International Journal of Foundations of Computer Science

A Formal Study of Practical Regular Expressions

Cezar Câmpeanu
Department of Mathematics and Computer Science, UPEI
Charlottetown, PE C1A 4P3
ccampeanu@upei.ca

Kai Salomaa
Computing and Information Science Department
Queen's University
Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Sheng Yu
Department of Computer Science
University of Western Ontario
London, Ontario N6A 5B7, Canada
syu@csd.uwo.ca

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Regular expressions are used in many practical applications. Practical regular expressions are commonly called "regex". It is known that regex are different from regular expressions. In this paper, we give regex a formal treatment. We make a distinction between regex and extended regex; while regex represent regular languages, extended regex represent a family of languages larger than regular languages. We prove a pumping lemma for the languages expressed by extended regex. We show that the languages represented by extended regex are incomparable with context-free languages and a proper subset of context-sensitive languages. Other properties of the languages represented by extended regex are also studied.

Keywords: Regular expressions, regex, extended regex, formal languages, programming languages.

1. Introduction

Regular expressions are used in many practical applications, e.g., Perl, Awk, Python, egrep, vi, and emacs. It is known that the practical regular expressions are different from the theoretical ones. The practical regular expressions [5] are often called regex. Regex were developed under the influence of theoretical regular expressions. For example, the regex in Lex [9] are similar to theoretical regular

expressions. However, regex are quite different in many other environments. Many regex in use now can express a larger family of languages than the regular languages. For example, Perl regex [3] can express $L_1 = \{a^n b a^n \mid n \geq 0\}$ and $L_2 = \{ww \mid w \in \{a, b\}^*\}$. However, Perl regex cannot express the language $L_3 = \{a^n b^n \mid n \geq 0\}$. It is relatively easy to show that a language can be expressed by a regex. For example, L_1 can be expressed by a Perl regex $(a*)b\backslash 1$ and L_2 by $((a|b)*)\backslash 1$. However, it is usually difficult to show that a language cannot be expressed by certain devices. For example, how do we prove that L_3 cannot be expressed by a Perl regex? It is clear that there is a need for some formal treatment of regex. This is the main purpose of the paper.

Regex are defined differently in different environments. However, in some sense, there is a common subset of the definitions. In this paper, we first give a formal definition of regex and extended regex according to the common subset of the definitions given in different environments. Then we prove a pumping lemma for extended regex languages. We also show that the family of extended regex languages is a proper subset of the family of context-sensitive languages, and it is incomparable with the family of context-free languages. Several other properties of extended regex languages are also studied in this paper.

Recent theoretical results on "standard" regular expressions can be found in [4] and in the references listed there. Some references for results on translating regular expressions into finite automata and on implementing such automata are [1, 2, 7, 8].

2. Basic definition of regex and extended regex

Let Σ be an ordered set of all printable letters except that each of the following letters is written with an escape character $\backslash$ in front of it: $(,), \{, \}, [,], \$, |, \backslash, ., ?, *$, and $+$. In addition, Σ also includes $\backslash n$ and $\backslash t$ as the new line and tab character, respectively. In the following, we give the definition of regex as well as the language it defines. For an expression e, we use $L(e)$ to denote the set of all words that match e, i.e., the language e defines.

Basic form of regex:

(1) For each $a \in \Sigma$, a is a regex and $L(a) = \{a\}$. Note that for each $x \in \{(,), \{, \}, [,], \$, |, \backslash, ., ?, *, +\}$, $\backslash x \in \Sigma$ and is a regex and $L(\backslash x) = \{x\}$. In addition, both $\backslash n$ and $\backslash t$ are in Σ and are regex, and $L(\backslash n)$ and $L(\backslash t)$ denote the languages consisting of the new line and the tab, respectively.

