Acta Informatica (2009) 46:193–207
DOI 10.1007/s00236-009-0090-y

ORIGINAL ARTICLE

On the closure of pattern expressions languages under intersection with regular languages

Cezar Câmpeanu · Nicolae Santean

Received: 27 February 2008 / Accepted: 26 January 2009 / Published online: 24 February 2009

Abstract 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. Since this family is not closed under complement, but is closed under reversal, another natural question has frequently been raised in the recent years, on whether particular languages such as the mirror language and the language of palindromes are PE languages. We give answers to these and other related questions as well, thus providing an insight on their descriptional power.

1 Introduction

Pattern expressions, PE for brevity, were introduced in [5] as an alternative to the formalism of the regular expressions, which are also known as *regex* in practice. Regular expressions are powerful programming tools available 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 nomenclature, these "practical regular expressions" [7] are quite different from their theoretical counterpart, namely the regular (or rational) expressions.

Despite being developed under the influence of theoretical regular expressions, the regex formalism and its descriptional power differ greatly across various environments. For example, regex implemented in Lex [11] bare a strong similarity to regular expressions, whereas those found in Perl [7] are significantly different. Perl regex are relatively easy to use; for example, $L^{(1)} = \{a^n b a^n \mid n \geq 0\}$ can be expressed in Perl by the regex $(a*)b\backslash 1$, and $L^{(2)} = \{ww \mid w \in \{a, b\}^*\}$ (the language of squares) can be expressed by

C. Câmpeanu (✉)
Department of Computer Science and IT, University of Prince Edward Island, Charlotte, Canada
e-mail: ccampeanu@upei.ca

N. Santean
Department of Computer and Information Sciences, Indiana University South Bend, South Bend, USA
e-mail: nsantean@iusb.edu

$((a|b)*)\backslash 1$—where the operator "$\backslash 1$" is a reference to (copy of) the content of the first pair of parentheses. Extended regular expressions are essentially the regex constructs in Perl, 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.

In [5] is presented a method for converting a pattern expression into an extended regular expression and vice-versa, for certain situations. Although Perl regex can express languages such as $L^{(1)}$ and $L^{(2)}$, Perl regex and pattern expressions cannot express the language $L^{(3)} = \{a^n b^n \mid n \geq 0\}$. Even more intriguing, despite the recent efforts on the study of regex and PE [3,5], some of their closure properties, as well as the nature 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 capability.

Unlike $L^{(3)}$, which has been proven that cannot be generated in Perl [3] and that is not a PE language [5], 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 write a Perl regex for it, whereas some others believe the opposite, yet they cannot provide a rigorous argument to support their claim. For the latter, the difficulty consists in the fact that both pumping lemmas for extended regular expressions and pattern expressions fail to give a contradiction for $L^{(4)}$ and languages alike. Thus, the present study, which incidentally solves this dilemma for pattern expressions, is expected to raise the interest of theorists as well as of programmers.

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 semantic ambiguities pointed out in [5]. Related to our study on pattern expressions we mention [2], where the power of pattern languages is extended by the introduction of a reversal operator, or [6], where a mirror operation is introduced for increasing the power of multi-pattern languages. Other variations on multi-pattern languages or similar constructs can be found in [6,9,10,14] and more recently in [10,13], 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 generative devices. It would be interesting and rather challenging to analyze the relationship between the formalism proposed and developed in [3,5] (and used throughout this paper) and those employed in the past. For example, here we use the pattern automata introduced in [5], which bear similarities with parallel communicating finite automata systems mentioned in [12]. It is our belief that the two models are not equivalent, matter which we plan to address in the future. Several parallels can be drawn between PE languages and other families of languages studied in the past. Despite their proximity, we could not identify a previous 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 strong impact on the behavior of the model. A conceptual difference between PE 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 (formulae) and automata (acceptors), whereas the previous work originated in the study of grammars (generators), and had only a purely theoretical justification.

To layout the difference between our work and that on patterns, multi-patterns, parallel communicating automata systems, etc., that can be found in [2,6,9,12,14] and more recently in [10,13], we emphasize some differences between PE and the 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 performed with words in a context-sensitive language, for example.
- 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 [9]) do not contain a Kleene operator (the word "iteration" refers rather to repeated substitutions), in contrast with pattern expressions whose definition includes intrinsically the Kleene operator.
- Arguably, a model that seems to be 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 language families 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 one way or another would lead to a scheme equivalent to that of PE.

