On Pattern Expression Languages

Cezar Câmpeanu[1] and Nicolae Santean[2]

[1] Department of Computer Science and IT, University of Prince Edward Island
[2] School of Computer Science, University of Waterloo

Abstract. Pattern Expressions and Regex (practical regular expressions) have recently been the subject of scrutiny of formal language theorists, in an effort to provide a conceptual basis to the existing implementations of regular expressions (Perl, Awk, Python, etc). In this paper we address a standing question on pattern expressions (PE), namely whether the family of PE languages is closed under the intersection with regular languages or not. Since this family is not closed under complement, but is closed under reverse, another natural question has been frequently raised in the past years, on whether particular languages such as the mirror language and the language of palindromes are PE languages or not. We provide answers to these and other related questions, hoping to give an insight on the descriptional power of these language specifications.

Keywords: Pattern expression; Practical regular expression; Pattern automata system

1 Introduction

Regular expressions are powerful programming tools present in many language implementations such as Perl, Awk and Python, as well as in shells and other software utilities, like egrep, vi, and emacs. Despite a similar (and unfortunate) nomenclature, these "practical regular expressions" are quite different from their theoretical counterpart, namely the regular (or rational) expressions. Practical regular expressions [5] are often abbreviated "Regex", and although developed under the influence of theoretical ones, their formalism and descriptional power varies greatly across several environments. For example, Regex implemented in Lex ([9]) bare a strong similarity to regular expressions, whereas those found in Perl are significantly different. Perl Regex [5] can express the languages $L_1 = \{a^n ba^n \mid n \geq 0\}$ and $L_2 = \{ww \mid w \in \{a, b\}^*\}$ (the language of squares). However, Perl Regex and pattern expressions (PE, for brevity – a different formalism for Regex) cannot express the language $L_3 = \{a^n b^n \mid n \geq 0\}$. Even more intriguing, despite the recent theoretical studies on Regex and PE ([2, 3]), some closure properties, as well as the membership to the family of PE languages of $L_4 = \{ww^R \mid w \in \{a, b\}^*\}$ (the mirror language), $L_5 = \{(abc)^n(cba)^n \mid n \geq 0\}$, $L_6 = \{w \mid w = w^R, w \in \{a, b\}^*\}$ (the language of palindromes), and other anthological languages, have remained unsolved. This theoretical gap has lead to the status quo of accepting these ubiquitous programming tools without an elementary understanding of their descriptional capabilities.

Regex are relatively easy to use; for example, L_1 can be expressed in Perl by the Regex $(a*)b\backslash 1$, and L_2 by $((a|b)*)\backslash 1$ – the operator "\1" is a reference to (copy of) the content of

the first pair of parentheses. Unlike L_3, which has been proven that cannot be generated in Perl ([2]) and is not a PE language ([3]), very little is known about L_4. There has been a long-standing controversy, on whether L_4 can or cannot 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 regular expressions and pattern expression fail to give a contradiction for L_4 and similar languages. Thus, the present study, which solves this dilemma among others, is expected to raise the interest of both theoreticians as well as practitioners.

In this paper we adopt the formalism of pattern expressions. One reason for this preference is that they seem to be more versatile, and they avoid some semantical ambiguities pointed out in [3]. In [3] can be found a method for converting a Regex into a pattern expression and vice-versa. Related to our study on pattern expressions (PE) we mention [1], where was considered the addition of a reverse operator to extend the power of pattern languages, or [4], 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 [7, 12, 4, 8] and more recently in [8, 11] which give a comprehensive survey on the topic as well. 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 generative 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, here we have used pattern automata introduced in [3], which bear similarities with parallel communicating finite automata systems mentioned in [10]. 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 reason for this status, 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 tremendous 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:

– 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 [7]) do not contain a Kleene operator (the word "iteration" refers to repeated substitutions), in contrast with pattern expressions whose definition uses the Kleene operator.
- A 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 apparent when their families of languages are compared to those in the Chomsky hierarchy.

In some sense, one can view the idea behind pattern expressions as a combination of the concepts used in multi-patterns and iterated patterns. Yet, we do not know whether combining these models in some way one can obtain a scheme equivalent with the PE formalism.

