obtained from the elements of $\Sigma^*$ (viewed as atomic formulae), the null operator $\lambda$, the binary operators $+$ and $\cdot$ (expressed as juxtaposition), and the unary operator $^*$. Since by convention $L(\emptyset^*) = \{\lambda\}$ (we omit the explanation here), we usually denote the regular expression $\emptyset^*$ by $\lambda$. A word belonging to the language of a regular expression is said that "matches the regular expression".

An extended regular expression (extended regex) is a regular expression which accepts the additional atoms "$\backslash n$", $n \in \mathbb{N}$, used as follows. Each pair of parentheses of the regular expression is numbered according to the order of open parentheses. An atom $\backslash n$ of the regular expression is replaced with the content of the $n$'th pair of parentheses during the matching of a word. For example, the word $a^2ba^2$ matches the expression $({}_1({}_2a^*)b)\backslash 2$ since the content of the second pair of parentheses matches the subword $a^2$ and $\backslash 2$ duplicates it. For more on extended regex consult [6].

Next we give a formal definition for *pattern expressions* followed by a few examples.

**Definition 1.** *Let $\Sigma$ be an alphabet and $V = \{v_0, \ldots, v_{n-1}\}$ be a finite set of variables such that $V \cap \Sigma = \emptyset$. A* regular pattern *is a regular expression over $\Sigma \cup V$. A* pattern expression *is a tuple of regular patterns $p = (r_0, r_1, \ldots, r_n)$ with the following properties:*

1. *$r_0$ is a regular expression over $\Sigma$;*
2. *for $i \in \{1, \ldots, n\}$, $r_i$ is a regular expression over $\Sigma \cup \{v_0, \ldots, v_{i-1}\}$.*

*The language $L(p)$ generated by $p$ is defined as follows:*
$L_0 = L(r_0)$*, as defined for a regular expression, and for all* $i \in \{1, \ldots, n\}$,
$L_i = \left\{(u_0/v_0)\ldots(u_{i-1}/v_{i-1})u_i \;\middle|\; u_i \in L(r_i), u_j \in L_j, j \in \{0, \ldots, i-1\}\right\}$,
*where $L(p) := L_n$ and the notation $(u/v)\alpha$ expresses the substitution of all occurrences of variable $v$ in $\alpha$ by the word $u$.*

For better handling pattern expressions, we use the notation $p = (v_0 = r_0, v_1 = r_1, \ldots, v_{n-1} = r_{n-1}, r_n)$ to track easily which variable is substituted by words in which language: variable $v_i$ is substituted by words in the language $L_i$ generated by the regular pattern $r_i$.

*Example 1.*

- $p = (v_0 = (a+b)^*, v_0abbv_0)$ is a pattern expression generating all double words over the alphabet $\{a, b\}$ separated by $abb$;

- for $p = (v_0 = ab^*aaa, v_1 = bv_0v_0a^*, v_0v_1av_1bv_0)$ we have:
  $L_0 = \{ab^naaa \mid n \geq 0\}$,
  $L_1 = \{bab^naaaab^naaaa^m \mid m \geq 0, n \geq 0\}$,
  $L_2 = \{ab^laaabab^naaaab^naaaa^mabab^naaaab^naaaa^mbab^laaa \mid$
  $l \geq 0, m \geq 0, n \geq 0\}$;

– for $p = (v_0 = ab^*, v_1 = baa^*, (v_0 + v_1)(v_0 + v_1))$ we have
$L(p) = \{ab^nab^n \mid n \geq 0\} \cup \{ba^nba^n \mid n \geq 1\} \cup$
$\{ab^nba^m \mid n \geq 0, m \geq 1\} \cup \{ba^mab^n \mid n \geq 0, m \geq 1\}$;

– for $p = (v_0 = ab^*, v_0^*cv_0)$, $L(p) = \{(ab^n)^mcab^n \mid n \geq 0, m \geq 0\}$.

We emphasize that pattern expressions are extensions of patterns ([1]), i.e., words containing letters and variables. A pattern language([1, 14]) is obtained from a pattern by substituting variables with arbitrary words.