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 [8,15–17].

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 ε, and $|\varepsilon| = 0$. We use the notation $u \preceq v$ to denote that u is a subword of v (we have $\varepsilon \preceq v$ 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 nullary operator ε, 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 [8]. If w is a word in $L(e)$, we say that "w matches the regular expression e".

Throughout the paper we have dealt with a number of languages, that we denoted using superscripts from 1 to 16. To help the reader, we have added below a legend for all these languages and their definitions:

$$
\begin{aligned}
L^{(1)} &= \{a^n b a^n \mid n \geq 0\} \\
L^{(2)} &= \{ww \mid w \in \{a, b\}^*\} \\
L^{(3)} &= \{a^n b^n \mid n \geq 0\} \\
L^{(4)} &= \{ww^R \mid w \in \{a, b\}^*\} \\
L^{(5)} &= \{(abc)^n (cba)^n \mid n \geq 0\} \\
L^{(6)} &= \{w \mid w = w^R, w \in \{a, b\}^*\}
\end{aligned}
$$

$$
\begin{aligned}
L^{(7)} &= \{w \in \{a,b\}^* \mid w = a^n b^n a^n b^n, n \geq 0\} \\
L^{(8)} &= \{w \mid w = a^n b^m a^n b^m, n, m \geq 0\} \\
L^{(9)} &= \{w \mid |w|_a = |w|_b\} \\
L^{(10)} &= \{(aababb)^n (bbabaa)^n \mid n \geq 0\} \\
L^{(11)} &= \{w \mid |w|_a = |w|_b\} \\
L^{(12)} &= \{w \mid |w|_b = 2|w|_a\} \\
L^{(13)} &= \{w \mid |w|_a = |w|_b = |w|_c\} \\
L^{(14)} &= \{w \mid |w|_a + |w|_b = |w|_c\} \\
L^{(15)} &= \left\{ucv \mid |u|_a + |u|_b = |v|_a + |v|_b,\ u, v \in \{a,b,c\}^*\right\} \\
L^{(16)} &= (L(p) \sqcup L(a^*)) \cap L((baa)^* c (ba)^*),\ p = (ucu, u = b^*)
\end{aligned}
$$

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 $(u_0/v_0) \ldots (u_k/v_k)u$ 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 $p = (v_0 = r_0, v_1 = r_1, \ldots, v_{n-1} = r_{n-1}, r_n)$ 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 –
- 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
  $L(p) = \{ab^n ab^n \mid n \geq 0\} \cup \{ba^n ba^n \mid n \geq 1\} \cup$
  $\{ab^n ba^m \mid n \geq 0, m \geq 1\} \cup \{ba^m ab^n \mid n \geq 0, m \geq 1\}$.

We also emphasize that pattern expressions are extensions of patterns [2], i.e., words containing letters and variables (a pattern language [2,14] is obtained from a pattern by substituting variables with arbitrary words).

Remark 1 [5] 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 reversal and homomorphism, and are not closed under complement, inverse homomorphism and finite substitution. Furthermore, PE languages over an unary alphabet may be neither regular, nor context-free, as the PE language $\{a^m \mid m$ is not prime $\}$ proves it.

Lemma 1 (Pumping Lemma for PE [5]) *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* $j > 0$ (*note that* y *cannot be deleted*).

A *pattern automaton* (PA for short, introduced in [5]) is an automata system $P = (A_0, A_1, \ldots, A_n)$, where

(i) $A_0 = (Q_0, \Sigma, \delta_0, q_{0,0}, F_0)$, and
(ii) $A_i = (Q_i, \Sigma \cup \{v_0, \ldots, v_{i-1}\}, \delta_i, q_{i,0}, F_i)$, $0 < i \leq n$,

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 designate one of the variables $v_0, \ldots, v_{i-1}$. 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 of stacks $V = (V_j)_{0 \leq j < n}$, 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 $x^t = \varepsilon$.

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

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

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 $push(S, q)$, $push(V_j, \varepsilon)$, $push(U_k, 0)$ 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 $x^t = \varepsilon$, then accept.