2 Notations and Definitions

In this section we provide some basic notions and notations used throughout the paper. Omitted definitions of elementary formal language concepts can be found in [6, 13, 15].

An alphabet Σ is a finite non-empty set. A word over Σ is an element of the free monoid Σ^*, that is, a finite string of symbols (letters) in Σ. For a word $w \in \Sigma^*$ we denote by $|w|$ its length, i.e., the total number of symbols in w, and by $|w|_a$ the number of occurrences of the letter a in w. The word with no letters (the empty word) is denoted by λ, and $|\lambda| = 0$. We use the notation $u \preceq v$ to denote that u is a subword of v (we have $\lambda \preceq v$ and $v \preceq v$).

A regular expression over Σ is the set of all well-formed parenthesized infix formulae obtained from the elements of Σ^* (viewed as atomic formulae), the null operator λ, the binary operators $+$ and $\cdot$ (expressed as juxtaposition), and the unary operator *. The language of a regular expression e is denoted by $L(e)$ and is defined as in [6]. If w is a word in $L(e)$, we say that "w matches the regular expression e".

Definition 1. *Let Σ 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 Σ;*
2. *for $i \in \{1, \ldots, n\}$, r_i is a regular pattern 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 $i \in \{1, \ldots, n\}$:

$L_i = \{(u_0/v_0) \ldots (u_{i-1}/v_{i-1})u_i \mid u_j \in L_j \text{ for } 0 \leq j \leq i-1, \text{ and } u_i \in L(r_i)\}$,

where the notation $(u_0/v_0)\dots(u_k/v_k)u$ expresses the substitution of all occurrences of variable v_j in u by the word u_j, for all $0 \leq j \leq k$. By definition, $L(p) = L_n$.

For better handling pattern expressions, we use the notation $p = (v_0 = r_0, v_1 = r_1, \dots, v_{n-1} = r_{n-1}, r_n)$ 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.

- for $p = (v_0 = a^*, v_0 b v_0)$, $L(p) = \{a^n b a^n \mid n \geq 0\}$;
- for $p = (v_0 = ab^*, v_0^* c v_0)$, $L(p) = \{(ab^n)^m c a b^n \mid n \geq 0, m \geq 0\}$;
- for $p = (v_0 = ab^*, v_1 = baa^*, (v_0 + v_1)(v_0 + v_1))$ we have
  $L(p) = \{ab^n ab^n \mid n \geq 0\} \cup \{ba^n ba^n \mid n \geq 1\} \cup$
  $\{ab^n ba^m \mid n \geq 0, m \geq 1\} \cup \{ba^m ab^n \mid n \geq 0, m \geq 1\}$.

In [3] can be found a method for converting a pattern expression into an extended regular expression and vice-versa. Extended regular expressions are essentially the Regex constructs in Perl (the so-called practical regular expressions), and are defined as regular expressions which accept the additional atoms "$\backslash n$", denoting "back-references". 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 regular expressions we refer the reader to [5]. We also emphasize that pattern expressions are extensions of patterns ([1]), i.e., words containing letters and variables. A pattern language([1, 12]) is obtained from a pattern by substituting variables with arbitrary words.

Remark 1. ([3]) It is clear that regular languages are PE languages, and it has been shown that PE languages are context-sensitive. Context-free languages and PE languages are incomparable – these families have a nonempty symmetric difference. PE languages are closed under reverse and homomorphism, and are not closed under complement, inverse homomorphism and finite substitution. PE languages under the unary alphabet may be neither regular, nor context-free, as the language $\{a^m \mid m$ is not prime $\}$ proves it.

Lemma 1. *(Pumping Lemma [3]) Let L be a pattern expression language or a Regex language. There exists a constant N, such that any word $w \in L$, $|w| > N$, has a decomposition $w = x_0 y x_1 y x_2 \cdots x_m$ for some $m \geq 1$, such that:*

1. $|x_0 y| \leq N$,
2. $|y| \geq 1$,
3. $x_0 y^j x_1 y^j x_2 \cdots x_m \in L$, *for all $j > 0$ (notice that y may <u>not</u> be deleted).*