The same Pumping Lemma holds for both regex and pattern expression languages, respectively:

**Lemma 1.** *([2]) Let $L$ be a pattern expression language or a regex language. Then there is a constant $N$, such that if $w \in L$ and $|w| > N$, there is a decomposition $w = x_0yx_1yx_2 \cdots x_m$ for some $m \geq 1$, such that:*

1. $|x_0y| \leq N$,
2. $|y| \geq 1$,
3. $x_0y^jx_1y^jx_2 \cdots x_m \in L$ *for all* $j > 0$.

In order to prove the closure of pattern expression languages to intersection with regular languages, we recall the notion of pattern automaton, introduced in [3]. A pattern automaton is an automata system $P = (A_0, A_1, \ldots, A_n)$ where

$$A_0 = (Q_0, \Sigma, \delta_0, q_{0,0}, F_0), \text{ and}$$

$$A_i = (Q_i, \Sigma \cup \{v_0, \ldots, v_{i-1}\}, \delta_i, q_{i,0}, F_i),\ 0 < i \leq n,$$

are finite automata, also called modules of $P$. $A_0$ is a DFA over $\Sigma$, and for each $i \in \{1, \ldots, n\}$, $A_i$ has the same structure as $A_0$, except for the transition labels which may be either a symbol in $\Sigma$ or one of the variables $v_0$, …, $v_{i-1}$. We assume that $Q_i \cap Q_j = \emptyset$ for $0 \leq i \neq j \leq n$, and we usually denote $Q = \bigcup_{i=0}^{n} Q_i$. This automata system mimics closely the structure of a pattern expression $p = (v_0 = r_0, v_1 = r_1, \ldots v_{n-1} = r_{n-1}, r_n)$, where $r_i$ is the regular pattern corresponding to automaton $A_i$, for all $i \in \{0, \ldots, n\}$. Then $p$ will represent a pattern expression associated to the pattern automaton $P$.

If $n = 0$, the pattern automaton consists of only one automaton which operates as an usual finite automaton. For $n > 0$, $P$ uses a stack $S$ storing elements of $Q$, an array of stacks $U = \{U_i \mid 0 \leq i < n\}$, whose stacks store elements of $\{0, 1\}$, and $V = \{V_j \mid 0 \leq j < n\}$, whose stacks store elements of $\Sigma^*$. The interpretation for $U$ and $V$ is as follows. Let $p = (v_0 = r_0, v_1 = r_1, \ldots, v_{n-1} = r_{n-1}, r_n)$ be a pattern expression associated with $P$. A computation step of $P$ consists in matching a prefix of the remaining input word with an expression $r_i$ of $p$, leading to the instantiation of variable $v_i$. When this happens, the top element of each $U_j$ indicates whether the variable $v_j$ has been instantiated, whereas the

top of stack $V_j$ stores the actual string which instantiates $v_j$. All stacks of $U$ and $V$ are bounded, each containing at most $n$ elements.

The current configuration of pattern automaton $P$ can be described by its current state $q \in Q$, the remaining of the input word $w \in \Sigma^*$, the current content of the state stack $S$ and of every stack of $U$ and $V$. Thus, the current configuration at step $t$ is $(s^t, x^t, S^t, U^t, V^t)$, where $s^t$ denotes the current state and $x^t$ denotes the remaining input. This configuration is an accepting configuration if $s^t \in F_n$ and $x^t = \lambda$.

Initially, $P$ holds an input string $w \in \Sigma^*$ on its tape, and its current (initial) state is $q_{n,0}$. $S$ is empty and all the stacks of $U$ and $V$ are empty. Thus, the initial configuration is described by

$$(s^0, x^0, S^0, U^0, V^0) = (q_{n,0}, w, \lambda, \lambda, \lambda).$$

The first step of $P$ is $push(U_i, 0)$, for all $0 \leq i < n$ (meaning that no variable has been instantiated yet). The transitions between consecutive configurations are defined by one of the following rules:

1. Let $x^t = ay \in \Sigma^*$, with $a \in \Sigma$. If $s^t = p \in Q_n$, then $s^{t+1} = q$ with $q \in \delta_n(p, a)$, and $x^{t+1} = y$. If $s^t = p \in Q_i$ for some $i < n$, then $s^{t+1} = q$ with $q \in \delta_i(p, a)$, $x^{t+1} = y$, and $top(V_i) = top(V_i)a$.