Note that the behavior of the pattern automata system is non-deterministic, in that for a given configuration of the system, each of the transitions outlined above can potentially be chosen at every step of the computation, when the corresponding conditions are met.

The language recognized by P is:

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

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

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

where

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

and

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

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.

Note that the PA behavior is non-deterministic, leading to believe that the membership problem takes exponential run-time. 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.

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 given 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 proper variable substitutions and avoid 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 [2, 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. □

A similar result was obtained by Aho for regex in [1]. Note that Theorem 1 has an important implication: any application/program that uses practical regular expression is most likely inefficient. This fact is arguably ignored by some programmers, when evaluating the computational complexity of algorithms that use PE as programming tools.

3 Intersection with context free and regular languages

3.1 Intersection with context free languages

We first tackle the intersection with context free (CF) languages. It is easy to see that the language $L^{(7)} = \{w \in \{a,b\}^* \mid w = a^n b^n a^n b^n, n \geq 0\}$ is not a context free language, simply by applying the Bar-Hillel lemma. Moreover, we can also prove that this language is not in PE:

Lemma 2 *The language $L^{(7)} = \{w \in \{a,b\}^* \mid w = a^n b^n a^n b^n, n \geq 0\}$ is not a pattern expression language.*

Proof Considering a word $w = a^n b^n a^n b^n$, with n "large enough", there exists a subword y of w such that pumping y into w as described in Lemma 1 would produce words in the language. For y we have the following possibilities:

1. $|y|_a > 0$ and $|y|_b > 0$, thus we have arbitrary many alternations of a and b;
2. $|y|_a = 0$ or $|y|_b = 0$, thus we do not have the same numbers of a's and b's.

Both situations lead to words not belonging to the language, thus contradicting the pumping lemma. Hence $L^{(7)}$ cannot be in PE. □

Theorem 2 *The family of PE languages is not closed under intersection with context-free languages.*

Proof We consider the following PE: $r = (wxwx, w = a^*, x = b^*)$, generating the language $L^{(8)} = \{w \mid w = a^n b^m a^n b^m, n, m \geq 0\}$ and the context free language $L^{(9)} = \{w \mid |w|_a = |w|_b\}$. We can see that $L^{(7)} = L^{(8)} \cap L^{(9)}$, but $L^{(7)}$ is not PE. □

Incidentally, this also proves that the family of context-free languages is not closed under the intersection with PE languages.

3.2 Intersection with regular languages

In Sect. 2 we have presented a pumping lemma for PE languages (which also holds for regex languages). This lemma turns out to be too weak for proving that a language like $L^{(4)}$ (mentioned in the Sect. 1) is not a PE language. To alleviate this problem, as well as for other theoretical and practical reasons, we prove this important closure property for the family of pattern expression languages, which has remained hidden.

Theorem 3 *The family of pattern expression languages is closed under the 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 $B = (Q_B, \Sigma, \delta_B, 0_B, F_B)$. We consider a pattern automaton P, such that $L(P) = L(p)$, and construct a pattern automaton P' 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 out to be rather difficult, mainly because of

over exponential size blow-up, and we will see that the constructed pattern automata system for the intersection uses significantly more variables than P.

One essential technique used in the following proof is to index the variables of the constructed PA in such manner, that the subscripts themselves provide information on where the run of B has paused at, or where it should resume from (in terms of the states of B), and on the value of instantiated variable. Hence, we anticipate that beside the normal indexing of variables in P, we need two more sets of subscripts.

In order to construct a PA for the intersection, we need to consider the family of functions $f_w : Q_B \to Q_B$ defined as $f_w(q) = \delta_B(q, w)$. Since Q_B is finite, the number of functions $\{f_w \mid w \in \Sigma^*\}$ is finite. These functions, together with composition and with f_ε as identity, form a finite monoid: the transition monoid T_B of B.