We now recall the notion of pattern automaton, introduced in [3]. A `pattern automaton` (PA) is an automata system $P = (A_0, A_1, \dots, A_n)$ 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 operates over Σ, and for each $i \in \{1, \ldots, n\}$, A_i has the same structure as A_0, except for the transition labels which may eventually be 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 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 $U = (U_i)_{0 \leq i < n}$, whose stacks store elements of $\{0, 1\}$, and an array $V = (V_j)_{0 \leq j < n}$, whose stacks store elements of Σ^*. 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 involving a transition labeled v_i consists of matching a prefix of the remaining input 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. One can observe that all stacks in 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 in 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 in 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 automaton 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 $i \in \{1, \ldots, n-1\}$,

$W_i = \{(u_0/v_0) \ldots (u_{i-1}/v_{i-1})u_i \mid u_i \in R_i, u_j \in W_j, 0 \leq j \leq i-1\}$.

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, we question whether is possible to construct some device that has a better running time than PA. The following theorem gives the answer, with a surprising practical implication: *Perl Regex constructs can not operate efficiently, regardless of their implementation in Perl.*

Theorem 1. *The membership problem for pattern expressions is NP-complete and has $O(n^2m)$ space complexity, where n is the number of regular patterns of the pattern expression and m is the length of the input word.*

Proof. We analyze the membership problem for pattern automata. Let P be a pattern automaton as previously defined, and w be 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), it takes $O(|w|)$ time to recognize w; thus, 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.

The space complexity results from the fact that all stack elements (words) used for simulating a pattern automaton have a length bounded by the length of the input word w (storing some subwords of w) and there are $2n + 1$ stacks, each of depth at most n. □

3 Main Result

In the previous section we presented a pumping lemma for PE languages (and for extended regular expression languages). This lemma turns out to be too weak for proving that a

language like L_4 (mentioned in introduction) is not a PE language. To alleviate this problem, as well as for other theoretical and practical reasons, we first prove an important closure property for the family of pattern expression languages, which has remained hidden.

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

Proof. Let $L = L(p)$ with $p = (r_0, r_1, \ldots, r_n)$ be a pattern expression language and R be a regular language accepted by a trim DFA $B = (\Sigma, Q_B, 0, \delta_B, F_B)$. 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. The simulation goes in parallel when P transits from state to state based on letters in Σ. When P meets 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 Σ, B is revived and advances again in parallel with P. *This idea is facing the challenge of designing this simulator as a pattern automaton.* The problem turned to be rather difficult, and we will see that the newly constructed pattern automaton uses significantly more variables than P – number increase (multiplied) of order $O(\mid Q_B \mid^2)$. One essential technique used in the proof is to index the newly introduced variables in such manner, that the subscripts themselves provide information on where the run of B has paused or where it should resume from, in terms of the states of B. Hence, we anticipate that beside the normal indexing of variables in P we 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:
$$x_0 \in B_{0,i_1}, \ \ u_1 \in B_{i_1,j_1}, \ \ x_1 \in B_{j_1,i_2}, \ \ \ldots$$
$$\ldots, x_{s-1} \in B_{j_{s-1},i_s}, \ \ u_s \in B_{i_s,j_s}, \ \ x_s \in B_{j_s,i_{s+1}}, \text{ and } i_{s+1} \in F_B.$$
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 to fit the present space constraints. *The construction and behavior of P' goes beyond an automaton cross product, P' having $(n+1)|Q_B|^2$ modules with a total of $|Q_B|^3|Q|$ states.*

Thus, the newly constructed 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 use Theorem 2 to prove that a few remarkable languages, such as the mirror language (L_4), are not pattern expression languages, despite the fact that the family of PE languages is closed under the reverse operation. We first prove the easier case, that of alphabets with at least three letters, and we start with a preliminary result.

Lemma 2. *The language $L_5 = \{(abc)^n(cba)^n \mid n \geq 0\}$ is not a pattern expression language.*

Proof. Assume by contrary, that L_5 is a PE language. Invoking the pumping lemma (Lemma 1), there exists a constant N such that any word $w \in L_5$ with $|w| > N$ can be factorized as $w = x_0yx_1 \dots x_{m-1}yx_m$ such that $m \geq 1$, $|x_0y| \leq N$, $\mid y \mid \geq 1$, and $w_i = x_0y^ix_1 \dots x_{m-1}y^ix_m \in L_5$, for all $i \geq 1$.