(2) For regex e_1 and e_2,

$(e_1)(e_2)$ (concatenation),
$(e_1)|(e_2)$ (alternation), and
$(e_1)*$ (Kleene star)

are all regex, where $L((e_1)(e_2)) = L(e_1)L(e_2)$, $L((e_1)|(e_2)) = L(e_1) \cup L(e_2)$, and $L((e_1)*) = (L(e_1))^*$. Parentheses can be omitted. When they are omitted, alternation, concatenation, and Kleene star have the increasing priority.

(3) A regex is formed by using (1) and (2) a finite number of times.

Shorthand form:

(1) For each regex e, $(e)+$ is a regex and $(e)+ \equiv e(e)*$.

(2) The character '.' means any character except '$\backslash n$'.

Character classes:

(1) For $a_{i_1}, a_{i_2}, \ldots, a_{i_t} \in \Sigma$, $t \geq 1$, $[a_{i_1}a_{i_2}\ldots a_{i_t}]$ is a regex and $[a_{i_1}a_{i_2}\ldots a_{i_t}] \equiv a_{i_1}|a_{i_2}|\ldots|a_{i_t}$.

(2) For $a_i, a_j \in \Sigma$ such that $a_i \leq a_j$, $[a_i\text{-}a_j]$ is a regex and $[a_i\text{-}a_j] \equiv a_i|a_{i+1}|\ldots|a_j$.

(3) For $a_{i_1}, a_{i_2}, \ldots, a_{i_t} \in \Sigma$, $t \geq 1$, $[\hat{}a_{i_1}a_{i_2}\ldots a_{i_t}]$ is a regex and $[\hat{}a_{i_1}a_{i_2}\ldots a_{i_t}] \equiv b_{i_1}|b_{i_2}|\ldots|b_{i_s}$, where $\{b_{i_1}, b_{i_2}, \ldots, b_{i_s}\} = \Sigma - \{a_{i_1}, a_{i_2}, \ldots, a_{i_t}\}$.

(4) For $a_i, a_j \in \Sigma$ such that $a_i \leq a_j$, $[\hat{}a_i\text{-}a_j]$ is a regex and $[\hat{}a_i\text{-}a_j] \equiv b_{i_1}|b_{i_2}|\ldots|b_{i_s}$, where $\{b_{i_1}, b_{i_2}, \ldots, b_{i_s}\} = \Sigma - \{a_i, a_{i+1}, \ldots, a_j\}$.

(5) Mixture of (1) and (2), or (3) and (4), respectively.

Anchoring:

(1) Start-of-line anchor $\hat{}$.

(2) End-of-line anchor \$.

We call an expression that satisfies the above definitions a *regex*. Note that the empty string is not a regex.

Example 1 *The expression "*$\hat{}.*\$$*" is a regex, which matches any line including the empty line.*

Example 2 *The expression "*[A-Z][A-Za-z0-9]$*$*" matches any word starting with a capital letter and followed by any number of letters and digits.*

It is clear that regex can express only regular languages. However, the following construct, which appears in Perl, Emacs, etc., is not a regular construct.

Back reference:

> '$\backslash m$', where m is a number, matches the content of the mth pair of parentheses before it.

Note that the pairs of parentheses in a regex are ordered according to the occurrence sequence of their left parentheses. Note also that if the mth pair of parentheses are in a Kleene star and '$\backslash m$' is not in the same Kleene star, then '$\backslash m$' matches the content of the mth pair of parentheses in the last iteration of the Kleene star. A more formal definition is given later. We first look at several introductory examples.

Example 3 *The expression "*$(a*)b\backslash 1$*" defines the language*

$$\{a^n b a^n \mid n \geq 0\}.$$

Example 4 *The expression "(The)(file ([0-9][0-9])).*\3" matches any string that contains "The file " followed by a two-digit number, then any string, and then the same number again.*

Example 5 *For the expression* $e = (a*b)*\backslash 1$, $aabaaabaaab \in L(e)$ *and* $aabaaabaab \notin L(e)$.

There appears to be no convenient way to recursively define the set of extended regex α satisfying the condition that any back reference in α occurs after the corresponding parenthesis pair. Thus below we first define an auxiliary notion of semi-regex and the extended regex are then defined as a restriction of semi-regex.