We partition Σ^* into equivalent classes, given by the equivalence of finite index: $w_1 \equiv w_2 \Leftrightarrow f_{w_1} = f_{w_2}$. We denote $W = \Sigma^*/\equiv$, the quotient of Σ^* under $\equiv$, and denote the functions in T_B by $\{f_c\}_{c\in W}$. Then, $W = \{c_1, \ldots, c_m\}$, with $c_1, \ldots, c_m \in \Sigma^*$ being chosen representatives for the equivalence classes of W. Thus, if $w_1, w_2 \in c$, then for all $i \in Q_B$,

$$\delta_B(i, w_1) = j \quad \text{iff} \quad \delta_B(i, w_2) = j, \quad \text{thus } f_{w1} = f_{w2}. \tag{1}$$

Note that we may have $\delta_B(i, w_1) = \delta_B(i, w_2)$, for $w_1 \in c_1$, $w_2 \in c_2$, $c_1, c_2 \in W$, and $c_1 \neq c_2$. However, if $w_1 \in c_1$,$w_2 \in c_2$, $c_1, c_2 \in W$, and $c_1 \neq c_2$, then it necessarily exists some $i \in Q_B$ such that $\delta_B(i, w_1) \neq \delta_B(i, w_2)$.

Also, if $\delta_B(i, w_1) = j$ and $\delta_B(i, w_2) = j$ for some $i \in Q_B$, then we may still have $w_1 \not\equiv w_2$. We can only say that w_1 and w_2 are equivalent if Eq. (1) holds for all $i \in Q_B$. Therefore, the transitions from a state i in B can be precisely determined for each class $c \in W$.

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. Then, beside the normal indexing of variables in P, we need extra information for subscripts: one component to keep track of the states from Q_B, and another component to keep track of the class of W for variable instantiation. The indices in Q_B are used to keep track of the states in B, while those in W are used for synchronizing the instantiation of the variables v_k.

We also require some minimal information associated to the states of P, in order to keep track of whether a variable has been initialized or not. Moreover, if a newly introduced variable, derived from a variable v_k in P, is initialized, it must be synchronized with all subsequently used v_k-derived variables. Therefore, if a state in the newly constructed PA will have out-transitions labeled with v_k, then its name should bear enough information to distinguish among various circumstances of variables' instantiation.

Since in the new PA all transitions labeled with a variable corresponding to some v_k in P will belong to a corresponding new automaton, we need to construct these new automata/modules and establish their initial states. The information stored in the "Q_B component" of those initial states will be reflected in the indices of all variables labeling the transitions of those automata. The following sets are constructed to store this information.

Let $\mathcal{S} = (Q_B W \cup \{\varepsilon\})^{n+1}$. For $k \in \{0, \ldots, n\}$, let $\mathcal{S}_k$ be the set

$$S_k = \{S \in \mathcal{S} \mid S = (\underbrace{\varepsilon, \ldots, \varepsilon}_{n-k}, \alpha_k, \alpha_{k-1}, \ldots, \alpha_0)\}.$$

For $0 \leq h \leq n-1$, and each $S \in \mathcal{S}$ we will also use the projections $\pi_h : \mathcal{S} \longrightarrow (Q_B W \cup \{\varepsilon\})$, defined as $\pi_h(S) = \alpha_h$. This h-component of S, α_h stores the following information:

1. if $\alpha_h = \varepsilon$, then the variable v_h has not been instantiated yet;
2. if $\alpha_h \neq \varepsilon$, then $\alpha_h = b_h c_h$, ($b_h \in Q_B$ and $c_h \in W$), the variable v_h has been instantiated with a word in the class c_h.

The elements S of $\mathcal{S}_k$ will be used as indices for states in such a way, that will allow us avoid instantiations of more than one variables $v_{k,S}$ for a same k and different S'es.

From now on, we follow the convention that $S \in \mathcal{S}_k$ can be expressed as $S = (\alpha_k, \ldots, \alpha_h, \ldots, \alpha_1, \alpha_0)$, i.e., we omit the $n-k$ empty words preceding the index k.