Let $w = (abc)^n(cba)^n$ with $3n > N$, and consider a factorization of w as above. It is clear that y can not consist of only one letter, since otherwise $w_3 \notin L_5$. If $|y| = 2$, then y can be one of the following words: ab, bc, ca, cc, cb, ba or ac, and one can check that $y^2 \not\preceq (abc)^n(cba)^n$ for any $n \geq 0$. It remains the case when $|y| \geq 3$.

We first notice that $y \preceq (abc)^n$, since $|x_0y| \leq N$. We also observe that y can only be in one of the following forms: bxa, cxb, axc, bxc, axb or cxa, with x a nonempty word (otherwise, $y^3 \not\preceq (abc)^n(cba)^n$, for any $n \geq 0$).

If y is either bxc, axb or cxa, then clearly y^2 should cross the middle of w_2, since it has one of the subwords cb, ba, or ac (found only in the second half), and the first occurrence of y is in the first half of w. Thus y^2 must produce a cc, and y^4 must produce two such groups – a contradiction.

It remains that y must be of the form bxa, cxb, or axc, and then, no occurrence of y can be in the second half of w. Indeed, y can not cross the middle of w, and it can not be completely in the second half of w since y^2 produces one of the sequences ab, bc, or ca found only in the first half. Then there exists k sufficiently large such that one factor y^k will cross the middle of w_k, since for each pumped y, the first half of w increases with $|y|$ symbols, whereas the middle shifts to the right $|y|/2$ positions. But this leads to a contradiction yet again, since once y^k has produced cc, y^{2k} will produce two such groups. Having exhausted all possibilities, we conclude that L_5 is not a PE language. □

Corollary 1. *Let Σ be an alphabet with at least three letters. Then the language $L'_4 = \{ww^R \mid w \in \Sigma^*\}$ is not a pattern expression language.*

Proof. We assume by contradiction that L'_4 is a pattern expression language, i.e., that there exists a pattern expression $p = (r_0, \dots, r_n)$ such that $L'_4 = L(p)$. Let a, b, c be three distinct letters of Σ. Invoking Theorem 2, it follows that $L_5 = L \cap (abc)^*(cba)^*$ is also a PE language. This contradicts Lemma 2. □

In the following we prove an analogue result for the binary alphabet, this time using a more intricate combinatorial apparatus. The idea is to take the language $L_9 =$

$\{(aababb)^n(bbabaa)^n \mid n \geq 0\}$ and prove that is not a pattern expression language. We first prove two useful combinatorial lemmas.

Lemma 3. *Let $v \in \{a,b\}^*$ be such that $v \preceq (aababb)^2$ and $0 < |v| < 6$. Then $v^k \not\preceq (aababb)^n$, for all $n \geq 0$ and $k > 2$.*

Proof. Let us list all the subwords v of $(aababb)^2$ with $0 < |v| < 6$: a and b, of length 1; ab, ba, bb and aa, of length 2; aab, aba, bab, abb, bba and baa, of length 3; $aaba$, $abab$, $babb$, $abba$, $bbaa$ and $baab$, of length 4; $aabab$, $ababb$, $babba$, $abbaa$, $bbaab$ and $baaba$, of length 5. One may check that if v is any of these subwords, then $v^3 \not\preceq (aababb)^n$, for any $n \geq 0$. For example, if $v = baaba$, then $v^3 = baababaababaaba$, and we observe that any two occurrences of b are separated by a's; nevertheless, this is not true for $(aababb)^n$. Thus, v^3 can not be a subword of $(aababb)^n$, therefore we have that $v^k \not\preceq (aababb)^n$, for all $n \geq 0$ and $k > 2$. □

Lemma 4. *Let $n \geq 0$ and $v \in \{a,b\}^*$ be such that $v \preceq (aababb)^n$. If $|v| \geq 6$, then $v \not\preceq ababb(bbabaa)^m$ for all $m \geq 0$.*