Definition 1 *A* semi-regex *is a regex over the infinite alphabet*

$$\Sigma \cup \{\backslash m \mid m \in N\}$$

where N is the set of natural numbers. Let α be a semi-regex. The matching parenthesis pairs of α are numbered from left to right according to the occurrence of the left parenthesis (the opening parenthesis) of each pair.

A semi-regex α is an extended regex *if the following condition holds. Any occurrence of a back reference symbol $\backslash m$ ($m \in N$) in α is preceded by the closing parenthesis of the mth parenthesis pair of α.*

Below we define the matches of an extended regex and the language defined by an extended regex. Intuitively, a match of an extended regex α is just a word denoted by the regex when each back reference symbol $\backslash m$ is replaced by the contents of the subexpression β_m corresponding to the mth pair of parentheses in α. Due to the star operation, a given subexpression occurrence β_m may naturally "contribute" many subwords to a match. Following the convention used e.g. in Perl, $\backslash m$ will be replaced by the contents of the last (rightmost) occurrence of β_m appearing before this occurrence of $\backslash m$.

The condition that each back reference symbol occurs after the corresponding parenthesis pair guarantees that a match as defined below cannot contain circular dependencies.

We denote the set of occurrences of subexpressions of an extended regex α as $\mathrm{SUB}(\alpha)$. Distinct occurrences of an identical subexpression are considered to be different elements of $\mathrm{SUB}(\alpha)$. A match of an extended regex α is defined as a tree T_α following the structure of α. Naturally, due to the star operation, T_α may make multiple copies of parts of the structure of α.

Definition 2 *A* match *of an extended regex α is a finite (directed, ordered) tree T_α. The nodes of T_α are labeled by elements of $\Sigma^* \times \mathrm{SUB}(\alpha)$ and T_α is constructed according to the following rules.*

(i) *The root of T_α is labeled by an element (w, α), $w \in \Sigma^*$.*

(ii) *Assume that a node u of T_α is labeled by (w, β) where $\beta = (\beta_1)(\beta_2) \in \mathrm{SUB}(\alpha)$. Then u has two successors that are labeled, respectively, by (w_i, β_i), $i = 1, 2$, where $w_1, w_2 \in \Sigma^*$ have to satisfy $w = w_1 w_2$.*

(iii) *Assume that a node u of T_α is labeled by (w, β) where $\beta = (\beta_1)|(\beta_2) \in \mathrm{SUB}(\alpha)$. Then u has one successor that is labeled by one of the elements (w, β_i), $i \in \{1, 2\}$.*

(iv) *Assume that a node u of T_α is labeled by (w, β) where $\beta = (\beta_1)* \in \mathrm{SUB}(\alpha)$. Then there are two cases: $w \neq \lambda$ or $w = \lambda$. If $w \neq \lambda$, u has $k \geq 1$ successors labeled by*

$$(w_1, \beta_1), \ldots, (w_k, \beta_1)$$

where $w_1 \cdots w_k = w$, $w_i \neq \lambda$, $i = 1, \ldots, k$. If $w = \lambda$, u has a single successor u_1 labeled by (λ, β) and u_1 is a leaf of T_α.

(v) *If (w, a), $a \in \Sigma$, occurs as a label of a node u then necessarily u is a leaf and $w = a$.*

(vi) *If $(w, \backslash m)$ occurs as a label of a node u then u is a leaf and $w \in \Sigma^*$ is determined as follows. Let β_m be the subexpression of α enclosed by the mth pair of parentheses and let u_{β_m} be the previous node of T_α in the standard left-to-right ordering that is labeled by an element where the second component is β_m that precedes u in the left-to-right ordering of the nodes. (Note that all nodes where the label contains β_m are necessarily independent and hence they are linearly ordered from left to right.) Then we choose w to be the first component of the label of u_{β_m}. Finally, if β_m does not occur in the label of any node of T_α then we set $w = \lambda$.*

The language *denoted by an extended regex α is defined as*

$$L(\alpha) = \{w \in \Sigma^* \mid (w, \alpha) \text{ labels the root of some match } T_\alpha\}.$$

Above the somewhat complicated condition (vi) means just that a back reference symbol $\backslash m$ is substituted by the previous occurrence of the contents of the subexpression β_m corresponding to the mth parenthesis pair. It is possible that β_m does not appear in the label of any node of T_α if β_m occurs inside a Kleene star and in (iv) we choose zero iterations of the star operation. In this case the contents of $\backslash m$ will be λ.