2. Let $s^t = p \in Q_i$ for some $i > 0$. If for an index $j \in \{0, \ldots, i-1\}$ we have $q \in \delta_i(p, v_j)$ and $top(U_j) = 0$, then $push(S, q)$, $push(V_j, \lambda)$, $push(U_k, 0)$ for all $0 \leq k < j$. Then set $s^{t+1} = q_{j,0}$ and leave $x^{t+1} = x^t$.

3. If $s^t = p \in F_i$ for $0 \leq i < n$ and $top(U_i) = 0$, then set $s^{t+1} = top(S)$, $pop(S)$, $pop(V_j)$ for $0 \leq j < i$, $pop(U_j)$ for all $0 \leq j < i$, and set $top(U_i) = 1$.

4. Let $s^t = p \in Q_i$ for some $i > 0$. If for an index $j \in \{0, \ldots, i-1\}$ we have $q \in \delta_i(p, v_j)$, $top(U_j) = 1$, and $x^t = top(V_j)y$, then set $s^{t+1} = q$, $top(V_i) = top(V_i)top(V_j)$ and $x^{t+1} = y$.

5. If $s^t \in F_n$ and $x^t = \lambda$, then accept.

The language recognized by $P$ is:

$$L(P) = \{w \mid (q_{n,0}, w, \lambda, \lambda, \lambda) \vdash^* (f, \lambda, S, U, V),\ f \in F_n\}.$$

If for each of the automata $A_i$ with $0 \leq i \leq n$ we denote $R_i = L(A_i) \subseteq (\Sigma \cup \{v_0, \ldots, v_{i-1}\})^*$, then is easy to observe that the language recognized by $P$ is

$W_n = \{(u_0/v_0) \ldots (u_{n-1}/v_{n-1})u_n \mid u_n \in R_n,\ u_i \in W_i,\ 0 \leq i \leq n-1\}$, where

$W_0 = R_0$, and for $i \in \{1, \ldots, n-1\}$ :

$W_i = \{(u_0/v_0)\dots(u_{i-1}/v_{i-1})u_i \mid u_i \in R_i, u_j \in W_j, 0 \leq j \leq i-1\}$.
Since $R_i = L(r_i)$, it follows that $W_i = L_i$, hence $L(P) = L(p)$, i.e., the automata system recognizes the same language as the language generated by the pattern expression $p$.

We can easily see that the PA operation is non-deterministic, making the running time exponential. Since pattern automata recognize languages generated by pattern expressions, one may wonder what is the running time for recognizing a word in the language and whether is possible to construct some device that has a better running time than PA. The following theorem answers this question.

**Theorem 1.** *The membership problem for pattern expressions is NP-complete.*

*Proof.* We analyze the membership problem for pattern automata. Let $P$ be a pattern automaton as previously defined, and $w$ an input word. If we guess the "right choice" in each module $A_i$ of $P$ (i.e., we always make the right variable substitutions, hence avoiding backtracking), to recognize $w$ takes $O(|w|)$ time, therefore the problem is in NP. Since the problem $w \in L(p)$ is NP-hard when $p$ is just a pattern (see [1, T. 3.2.3, p.133]), we conclude that our problem is also NP-hard, therefore NP-complete.

*Remark 1.* For a pattern expression $p$ with $n$ regular patterns, the membership problem has $O(n^2m)$ space complexity, where $m$ is the length of the input word. This results from the fact that all words in the stacks used for simulating a pattern automaton have a length bounded by the length of the input word $w$ (storing some sub-words of $w$) and there are $2n+1$ stacks, each of depth at most $n$.

## 3 Main Results

As mentioned in the previous section, there already exists a pumping lemma holding for both pattern expression languages and extended regex languages. This lemma turns out to be too weak for proving that a language like $L_4$ (mentioned in introduction) is not a pattern expression or an extended regex language. To go around this problem, we first prove an important closure property for the family of pattern expression languages.