Proof. We observe that any subword of $(aababb)^n$, of length at least 6, must have as a prefix one of the following words: *aababb*, *ababba*, *babbaa*, *abbaab*, *bbaaba* and *baabab*. We also notice that each of these prefixes have one of the following subwords: *aaba*, *bbaa* and *babba*. However, one may check that none of these three subwords is a subword of $ababb(bbabaa)^m$, for any $m \geq 0$. For example, *aaba* can not be a subword, since in $ababb(bbabaa)^m$ any double occurence of a is followed by two b's. Thus, the conclusion follows. □

Theorem 3. *Let Σ be an alphabet with at least two symbols. The language $L_4 = \{ww^R \mid w \in \Sigma^*\}$ is not a PE language.*

Proof. The language $L_4 \cap \{(aababb)^n(bbabaa)^m \mid n, m \geq 0\}$ can be written as $L' = \{(aababb)^n(bbabaa)^n \mid n \geq 0\}$. Indeed, the middle of a word $w = (aababb)^i(bbabaa)^j$ must divide w in two factors of equal length, multiple of 3. Since w is a "mirror word", one can check that the middle of w must separate precisely the iterations of *aababb* from those of *bbabaa*.

Invoking the closure of PE languages to intersection with regular languages, it suffices to prove that L' is not a PE language. Assume by contradiction that L' is a PE language. Then, by applying the pumping lemma, there exists a constant N such that any word $w \in L'$ of length greater than N has a decomposition $w = x_0yx_1y \dots x_{t-1}yx_t$ for some $t \geq 1$, such that $|x_0y| \leq N$, $|y| \geq 1$, and $w_k = x_0y^kx_1y^k \dots x_{t-1}y^kx_t \in L'$, for all $k > 0$.

The pumping lemma applies for the word $w = (aababb)^n(bbabaa)^n$ with $6n > N+6$, and w has a decomposition as above. We observe that the factor y must be a proper subword of $(aababb)^n$ (since $|y| \leq N$). We distinguish the following two choices: $|y| < 6$ or $|y| \geq 6$.

If $|y| < 6$, we have that $y \preceq (aababb)^2$, thus we can apply Lemma 3 and infer that $y^3 \not\preceq (aababb)^m$, for any $m > 0$. Notice that pumping two extra y after x_0 in w, would

shift the middle of w at most $|y| < 6$ symbols to the left (at most half of how much was pumped), thus the resulting subword y^3 would still be in the first half of w_3 (since y occurs within the first N symbols of w, the middle of w is beyond $N + 6$, y^3 occurs within the first $N + 2|y|$ symbols, and the middle of w_3 is beyond $N + |y| + 6$). This is in contradiction with the fact that y^3 is a subword of $(aababb)^m$. Thus, $w_3 \notin L'$, contradicting the pumping lemma.

If $|y| \geq 6$, we first show that all occurrences of y must necessarily be in the first half of w. We have that $y \preceq (aababb)^n$ by $|x_0 y| \leq N$; if a subsequent occurrence of y was spanning across the middle of w, then y would have b^3 as a subword. Since this is impossible, we infer that no occurrence of y can cross the middle of w. Furthermore, since $y \preceq (aababb)^n$ and $|y| \geq 6$, by applying Lemma 4 it follows that $y \not\preceq ababb(bbabaa)^n$, thus y cannot occur in the second half of w either. It follows that all possible occurrences of y are in the first half of w. Notice that in this case, for each y "pumped", the middle of the word shifts to the right $|y|/2$ symbols (the word expands twice as fast as the displacement of its middle). Thus, for k sufficiently large, there must be an occurrence of y which goes beyond the middle of w_k. This leads to a contradiction by the same arguments used in the first place to prove that y must occur within the first half of w.

We have obtained a contradiction in both cases, thus L' is not a PE language, and by Theorem 2 it follows that L_4 can not be a PE language either. □

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 this technique. We use a, b, c for letters and u, v, w for words.

Corollary 2. *The following languages are not pattern expression languages:*

$L_6 = \{w \mid \ w = w^R\}$, *(the language of palindromes)*
$L_7 = \{w \mid \ |w|_a = |w|_b\}$, *(the language of balanced words)*
$L_8 = \{w \mid \ |w|_b = 2|w|_a\}$, *(the language of semi-balanced words)*
$L_9 = \{w \mid \ |w|_a = |w|_b = |w|_c\}$,
$L_{10} = \{w \mid \ |w|_a + |w|_b = |w|_c\}$,
$L_{11} = \{ucv \mid \ |u|_a + |u|_b = |v|_a + |v|_b\}$.