If the nondeterministic top-down construction of T_α produces a leaf symbol violating one of the conditions (v) or (vi), then the resulting tree is not a well-formed match of α. The fact that a back reference symbol $\backslash m$ has to occur after the expression corresponding to the mth parenthesis pair guarantees that in (vi) the label of u_{β_m} can be determined before the label of u is determined.

Example 6 *Let $\alpha = (((a|b)*)c\backslash 2)*$. Then $L(\alpha) = L_1^*$ where*

$$L_1 = \{wcw \mid w \in \{a, b\}^*\}.$$

In the above examples, and in what follows, for simplicity we often omit several parenthesis pairs and only the remaining parentheses are counted when determining which pair corresponds to a given back reference.

When we compare the extended regex languages with other language families it is convenient to have a regex that denotes the empty set. In the following, we assume the symbol φ is a regex and it denotes the empty set. As usual, φ^* denotes the set $\{\lambda\}$ containing the empty word only. We can also consider that the alphabet Σ for a regex or extended regex α consists of only the characters that occurs in α if Σ is not explicitly defined. We did not include these technicalities in the definition given at the start of this section in order to keep it close to the practical definitions.

3. A pumping lemma for extended regex languages

There are several pumping lemmas for regular languages [6, 11], which are useful tools for showing that certain languages are not regular languages. In the following, we prove a pumping lemma for extended regex languages and give several examples to show how to use the lemma to prove that certain languages are not extended regex languages.

For a string (word) $x \in \Sigma^*$, denote by $|x|$ the length of x in the following.

Lemma 1 *Let α be an extended regex. Then there is a constant N such that if $w \in L(\alpha)$ 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. $x_0y^jx_1y^j\cdots x_m \in L$ *for all* $j > 0$.

Proof. Let $N = |\alpha|2^t$ where t is the number of back-references in α. Let $w \in L(\alpha)$ and $|w| > N$. By the definition of N it is clear that some part of w matches a Kleene star or plus in α that has more than one iteration, because each back reference that does not appear inside a Kleene star can at most double the length of the word it matches.

Let $w = x_0yz$ and y is the first nonempty substring of w that matches a Kleene star or plus. In order to satisfy $|x_0y| \leq N$, we let y match the first iteration of the star or plus. In other words, we have $e*$ or $e+$ in the extended regex; the y in x_0y matches e and is the first iteration of the star or plus. Then we have $|x_0y| \leq N$ and $|y| \geq 1$. There are the following two cases: Case 1, $e*$ (or $e+$) is not back-referenced. Then it is clear that the lemma holds for $m = 1$. Case 2, $e*$ (or $e+$) is back-referenced. Let $z = x_1yx_2yx_3\cdots x_m$ where y's are all the back-references of y. Then we have $w = x_0yx_1yx_2yx_3\cdots x_m$, and it is clear that $x_0y^jx_1y^jx_2y^j\cdots x_m \in L(\alpha)$ for all $j \geq 1$.