**Theorem 2.** *The family of pattern expression languages is closed under intersection with regular languages.*

*Proof.* Let $L = L(p)$ with $p = (r_0, r_1, \dots, r_n)$ be a pattern expression language and $R$ be a regular language accepted by a trim DFA $B = (\Sigma, Q_B, 0, \delta_B, F_B)$.

The idea of the proof, based on a Cartesian product construction [7, 4], is as follows. We consider a pattern automaton $P$, such that $L(P) = L(p)$, and construct a pattern automaton which simulates the run of $P$ in "parallel" with $B$. This simulation works purely in parallel when $P$ transits from state to state based on letters in $\Sigma$. When $P$ chooses a transition labeled with a variable name "$v$", $B$ is put on hold, and $P$ calls the proper

module which takes over the resolution of $v$. Whenever a module called recursively uses a transition labeled with a letter in $\Sigma$, $B$ is revived and advances again in parallel with $P$. This idea is facing the challenge of making this simulator look like a pattern automaton. The problem has turn to be rather complicate, and we will see that the newly constructed pattern automaton uses significantly more variables than $P$ – number increased of order $O(\mid Q_B \mid^2)$. One essential trick used in this proof is to index the new variables in such manner, that the subscripts themselves provide information on where the run of $B$ has paused or where it has to resume from, in terms of the states of $B$. Hence, we anticipate that beside the normal indexing of variables in $P$ we will need two more sets of subscripts.

Let $p = (r_0, ..., r_n)$ be the initial pattern expression with $n$ regular patterns and variables $v_0, \ldots, v_{n-1}$, and let $P = (A_0, A_1, \ldots, A_n)$ be the corresponding pattern automaton, where $A_i = (Q_i, \Sigma \cup \{v_0, \ldots, v_{i-1}\}, \delta_i, q_{i,0}, F_i)$ are the modules of $P$ for $i \in \{0, \ldots, n\}$. We construct a pattern automaton $P'$ consisting of the following finite automata:

1. for all $i, j \in Q_B$:

   a. $A_{0,i,j} = \Big(Q_0 \times Q_B, \Sigma, (q_{0,0}, i), \delta_{0,B}, F_{0,j}\Big)$, where
   $\forall (p, l) \in Q_0 \times Q_B, \forall a \in \Sigma: \ \delta_{0,B}\big((p, l), a\big) = \big(\delta_0(p, a), \delta_B(l, a)\big)$,
   and $F_{0,j} = F_0 \times \{j\}$;

   b. for all $k \in \{1, \ldots, n-1\}$:

   $A_{k,i,j} = \Big(Q_k \times Q_B, \Sigma \cup \{v_{k',i',j'} | i', j' \in Q_B,$
   $k' < k\}, (q_{k,0}, i), \delta_{k,B}, F_{k,j}\Big)$, where

   $\forall (p, l) \in Q_k \times Q_B, \ a \in \Sigma: \ \delta_{k,B}\big((p, l), a\big) = \big(\delta_k(p, a), \delta_B(l, a)\big)$,
   $\forall k' < k, \forall p \in Q_k, \forall i', j' \in Q_B : \ \delta_{k,B}\big((p, i'), v_{k',i',j'}\big) = \big(\delta_k(p, v_{k'}), j'\big)$, and
   $F_{k,j} = F_k \times \{j\}$;

2. $A_{n,0,F_B} = \Big(Q_n \times Q_B, \Sigma \cup \{v_{k,i,j} | i, j \in Q_B, k < n\},$
   $(q_{n,0}, 0), \delta_{n,B}, F_{n,B}\Big)$, where
   $\forall (p, l) \in Q_k \times Q_B, \forall a \in \Sigma: \ \delta_{n,B}\big((p, l), a\big) = \big(\delta_n(p, a), \delta_B(l, a)\big)$,
   $\forall k < n, \forall p \in Q_n, \forall i, j \in Q_B: \ \delta_{n,B}\big((p, i), v_{k,i,j}\big) = \big(\delta_n(p, v_k), j\big)$,
   and $F_{n,j} = F_n \times F_B$.