Let $p = (r_0, \ldots, r_n)$ be the initial pattern expression with $n+1$ regular patterns and variables $v_0, \ldots, v_{n-1}$, and let $P = (C_0, C_1, \ldots, C_n)$ be the corresponding pattern automaton system, where

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

are the modules of P. We construct a pattern automaton system P' for the intersection $L(p) \cap L(B)$, consisting of the following finite automata:

1. for all k and $S \in \mathcal{S}_0 \backslash \{(\varepsilon)\}$, i.e., $S = (bc)$, for some $b \in Q_B$ and $c \in W$,
   $A_{0,S} = \big(Q_0 \times Q_B, \Sigma, (0_0, b), \delta_{0,S}, F_{0,j}\big)$, $j = f_c(b)$;
   for all $(p, l) \in Q_0 \times Q_B$ and for all $a \in \Sigma$:

   $$\delta_{0,S}((p, l), a) = \{(q, j') \mid q \in \delta_0(p, a),\ j' = \delta_B(l, a)\}$$

   and $F_{0,j} = F_0 \times \{j\}$;
   one can see that C_0 does not have transitions labeled with variables and in this case variables are not used, thus the construction is the usual automata Cartesian product [8], $C_0 \times B_{b, f_c(b)}$;
2. for all $k \in \{1, \ldots, n-1\}$ and $S \in \mathcal{S}_k$, with $\pi_k(S) = b_k c_k$; and denoting $b = b_k$, $d = c_k$, and $j = f_d(b)$, we have:

   $$A_{k,S} = \Bigg(Q_k \times Q_B \times S_{k-1},\ \Sigma \cup \{v_{k',S'} \mid k' < k,\ S' \in S_{k-1}\},$$
   $$(0_k, b, \underbrace{\varepsilon, \ldots, \varepsilon}_{k}),\ \delta_{k,S},\ F_{k,j}\Bigg),$$

   where $F_{k,j} = F_k \times \{j\}$, and

   (a) for all $(p, l, S') \in Q_k \times Q_B \times S_{k-1}$, $a \in \Sigma$:

   $$\delta_{k,S}\big((p, l, S'), a\big) = \{(q, j', S') \mid q \in \delta_k(p, a),\ j' = \delta_B(l, a)\};$$

   (b) for all $(p, b', S') \in Q_k \times Q_B \times S_{k-1}$ and $T' \in S_{k-1}$, such that there exists $k' < k$ verifying $\pi_{k'}(S') = \varepsilon$, $\pi_h(S') = \pi_h(T')$ for all $h \neq k'$, and $\pi_{k'}(T') = b'd$, we have:

   $$\delta_{k,S}\big((p, b', S'), v_{k',T'}\big) = \{(q, j', T') \mid q \in \delta_k(p, u_{k'}),\ j' = f_d(b')\};$$

   (c) for all $(p, b'', S') \in Q_k \times Q_B \times S_{k-1}$, $k' < k$, such that $\pi_{k'}(S') = b'd \neq \varepsilon$, we have:

   $$\delta_{k,S}\big((p, b'', S'), v_{k',S'}\big) = \{(q, j', S') \mid q \in \delta_k(p, v_{k'}),\ j' = f_d(b'')\}.$$

Note that, for any $k' < k$, a transition defined in (c) must be preceded by a transition in (b) that can be triggered at most once. This ensures a synchronization between subsequent transitions with $v_{k',S'}$, which mimics the synchronization present in C_k.

3. $A_{\mathcal{S}} = \Big(Q_n \times Q_B \times S_{n-1},\ \Sigma \cup \{v_{k',S'} \mid k' < n,\ S' \in S_{n-1}\},$

$$(0_n, 0, \underbrace{\varepsilon, \ldots, \varepsilon}_{n}),\ \delta_{n,F_B},\ F_{n,F_B} \Big),$$

where $F_{n,F_B} = F_n \times F_B$, and

(a) for all $(p, l, S') \in Q_k \times Q_B \times S_{n-1}$ and $a \in \Sigma$:

$$\delta_{n,F_B}\big((p, l, S'), a\big) = \{(q, j', S') \mid q \in \delta_n(p, a),\ j' = \delta_B(l, a)\};$$

(b) for all $(p, b', S') \in Q_n \times Q_B \times S_{n-1}$ and $T' \in S_{n-1}$, such that there exists $k' < n$ verifying $\pi_{k'}(S') = \varepsilon$, $\pi_h(S') = \pi_h(T')$ for all $h \neq k'$, and $\pi_{k'}(T') = b'd$, we have:

$$\delta_{n,F_B}\big((p, b', S'), v_{k',T'}\big) = \{(q, j', T') \mid q \in \delta_n(p, u_{k'}),\ j' = f_d(b')\};$$