Note that in the case that e in $(e)*$ (or $(e)+$), rather than $e*$ (or $e+$), is back-referenced and y matches e, the lemma clearly holds for $m = 1$. □

Example 7 *Consider some special cases of regex for the pumping lemma. Let $e_1 = (ab|ba)(\backslash 1)*$. Then the constant N for the pumping lemma is $|e_1| \times 2 = 24$. Since the Kleene star is not referenced, any word w that matches e_1 and $|w| > N$ can be decomposed into xyz such that $|xy| \leq N$, $|y| \geq 1$, and $xy^jz \in L(e_1)$ for all $j > 0$. For example, $w = (ba)^{13}$. Then $x = ba$, $y = ba$, and $z = (ba)^{11}$.*

Let $e_2 = bab((a|b)c\backslash 2)*$. Then $N = 28$. Again the Kleene star is not referenced. So, any word $w \in L(e_2)$ and $|w| > N$ can be decomposed into xyz such that $|xy| \leq N$, and $|y| \geq 1$, and $xy^jz \in L(e_2)$ for any $j > 0$. For example, let $w = babacabcbbcbacabcbacaacaacaaca$. Then $x = bab$, $y = aca$, and $z = bcbbcbacabcbacaacaacaaca$.

Let $e_3 = (a*)b\backslash 1bb\backslash 1bbb$. Then $N = 14 \times 2^2 = 56$. Any word $w \in L(e_3)$ and $|w| > N$ can be decomposed into $x_0yx_1yx_2yx_3$ such that $|x_0y| \leq N$, $|y| \geq 1$, and $x_0y^jx_1y^jx_2y^jx_3 \in L(e_3)$ for all $j > 0$.

Next we show how the pumping lemma is used.

Example 8 *The language $L = \{a^nb^n \mid n > 0\}$ cannot be expressed by an extended regex.*

Proof. Assume that L is expressed by an extended regex. Let N be the constant of Lemma 1. Consider the word a^Nb^N. By Lemma 1, a^Nb^N has a decomposition $x_0yx_1yx_2 \cdots x_m$, $m \geq 1$, such that (1) $|x_0y| \leq N$, (2) $|y| \geq 1$, and (3) $x_0y^jx_1y^j \cdots x_m \in L$ for all $j > 0$. According to (1) and (2), $y = a^i$ for some $i > 0$. But then the word $x_0y^2x_1y^2 \cdots x_m$ is clearly not in L. It is a contradiction. Therefore, L does not satisfy Lemma 1 and, thus, L cannot be expressed by any extended regex. □

Similarly, we can prove that the language $\{a^nb^nc^n \mid n \geq 1\}$ is not an extended regex language. As an application of the pumping lemma we get also the following.

Example 9 *The language $L = \{\{a, b\}^nc\{a, b\}^n \mid n \geq 0\}$ is not an extended regex language.*

Proof. Suppose that L is an extended regex language. Then, by the pumping lemma, there exists an integer $N > 0$ such that $a^Ncb^N = x_0yx_1y \cdots x_m$ for some $m \geq 1$ and $|y| > 0$, and $x_0y^ix_1y^i \cdots x_m \in L$ for all $i > 0$. It is clear that y cannot contain c. Then we have either $y = a^k$ or $y = b^k$ for some $k > 0$. In either case, we have $x_0y^2x_1 \cdots x_m \notin L$. This is a contradiction. So, L is not an extended regex language. □

Lemma 2 *The family of extended regex languages is not closed under complementation.*

Proof. The language

$$L_1 = \{a^m \mid m > 1 \text{ is not prime } \}$$

is expressed by the extended regex $(aaa*)\backslash 1(\backslash 1)*$. Assume that the complement of L_1, L_1^c, is an extended regex language and apply Lemma 1 to L_1^c. This implies that there exist $n_1 \geq 0$, $n_2 \geq 1$ such that for all $j > 0$,

$$a^{n_1+j\cdot n_2} \in L_1^c.$$

This is a contradiction since it is not possible that $n_1 + j \cdot n_2$ is prime for all $j > 0$. □

We can note also that the language L_1 in the proof of Lemma 2 is an extended regex language over a one-letter alphabet that is not context-free. In particular, this

means that there are extended regex languages that do not belong to the Boolean closure of context-free languages.

4. Other properties of regex and extended regex languages

Here we prove that every extended regex language is a context-sensitive language. We also establish the relationship between extended regex languages and context-free languages.