Then, the sought pattern automaton $P'$ is obtained by ordering all the above automata as follows:

$$P' = (\ A_{0,0,0}, \ldots, A_{0,0,t}, A_{0,1,0}, \ldots, A_{0,1,t}, A_{0,2,0} \ldots, A_{0,2,t}, \ldots, A_{0,t,t},$$
$$A_{1,0,0}, \ldots, A_{n-1,t,t},\ A_{n,0,F_B}\ ),$$

where $t =\mid Q_B \mid$. We make the following observations which justify the correctness of our construction:

1. If in the pattern automaton $P'$ we consider only the first component of each state and ignore the extra subscripts (i.e., the above $i$ and $j$), we discover that a computation in $P$ for an input word $w$ is successful if and only if there exists a successful computation for $w$ in this reduced version of $P'$, since all automata $A_{k,i,j}$ are identical with $A_k$, for all $i, j$.

2. For $i, j \in Q_B$ denote by $B_{i,j}$ the automaton obtained from $B$ by setting $i$ to be the initial state and $j$ the only final state. Also denote $p_k = (r_0, \ldots, r_k)$ the pattern expression obtained from $p$ considering only the first $k+1$ patterns with $0 \leq k < n$, and similarly denote $P'_{k,i,j}$ to be the pattern automaton obtained from $P'$ considering only the automata from $A_{0,0,0}$ to $A_{k,i,j}$, for $0 \leq k < n$ and $i, j \in Q_B$. Then one can check by induction that

$$\forall i, j \in Q_B, \forall k \in \{0, n-1\} :\ \ L(p_k) \cap L(B_{i,j}) = L(P'_{k,i,j}).$$

3. If a word $w$ belongs to $L(p)$, then it can be factorized as $w = x_0 u_1 x_1 \ldots u_s x_s$, where we have all $x_i \in \Sigma^*$ and each $u_i \in L(p_r)$ for some $r \in \{0, \ldots, n-1\}$. The words $u_i$ are the substitution words for the variables in the pattern $r_n$ used for generating $w$. If $w$ belongs to $L(B)$ as well, then we have the following sequence:

   $x_0 \in B_{0,i_1},\ u_1 \in B_{i_1,j_1},\ x_1 \in B_{j_1,i_2},\ \ldots$
   $\ldots, x_{s-1} \in B_{j_{s-1},i_s},\ u_s \in B_{i_s,j_s},\ x_s \in B_{j_s,i_{s+1}}$, and $i_{s+1} \in F_B$.

   Since each $u_l$ also belongs to a language $L(p_t)$ for some $t \in \{0, \ldots, n-1\}$, we obtain $u_l \in L(p_t) \cap B_{i_l,j_l} = L(P'_{t,i_l,j_l})$. Using this relation, it can be checked that the automaton $A_{n,0,F_B}$ should accept $w$ as well, hence that $w \in L(P')$.

The details, as well as the reciprocal of the last observation are omitted, being too elaborated for the present space constraints. Thus, our automata system $P'$ recognizes the intersection between $L(P)$ and $L(B)$, proving that the intersection is a pattern expression language.

## 4 Limitations of Pattern Expressions

In this section we prove that although pattern expression languages are closed under the reverse operation, the language $L_4$ is not a pattern expression language. This result was the initial motivation of the paper and we address it here.

Let $\Sigma$ be an alphabet with at least 3 letters.

**Proposition 1.** *The language* $L = \left\{ uu^R \middle| u \in \Sigma^* \right\}$ *is not a pattern expression language.*

*Proof.* We assume by contradiction that the language $L$ is a pattern expression language, i.e., that there exist a pattern expression $p = (r_0, \ldots, r_n)$ such that $L = L(p)$. Let $a, b, c$ be three distinct letters of $\Sigma$. Invoking Theorem 2, it follows that $L_5 = L \cap (abc)^*(cba)^*$ is also a PE language.

It is easy to observe that $L_5 = \{(abc)^j(cba)^j | j \geq 0\}$. Since this language is a PE language, we can invoke the pumping lemma for PE languages (Lemma 1), which ensures that there exists a constant $N$ such that for all $w \in L_5$ with $|w| > N$, $w$ can be factorized as $w = x_0yx_1 \ldots x_{m-1}yx_m$ such that $m \geq 1$, $|x_0y| < N$, $| \ y \ | \geq 1$, and $w_i = x_0y^ix_1 \ldots x_{m-1}y^ix_m \in L_5$ for all $i \geq 0$.