(c) for all $(p, b'', S') \in Q_n \times Q_B \times S_{n-1}$, $k' < n$, such that $\pi_{k'}(S') = b'd \neq \varepsilon$, and $j' = f_d(b'')$:

$$\delta_{n,F_B}\big((p, b'', S'), v_{k',S'}\big) = \{(q, j', T') \mid q \in \delta_n(p, v_{k'})\}.$$

Then, the sought pattern automaton P' is obtained by ordering all the above automata with respect to their dependencies. Then, 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 indices S, we notice 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 $A_{k,S}$ are identical with A_k, for all S.
2. Denote $p_k = (r_0, \ldots, r_k)$ the pattern expression obtained from p considering only the first $k+1$ patterns with $0 \leq k < n$. Similarly, for $S \in S_k$, denote $P'_{k,S}$ the pattern automaton obtained from P' considering the automata $A_{k,S}$ and all its dependency modules $A_{k',S}$, with $k' < k$. Then one can check by induction that $L(p_k) \cap L(B_{b,j}) = L(P'_{k,S})$, where $\pi_k(S) = bc$ and $f_c(b) = j$.
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:

$$x_0 \in B_{0,b_1},\quad u_1 \in B_{b_1,j_1},\quad x_1 \in B_{j_1,b_2},\ \ldots,$$
$$x_{s-1} \in B_{j_{s-1},b_s},\quad u_s \in B_{b_s,j_s},\quad x_s \in B_{j_s,b_{s+1}},\quad \text{and}\quad b_{s+1} \in F_B.$$

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

For the reciprocal we can verify that all the above implications are in fact equivalences. 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. □

Unlike what we initially expected, the construction in Theorem 3 turned out to be substantially more complex than a simple automaton cross product. Indeed, the simple simulation of automaton B in parallel with the pattern automaton P does not suffice for recognizing the sought language intersection: one needs to keep track of variable instances and multiple non-deterministic computations performed by various modules of P, using only the means provided by a pattern automaton. Consequently, the resulting automaton has a size significantly bigger then the product of the sizes of the initial automata.

4 Limitations of pattern expressions

In this section we use Theorem 3 to prove that several 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 reversal operation. We first prove it on an easier case, that of alphabets with at least three letters, and we start with a preliminary result.

Lemma 3 *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 $m \geq 1$, $|x_0y| \leq N$, $|y| \geq 1$, 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 cannot 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 note 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 cannot cross the middle of w, and it cannot 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. □

Remark 2 Note that Theorem 2 is consequence of Lemma 3.

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 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 3, it follows that $L^{(5)} = L(p) \cap (abc)^*(cba)^*$ is also a PE language. This contradicts Lemma 3. □

In the following we prove an analogous result for the binary alphabet, this time using a more intricate combinatorial apparatus. The idea is to take the language $L^{(10)} = \{(aababb)^n(bbabaa)^n \mid n \geq 0\}$ and prove that is not a pattern expression language. We first prove two useful combinatorial lemmas.

Lemma 4 *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 cannot 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 5 *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 note 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 can be a subword of $ababb(bbabaa)^m$, for any $m \geq 0$. For example, $aaba$ cannot be a such subword, since in $ababb(bbabaa)^m$ any double occurrence of a is followed by two b's. The conclusion follows. □

Theorem 4 *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 intersection $L^{(4)} \cap \{(aababb)^n(bbabaa)^m \mid n, m \geq 0\}$ is exactly the language $L^{(10)} = \{(aababb)^n(bbabaa)^n \mid n \geq 0\}$. 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 $L^{(10)}$ is not a PE language. Assume by contradiction that $L^{(10)}$ is a PE language. Then, by applying the pumping lemma, there exists a constant N such that any word $w \in L^{(10)}$ of length greater than N has a decomposition $w = x_0yx_1y \ldots x_{t-1}yx_t$ for some $t \geq 1$, such that $|x_0y| \leq N$, $|y| \geq 1$, and $w_k = x_0y^kx_1y^k \ldots x_{t-1}y^kx_t \in L^{(10)}$, for all $k > 0$.