Theorem 1 *Extended regex languages are context-sensitive languages*

Proof. It suffices to show that each extended regex language is accepted by a linear-bounded automaton (LBA), i.e., a nondeterministic Turing machine in linear space. For a given extended regex, if there is no back reference, we can simply construct a finite automaton that will accept all words that match the regex.

For handling back references, we need store each string that matches a subexpression that is surrounded by a pair of parentheses for later use. If there are m pairs of parentheses in the extended regex, we need m buffers. We store the part of the input strings that match the content of the ith pair of parentheses in the ith buffer.

For a given extended regex, we first label all the parentheses with their appearance number. For example, the extended regex "$(a*(ba*)\backslash 2)b$" would become "$({}_1a*({}_2ba*)_2\backslash 2)_1b$". Note that for a fixed extended regex the number of parenthesis pairs is a constant that does not depend on the input word, and hence the numbering can be done easily in linear space. Then we construct a nondeterministic finite automaton (NFA) which treats each labeled parenthesis and each back reference as an input symbol.

Then we build a Turing machine which works according to the NFA as follows. It reads the input symbols and follows the transitions of the NFA. For a $({}_i$ transition, it does not read input symbol but starts to store the next matching input symbols into the ith buffer. For a $)_i$ transition, it also does not read any input symbol but stops storing the matching input symbols into the ith buffer. Whenever there is a back reference in the extended regex, say "$\backslash i$", it just compares the string with the content of the ith buffer. It accepts the input string if there is a way to reach a final state when just finishes reading the input. Note that if the ith pair of parentheses are in a Kleene star, it may store input symbols in the ith buffer several times. Then "$\backslash i$" always matches the latest word stored in the ith buffer.

Each buffer needs at most the space of the input word, and there are a constant number of buffers. Thus, an extended regex language is accepted by an LBA and, thus, is a context-sensitive language.

□

Theorem 2 *The family of extended regex languages is incomparable with the family of context-free languages.*

Proof. The language $L = \{a^nba^nba^n \mid n \geq 1\}$ is clearly an extended regex language. It can be expressed as "$(a+)b\backslash 1b\backslash 1$". However, L is not a context-free language. We know that $\{a^nb^n \mid n \geq 0\}$ is a context-free language, but it is not an extended regex language as we have proved in Section 3. □

Theorem 3 *The family of extended regex languages is a proper subset of the family of context-sensitive languages.*

We study several more closure properties of extended regex languages in the following.

Theorem 4 *The family of extended regex languages is closed under homomorphism.*

Proof. Let L be an extended regex language and $L = L(\alpha)$ for some extended regex α. Let $h : \Sigma^* \to \Sigma^*$ be a homomorphism. Then clearly $h(L)$ can be denoted by an extended regex β that is obtained from α as follows: (1) Replace each letter $a \in \Sigma$ by $h(a)$. (2) Keep all the operators, parentheses and back references unchanged. (3) Add parentheses if necessary. For example $h(a) = ab, h(b) = b$ and $\alpha = ba^*bb$. Then $\beta = b(ab)^*bb$. However, the numbers in the back references have to be changed accordingly if new parentheses are added. □

Theorem 5 *The family of extended regex languages is not closed under inverse homomorphisms.*

Proof. The language $L_1 = \{a^n ca^n \mid n \geq 0\}$ is an extended regex language since $L_1 = L(\alpha)$, where $\alpha = (a^*)c\backslash 1$. Define a homomorphism $h : \{a, b, c\}^* \to \{a, b\}^*$, $h(a) = h(b) = a$ and $h(c) = c$. We have shown in Example 9 that $L = \{\{a, b\}^n c\{a, b\}^n \mid n \geq 0\}$ is not an extended regex language. Since $L = h^{-1}(L_1)$, we have proved the theorem. □

Similarly, we can prove the following.