Considering the word $w = (abc)^N(cba)^N \in L_5$ and a factorization of $w$ as stated by the pumping lemma, i.e., $(abc)^N(cba)^N = x_0yx_1 \ldots x_{m-1}yx_m$. It follows that $y$ is a sub-word of $(abc)^N$. We distinguish two cases: $y$ is also a sub-word of $(cba)^N$, and $y$ is not a sub-word of $(cba)^N$. In the first case, $y$ must be a letter since no word of two or more letters is simultaneously a sub word in of both $(abc)^N$ and $(cba)^N$. Since $x_0x_1 \ldots x_m \in L_5$, but $x_0x_1 \ldots x_m$ is obtained from $(abc)^N(cba)^N$, by deleting the same letter $m$ times, it follows that $x_0x_1 \ldots x_m \notin L_5$, contradiction. In the second case, the word $x_0x_1 \ldots x_m \in L_5$, but $x_0x_1 \ldots x_m$ is obtained from $(abc)^N(cba)^N$, by deleting only sub-words of $(abc)^N$, so $x_0x_1 \ldots x_m \notin L_5$, contradiction.

Reaching a contradiction, we conclude that $L_4$ is not a PE language.

Of course, one may ask whether it is necessary to use a ternary alphabet and if the result holds for binary alphabets (for unary alphabets the language $L_5$ becomes a regular language, thus it avoids the contradiction). The question remains open.

The closure of PE languages under the intersection with regular languages is a very powerful tool, in particular for proving that certain languages are not PE. The following results illustrate the technique.

**Proposition 2.**

1. *The following languages over an alphabet with at least three letters are not PE languages:*
   - $L_6 = \left\{ u \middle| u = u^R \right\}$ *(the language of palindromes),*
   - $L_7 = \left\{ \ ucv \ \middle| \ | \ u \ |_a + | \ u \ |_b = | \ v \ |_a + | \ v \ |_b \ \right\}$,
   - $L_8 = \left\{ \ w \ \middle| \ | \ w \ |_a = | \ w \ |_b = | \ w \ |_c \ \right\}$,
   - $L_9 = \left\{ \ uv \ \middle| \ | \ u \ |_a + | \ u \ |_b = | \ v \ |_c \ \right\}$.
2. *The following languages over an alphabet with at least two letters are not PE languages:*
   - $L_{10} = \left\{ \ w \ \middle| \ | \ w \ |_a = | \ w \ |_b \ \right\}$, $L_8 \subseteq \{a, b\}^*$,
   - $L_{11} = \left\{ \ w \ \middle| \ | \ w \ |_b = 2 \ | \ w \ |_a \ \right\}$, $L_9 \subseteq \{a, b\}^*$.

*Proof.* We observe that

$L_6 \cap (abc)^*(cba)^* = \left\{ \ (abc)^n(cba)^n \ \middle| n \geq 0 \right\}$, which is not a PE language ([3, Example 7]);
$L_7 \cap (a+b)^*c(a+b)^* = \left\{ \ \{a,b\}^n c\{a,b\}^n \ \middle| n \geq 0 \right\}$, which is not a PE language ([3]);
$L_8 \cap a^*b^*c^* = \left\{ \ a^n b^n c^n \ \middle| n \geq 0 \right\}$, which is not a PE language ([3]);
$L_9 \cap (a+b)^*c^* = \left\{ \ \{a,b\}^n c^n \ \middle| n \geq 0 \right\}$, which is not a PE language ([3, Example 8]);
$L_{10} \cap a^*b^* = \left\{ \ a^n b^n \ \middle| n \geq 0 \right\}$, which is not a PE language ([3]);
$L_{11} \cap a^*b^* = \left\{ \ a^{2n} b^n \ \middle| n \geq 0 \right\}$, which is not a PE language ([3, Example 7]);

If any of $L_6, \ldots, L_{11}$ was a PE language, so would be the corresponding intersections – a contradiction.

We also remark that some previous results which involved substantial effort to prove, such as Lemma 3 in [2], are direct consequence of this closure property of PE languages.

*Remark 2.* We remind that PE languages under the unary alphabet may be neither regular, nor context-free, as has been proven for the language $\{a^m \mid m \text{ is not prime } \}$ in [3].