The pumping lemma applies to the word $w = (aababb)^n(bbabaa)^n$ with $6n > N + 6$, thus 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 4 and infer that $y^3 \not\preceq (aababb)^m$, for any $m > 0$. Note 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 a contradiction considering that y^3 is a subword of $(aababb)^m$. Thus, $w_3 \notin L^{(10)}$, 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 $y \preceq (aababb)^n$ because $|x_0 y| \leq N$; 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, using $y \preceq (aababb)^n$ and $|y| \geq 6$, by applying Lemma 5 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. Note that in this case, for each "pumped" y, 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 $L^{(10)}$ is not a PE language, and by Theorem 3 it follows that $L^{(4)}$ cannot 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:*

$$
\begin{aligned}
L^{(6)} &= \left\{ w \mid \ w = w^R \right\}, && (\textit{the language of palindromes}) \\
L^{(11)} &= \{ w \mid \ |w|_a = |w|_b \}, && (\textit{the language of balanced words}) \\
L^{(12)} &= \{ w \mid \ |w|_b = 2|w|_a \}, && (\textit{the language of semi-balanced words}) \\
L^{(13)} &= \{ w \mid \ |w|_a = |w|_b = |w|_c \}, \\
L^{(14)} &= \{ w \mid \ |w|_a + |w|_b = |w|_c \}, \\
L^{(15)} &= \left\{ ucv \mid \ |u|_a + |u|_b = |v|_a + |v|_b,\ u, v \in \{a, b, c\}^* \right\}.
\end{aligned}
$$

Proof We observe that: $L^{(6)} \cap (aababb)^*(bbabaa)^* = \{(aababb)^n(bbabaa)^n \mid n \geq 0\}$, which is not a PE language (Lemma 3); $L^{(11)} \cap a^*b^* = \{a^n b^n \mid n \geq 0\}$, which is not a PE language [5]; $L^{(12)} \cap a^*b^* = \left\{a^{2n}b^n \mid n \geq 0\right\}$, which is not a PE language [5, Example 7]; $L^{(13)} \cap a^*b^*c^* = \left\{a^n b^n c^n \middle| n \geq 0\right\}$, which is not a PE language [5]; $L^{(14)} \cap (a+b)^*c^* = \{\{a, b\}^n c^n \mid n \geq 0\}$, which is not a PE language [5, Example 8]; and $L^{(15)} \cap (a+b)^*c(a+b)^* = \{\{a, b\}^n c\{a, b\}^n \mid n \geq 0\}$, which is not a PE language [5]. □

Corollary 3 *The family of PE languages is not closed under shuffle with regular languages.*

Proof Assume by contradiction that PE languages are closed under shuffle with regular languages. Choose $p = (ucu, u = b^*)$ and define

$$L^{(16)} = (L(p) \sqcup L(a^*)) \cap L((baa)^*c(ba)^*).$$

By our assumption, and using the closure under the intersection with regular languages, it follows that $L^{(16)}$ is a PE language. However, using an argument similar to the one used in [3], we can prove that $L^{(16)}$ is not a PE language, since it does not satisfy the pumping Lemma for PE languages. We give the proof for completeness.

Let N be the constant obtained from the pumping lemma for the language $L^{(16)}$ and consider

$$w = (baa)^N c(ba)^N \in L^{(16)}.$$

By the pumping lemma we can write $w = x_0 y x_1 y \cdots y x_m$, where

$$w_j = x_0 y^j x_1 y^j \cdots y^j x_m \tag{2}$$

is in $L^{(16)}$ for any $j \geq 1$. Clearly, the subword y cannot contain the marker c, since it has a unique occurrence in words of $L^{(16)}$. We have four possibilities to consider.

1. If y contains a b and all occurrences of y in w are before the marker c, then w_2 contains more b's before the marker c than after it.
2. The case where all occurrences of y are after the marker is not possible because the prefix of w before the marker has length greater than N.
3. Consider the case where y contains a b and some subwords y occur before the marker c and some occur after it. Since w_2 has to be in $L((baa)^*c(ba)^*)$ and some subword y occurs before c, it follows that $2 \cdot |y|_b = |y|_a$. Similarly, since another subword occurrence y lies after the marker c, it follows that $|y|_b = |y|_a$. This is impossible, since $y \neq \varepsilon$.
4. Finally, if y consists only of a's (that is, $y = a$ or $y = aa$), then $w_3 \notin L((baa)^*c(ba)^*)$.