Theorem 6 *The family of extended regex languages is not closed under finite substitutions.*

Proof. We can use the same languages L_1 and L as in the proof for the previous theorem. Let a finite substitution π be defined as $\pi(a) = \{a, b\}$ and $\pi(c) = \{c\}$. Clearly, $\pi(L_1) = L$. The we can conclude the proof since we know that L_1 is an extended regex language while L is not. □

As another negative closure result we show that the extended regex languages are not closed under shuffle, and not even under shuffle with a regular language. The shuffle operation is used to model parallel decomposition of words and languages. We briefly recall here the definition of shuffle, for more details the reader is referred to [10].

The *shuffle* of words $w_1, w_2 \in \Sigma^*$ is the set

$$\begin{aligned} w_1 \sqcup\!\sqcup w_2 \quad = \quad & \{u_1 v_1 u_2 v_2 \cdots u_m v_m \mid \\ & u_i, v_i \in \Sigma^*,\ i = 1, \ldots, m,\ w_1 = u_1 \cdots u_m,\ w_2 = v_1 \cdots v_m\}. \end{aligned}$$

The shuffle of two languages L_1 and L_2 is then defined as

$$L_1 \sqcup\!\sqcup L_2 = \bigcup_{w_1 \in L_1, w_2 \in L_2} w_1 \sqcup\!\sqcup w_2.$$

It is well known that the family of regular languages is closed under shuffle.

For our non-closure result we need the following technical lemma.

Lemma 3 *Let α be an extended regex over the alphabet $\Sigma = \{a, b, c\}$ and denote*

$$\beta = (baa) * c(ba) * .$$

Then $L(\alpha) \cap L(\beta)$ is an extended regex language.

Proof. First we construct an extended regex α' that defines the subset of $L(\alpha)$ consisting of all words that have exactly one occurrence of c. This can be done by restricting any *-subexpression that contains c to "repeat" only one or zero times and then taking all combinations of repeats of different subexpressions that produce exactly one occurrence of c. In particular, no back-reference symbol is allowed to refer to a subexpression producing c. For example, if $\backslash n$ refers to (δ^*) where instances of δ contain c, this occurrence of δ^* is replaced by λ. If no back-reference symbol refers to (δ^*), it is replaced by λ or δ depending on the subexpressions it is catenated with. If only some instances of δ contain c (δ is a union of subexpressions), the construction has to be divided into separate cases.

A problem with α' is that it may still contain back-references that refer to a subword occurrence on the "other side" of the marker c. However, in any match of α' that belongs to $L(\beta)$ such back-references must necessarily be substituted by a word of length at most 2 (since any word in $L(\beta)$ does not contain subwords of length greater than 2 that would occur on both sides of the marker c). Thus we can eliminate back-references that refer to the other side of the marker by replacing them on both sides with the different possible constant strings. The resulting extended regex α'' is not equivalent to α', but we have $L(\alpha'') \cap L(\beta) = L(\alpha') \cap L(\beta)$.

Now the extended regex α'' can be written as a finite union of expressions $\gamma_{1,i} c \gamma_{2,i}$, $i = 1, \ldots, m$, where $\gamma_{1,i}$ and $\gamma_{2,i}$ do not contain back-references to each other. (Note that in α' the symbol c does not occur inside a star.) It is sufficient to construct extended regex $\delta_{1,i}$ (respectively, $\delta_{2,i}$) such that $L(\delta_{1,i}) = L(\gamma_{1,i}) \cap L((baa)*)$ (respectively, $L(\delta_{2,i}) = L(\gamma_{2,i}) \cap L((ba)*)$), $i = 1, \ldots, m$.

We construct $\delta_{1,i}$ by recursively replacing each subexpression $(\mu_1 | \cdots | \mu_k)*$ of $\gamma_{1,i}$ by a finite union of all expressions of the following type that can be denoted by $(\mu_1 | \cdots | \mu_k)*$:

$$u(baa) * v, \quad u(baa)^h v \tag{1}$$

where u is a suffix of baa, v is a prefix of baa, and $0 \leq h$ is bounded by the maximum length of the expressions $\mu_1, \ldots, \mu_k$.