Proof. We observe that: $L_6 \cap (aababb)^*(bbabaa)^* = \{(aababb)^n(bbabaa)^n \mid n \geq 0\}$, which is not a PE language (Lemma 2); $L_7 \cap a^*b^* = \{a^n b^n \mid n \geq 0\}$, which is not a PE language ([3]); $L_8 \cap a^*b^* = \{a^{2n}b^n \mid n \geq 0\}$, which is not a PE language ([3, Example 7]); $L_9 \cap a^*b^*c^* = \{a^n b^n c^n | n \geq 0\}$, which is not a PE language ([3]); $L_{10} \cap (a+b)^*c^* = \{\{a,b\}^n c^n \mid n \geq 0\}$, which is not a PE language ([3, Example 8]); and $L_{11} \cap (a+b)^*c(a+b)^* = \{\{a,b\}^n c\{a,b\}^n \mid n \geq 0\}$, which is not a PE language ([3]).

If any of $L_6, \ldots, L_{11}$ was a PE language, so would be its corresponding intersection, leading to a contradiction. □

Finally, we should mention that some previous results involving several pages of elaborate proofs, such as Lemma 3 in [2], are trivially validated by the closure property of PE languages under the intersection with regular languages, as given by Theorem 2.

5 Conclusion

In this paper we used pattern automata systems to prove the closure of pattern expression languages under the intersection with regular languages. This property turned out to be a very useful tool in showing that several anthological languages, such as the mirror language, the language of palindromes or the language of balanced words, are not PE, thus revealing some of the limitations of pattern expression languages unforeseen before. Incidentally, we have also raised a warning on the inefficiency of the membership testing for pattern expressions. Due to the parallel between pattern expressions and Perl Regex, these results reach out beyond theory, to programmers and other users of practical regular expressions.

6 Acknowledgments

We express acknowledgments to Dr. Max Burke for suggesting the 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 and S. Yu: A Formal Study of Practical Regular Expressions. *IJFCS*, **14(6)** (2003) 1007–1018.
3. C. Câmpeanu and S. Yu: Pattern Expressions and Pattern Automata. *IPL*, **92** (2004) 267–274.
4. S. Dumitrescu, G. Păun and A. Salomaa: Pattern Languages versus Parallel Communicating Grammar Systems. *TUCS Report*, **42** September 1996.
5. J.E.F. Friedl: *Mastering Regular Expressions*, O'Reilly & Associates, Inc., Cambridge, (1997).
6. J.E. Hopcroft, R. Motwani, and J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Reading Mass, (2006).
7. L. Kari, A. Mateescu, G. Păun, and A. Salomaa: Multi-Pattern Languages. *Theoretical Computer Science* **141** (1995) 253–268.
8. S. Kobayashi, V. Mitrana, G. Păun, and G. Rozenberg: Formal Properties of PA-matching. *Theoretical Computer Science*, **262(1-2)** (2001) 117–131.
9. M.E. Lesk: Lex - a Lexical Analyzer Generator. *Computer Science Technical Report*, AT&T Bell Laboratories, Murray Hill, N.J, **39** (1975).
10. C. Martín-Vide and V. Mitrana: Some Undecidable Problems for Parallel Communicating Finite Automata Systems. *Information Processing Letters*, **77** (2001) 239–245.
11. C. Martín-Vide and V. Mitrana: Remarks on Arbitrary Multiple Pattern Interpretations. *Information Processing Letters*, in press, available online 24 October 2006.
12. V. Mitrana, G. Păun, G. Rozenberg, and A. Salomaa: Pattern Systems. *Theoretical Computer Science*, **154(2)** (1996) 183–201.
13. A. Salomaa: *Theory of Automata*. Pergamon Press, Oxford, (1969).
14. A. Salomaa: *Formal Languages*. Academic Press, New York, (1973).
15. S. Yu: *Regular Languages*. In: A. Salomaa and G. Rozenberg (eds.), *Handbook of Formal Languages*, Springer Verlag (1997) 41–110.