If $L(p) \sqcup L(a^*)$ was a PE language, so would be $L^{(16)}$ and we just proved that it is not. □

Since the class of PE languages is not closed under inverse homomorphism, the result of Corollary 3 can be alternatively proven by invoking the following known property (we provide a proof for completeness):

Lemma 6 *Every family $\mathcal{F}$ of languages, closed under homomorphism, intersection with regular languages and shuffle with regular languages is closed under inverse homomorphism.*

Proof Assume that $\mathcal{F}$ is a family of languages with the closure properties listed above. Let $h : \Sigma^* \longrightarrow \Delta^*$ be an arbitrary homomorphism and $L \subseteq \Delta^*$ be a language in $\mathcal{F}$. Also let $\Sigma' = \{a' \mid a \in \Sigma\}$ be a copy of Σ such that $\Sigma' \cap \Delta = \emptyset$. The following equality is immediate:

$$h^{-1}(L) = g((L \sqcup \Sigma') \cap \{h(a)a' \mid a \in \Delta\}^*),$$

where g is a homomorphism that removes all letters in Σ and restores the quoted letters to the original ones. □

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

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 popular 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 expressions unforeseen before. Recently, we have proven similar results for regex languages, however, in a different framework (see [4]). Incidentally, we have also raised a warning on the inefficiency of the membership testing for pattern expressions. Thus, it is in our belief that our results reach out beyond theory, to programmers and other users of PE and practical regular expressions.

Acknowledgments We are thankful to Dr. Max Burke for suggesting the language $L^{(5)}$, and to the anonymous referee who brought to our attention Lemma 6 and who made other suggestions that helped improve the quality of this presentation.

References

1. Aho, A.V.: Algorithms for Finding Patterns in Strings. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Algorithms and Complexity, vol. A, pp. 255–300. Elsevier, MIT Press, Amsterdam, New York (1990)
2. Angluin, D.: Finding patterns common to a set of strings. J. Comput. Syst. Sci. **21**, 46–62 (1980)
3. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. IJFCS **14**(6), 1007–1018 (2003)
4. Câmpeanu, C., Santean, N.: Addressing an Open Problem on Regex. Technical Report CS-2007-10, School of Computer Science, The University of Waterloo. http://www.cs.uwaterloo.ca/research/tr/2007/
5. Câmpeanu, C., Yu, S.: Pattern expressions and pattern automata. IPL **92**, 267–274 (2004)
6. Dumitrescu, S., Păun, G., Salomaa, A.: Pattern Languages versus Parallel Communicating Grammar Systems. TUCS Report, vol. 42, September 1996
7. Friedl, J.E.F.: Mastering Regular Expressions. O'Reilly & Associates, Inc., Cambridge (1997)
8. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Reading (2006)
9. Kari, L., Mateescu, A., Păun, G., Salomaa, A.: Multi-pattern languages. Theor. Comp. Sci. **141**, 253–268 (1995)
10. Kobayashi, S., Mitrana, V., Păun, G., Rozenberg, G.: Formal Properties of PA-matching. Theor. Comp. Sci. **262**(1-2), 117–131 (2001)
11. Lesk, M.E.: Lex—a Lexical Analyzer Generator. Computer Science Technical Report, vol. 39, AT&T Bell Laboratories, Murray Hill (1975)
12. Martín-Vide, C., Mitrana, V.: Some Undecidable Problems for Parallel Communicating Finite Automata Systems. Inf. Process. Lett. **77**, 239–245 (2001)
13. Martín-Vide, C., Mitrana, V.: Remarks on Arbitrary Multiple Pattern Interpretations. Inf. Process. Lett. (in press), available online 24 October (2006)
14. Mitrana, V., Păun, G., Rozenberg, G., Salomaa, A.: Pattern systems. Theor Comp. Sci. **154**(2), 183–201 (1996)
15. Salomaa, A.: Theory of Automata. Pergamon Press, Oxford (1969)
16. Salomaa, A.: Formal Languages. Academic Press, New York (1973)
17. Yu, S.: Regular Languages. In: Salomaa, A., Rozenberg, G. (eds.) Handbook of Formal Languages, pp. 41–110. Springer, Heidelberg (1997)