## 5 Open Questions and Further Work

In the following we list a few problems remained unaddressed, and which are the subject of our proposed further work. They are mostly concerned with extensions of extended regex and pattern languages.

How much would the power of extended regex increase if we add a reverse operator? We expect that although more context-free languages would be expressed by such expressions, they will still not cover the entire family of context-free languages.

We propose the addition of the new operator "$*_i$": if we number the Kleene star operators in a PE or regex, we can use later the "same star" which will synchronize the number of its iterations with the number of iterations of the referenced star. For example, the language $\{a^n b^n \mid n \geq 0\}$ which is not a PE/regex language would be recognized this time by $p = (u = a^*, v = b^{*_1}, uv)$ or by the regex expression $(a^*b^{*_1})$. This addition seems still not to cover the entire family of context-free languages, since the language of all words which have a same number of $a$'s and $b$'s seems to be inexpressible by PE/regex together with these new additional operators.

The previous observation leads us to a third proposed addition, that of a shuffle operator. It would be interesting to investigate the power of these newly defined PE/regex.

We point out to a problem whose resolution is still uncertain, that is, whether PE and regex languages are the same family. As we mentioned in the introduction, there exist methods which allegedly convert one into another, however, no rigorous proof of equivalence has been provided yet. Finally, we intend to investigate further the relationship between PE and regex languages and other families of pattern-related languages. We envision a rigorous comparison between the present and past models.

## 6 Acknowledgments

We express acknowledgments to Dr. Max Burke for his suggestion on the use of language $L_5$.

## References

1. D. Angluin, Finding Patterns common to a set of strings. *Journal of Comput. System Sci.* 21 (1980) 46-62.
2. C. Câmpeanu, K. Salomaa, S. Yu: A Formal Study of Practical Regular Expressions, IJFCS, pp.
3. C. Câmpeanu and S. Yu: Pattern Expressions and Pattern Automata, IPL, pp.
4. C. Câmpeanu, "Regular languages and programming languages", *Revue Roumaine de Linguistique - CLTA*, 23 (1986), 7-10.
5. S. Dumitrescu, G. Paun, A. Salomaa, Pattern Languages versus Parallel Communicating Grammar Systems TUCS report 42, September 1996.
6. J. E.F. Friedl *Mastering Regular Expressions*, O'Reilly & Associates, Inc., Cambridge, 1997.
7. J.E. Hopcroft, R. Motwani, and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Reading Mass, 2006.
8. L. Kari, A. Mateescu, Gh. Păun, A. Salomaa, Multi-pattern languages, *Theoretical Computer Science*, **141** (1995), 253-268.
9. S. Kobayashi, V. Mitrana, G. Păun, and G. Rozenberg, Formal properties of PA-matching *Theoretical Computer Science*, Volume 262, Issues 1-2, 6 July 2001, Pages 117-131.
10. M.E. Lesk, "Lex - a lexical analyzer generator", *Computer Science Technical Report* (1975) 39, AT&T Bell Laboratories, Murray Hill, N.J.
11. C. Martín-Vide and V. Mitrana, Some undecidable problems for parallel communicating finite automata systems *Information Processing Letters*, 77 (2001), 239245.
12. C. Martín-Vide and V. Mitrana, Remarks on arbitrary multiple pattern interpretations *Information Processing Letters*, In Press, Corrected Proof, Available online 24 October 2006.
13. V. Mitrana and R. Stiebe, Extended finite automata over groups *Discrete Applied Mathematics*, Volume 108, Issue 3, 15 March 2001, Pages 287-300
14. V. Mitrana, G. Păun, G. Rozenberg, and A. Salomaa, Pattern systems *Theoretical Computer Science*, Volume 154, Issue 2, 5 February 1996, Pages 183-201(19)
15. A. Salomaa, *Theory of Automata*, Pergamon Press (1969), Oxford.
16. A. Salomaa, Formal Languages (Academic Press, New York, 1973).
17. S. Yu, Regular Languages. In: A. Salomaa and G. Rozenberg (ed.), *Handbook of Formal Languages*, Springer Verlag, 1997, 41–110.