When a back-reference symbol $\backslash n$ refers to the original expression $(\mu_1 | \cdots | \mu_k)*$, in the corresponding finite union copies of $\backslash n$ are made to refer to expressions of the form (1), and the pair (u, v) is called the type of the back-reference symbol. When using concatenation, the combination of subexpressions (or back-reference symbols) of types (u_1, v_1), (u_2, v_2) is allowed only when $v_1 u_2 = baa$. In this way, the construction of $\delta_{1,i}$ can enforce that the word matched by $\gamma_{1,i}$ is in $L((baa)*)$.

The construction of $\delta_{2,i}$ is exactly similar. □

Theorem 7 *The family of extended regex languages is not closed under shuffle with regular languages.*

Proof. For the sake of contradiction assume that the extended regex languages are closed under shuffle with regular languages. Choose $\gamma = (b*)c\backslash 1$ and define

$$L_1 = (L(\gamma) \sqcup\!\sqcup L(a*)) \cap L((baa) * c(ba)*).$$

By our assumption and Lemma 3 it follows that L_1 is an extended regex language.

Let N be the constant obtained from the pumping lemma for the language L_1 and consider

$$w = (baa)^N c(ba)^N \in L_1.$$

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_0 for any $j \geq 1$. Clearly the subword y cannot contain the marker c since it has a unique occurrence in words of L_1. 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 $y \neq \lambda$.

4. Finally, if y consists only of a's (that is, $y = a$ or $y = aa$), then $w_3 \notin L((baa) * c(ba)*)$.

□

5. Conclusions

We have given "practical regular expressions", i.e., regex and extended regex, a rather formal treatment. We have introduced a pumping lemma for extended regex languages. We have shown that the family of extended regex languages is a proper subset of the family of context-sensitive languages and is incomparable with the family of context-free languages. It is clear that extended regex languages are closed under union and we have proved that they are not closed under complementation; however, it remains open whether or not the extended regex languages are closed under intersection. In fact, it is not even clear how the proof of Lemma 3 could be extended to handle arbitrary regular expressions, and thus it is also open whether or not the extended regex languages are closed under intersection with a regular language.

References

1. J.-M. Champarnaud, Implicit Structures to Implement NFA's from Regular Expressions. CIAA 2000 Revised Papers, (S. Yu and A. Paun, Eds.) Lect. Notes Comput. Sci. 2088, Springer-Verlag, 2001, pp. 80–93.
2. J.-M. Champarnaud and D. Ziadi, New Finite Automaton Constructions Based on Canonical Derivatives. CIAA 2000 Revised Papers, (S. Yu and A. Paun, Eds.) Lect. Notes Comput. Sci. 2088, Springer-Verlag, 2001, pp. 94–104.
3. N. Chapman, *Perl — The Programmer's Companion*, Wiley, Chichester, 1997.
4. K. Ellul, J. Shallit and M. Wang, Regular Expressions: New Results and Open Problems. *Proceedings of DCFS'02,* (London, Ontario, August 2002), pp. 17–34.
5. Jeffrey E.F. Friedl *Mastering Regular Expressions*, O'Reilly & Associates, Inc., Cambridge, 1997.
6. J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.
7. J. Hromkovic, S. Seibert and T. Wilke, Translating Regular Expressions into Small ε-Free Nondeterministic Finite Automata. *Journal of Computer and System Sciences* 62 (2001) 565–588.
8. L. Ilie and S. Yu, Algorithms for computing small NFAs. *Mathematical Foundations of Computer Science*, LNCS 2420 (2002) 328-340.
9. M.E. Lesk, "Lex - a lexical analyzer generator", *Computer Science Technical Report* (1975) 39, AT&T Bell Laboratories, Murray Hill, N.J.
10. A. Mateescu, G. Rozenberg and A. Salomaa, Shuffle on trajectories: Syntactic constraints, *Theoretical Computer Science* 197 (1998) 1–56.
11. S. Yu, "Regular Languages", in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa eds. pps. 41-110, Springer, 1998.