On Expressive Power of Regular Expressions with Subroutine Calls and Lookaround Assertions

Ondřej Guth

Czech Technical University in Prague
Thákurova 9
16000 Praha
Czech Republic
ondrej.guth@fit.cvut.cz

Abstract. Many regular expression engines employ syntactical extensions to provide simple, expressive support for real-world needs. These features are subroutine calls, zero-width lookaround assertions, DEFINE rules, and named parenthesised expressions. A subroutine call executes a specified subpattern where the call is placed, possibly recursively. Lookaround assertions are either lookahead or lookbehind: a lookahead is a conditional within a subpattern: when it fails, the match at the current position of the whole subpattern fails, while a lookahead itself does not consume any input; a lookbehind works as a lookahead except it checks the input prior to the current position. A DEFINE rule introduces a subpattern for use by a subroutine call, while not involved in matching where the rule is placed. A named parenthesised expression can be executed by its name in addition to the parenthesis number. This paper presents a formalisation of subroutine calls, DEFINE rules, and named parenthesised expressions using the matching relation while attempting to mimic the behaviour of real-world regular expression engines. Also, we give an alternative constructive proof of equivalence of expressive power of regular expressions extended with subroutine calls and the class of context-free languages: a conversion between such expressions and context-free grammars. Finally, the question of whether regular expressions with operations lookaround assertion combined with subroutine call have greater expressive power than expressions with only subroutine call is answered positively.

1 Introduction

Regular expressions were introduced by Kleene[13] as a theoretical concept with expressive power equivalent to regular languages (these are further referred to as classical regular expressions or RE). This concept plays an important role in pattern matching and its variants with multiple (finite or infinite) patterns (see the taxonomy of pattern matching problems by Melichar and Holub[16]). So-called regular expressions have been implemented in many tools (e.g., UNIX text filters, text editors), programming languages, and libraries (these expressions are often referred to as extended regular expressions, practical regular expressions, or regexes). Unlike classical regular expressions, regexes "seem to have been invented entirely on the level of software implementation, without prior theoretical formalisation" (Schmid[22]). Moreover, both the syntax and semantics of the regex flavours used in implementations differ from each other (differences among the flavours were described by Friedl[8]). However, researchers have been exploring the algorithmic and language properties of practical regular expressions. Research results address properties of combinations of particular features used in regexes rather than complete flavours.

One of the fields in which this paper is concerned focuses on the expressive power of regexes and its relation to known language classes. Some syntactic constructs (not

Ondřej Guth: On Expressive Power of Regular Expressions with Subroutine Calls and Lookaround Assertions, pp. 68–82.
Proceedings of PSC 2023, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-07206-6 © Czech Technical University in Prague, Czech Republic

used in classical regular expressions) are known to be mere syntax sugar: they can be rewritten to equivalent REs. Among these constructs are positive iteration (e.g., `a+`), character class (e.g., `[abc]`), or counting constraint (interval quantifier, e.g., `a{3,8}`) as pointed out, among others, by Câmpeanu et al.[4]. However, some features of regexes impact their expressive power: nonregular languages can be matched. A backreference indicates that a substring matched by a corresponding parenthesised subpattern should be matched again at the positions where the backreference is placed. Regexes with backreferences can match a proper subset of the class of context-sensitive languages (the expressive power of backreferences was studied, among others, by Câmpeanu et al.[4], Berglund et al.[2], or Schmid[22]). The expressive power of a practical regular expression with features of RE extended by lookahead stays within regular languages (Berglund et al.[3]). Regexes with subroutine calls describe context-free languages (addressed in master's thesis by Hruša[12]). When a practical regular expression with backreferences is extended by lookaheads, its expressive power supersedes the expressive power of regex with only backreferences (Chida and Terauchi[5]). Therefore, it is natural to wonder whether adding lookahead to a regex with subroutine calls also impacts expressive power. This question is addressed in this paper.

The formalisation of syntax and semantics of features of practical regular expressions is an essential part of proving their expressive power. Aho[1] gave a relatively informal definition of a regex with backreferences: the definition uses named variables, while any variable can be reassigned. Câmpeanu, Salomaa, and Yu[4] precisely formalised the numbered backreferences. Another formalisation of backreferences, factor-referencing, was introduced by Schmid[22]: it uses named variables which can be reassigned, and unlike the first formalisation[1], it deals with details of both syntax and semantics. Regexes with lookahead were originally formalised by Morihata[17] according to Berglund et al.[3]: the definition uses lookahead language. Chida and Terauchi[6] formalised the regexes with lookaheads and numbered backreferences using the matching relation. The syntax and semantics of the regexes with numbered subroutine calls were defined by Hruša[12]. To the author's knowledge, there is no formalisation of DEFINE rules, named parenthesised expressions, or named subroutine calls. This paper fills this gap by extending the notion matching relation.

Finding the expressive power of regex flavours is motivated by more than scientific curiosity. Users of pattern matching tools need to know what can and can not be matched by particular flavours of practical regular expressions[1].

To the author's knowledge, there has been almost no research on the expressive power of regexes with subroutine calls. The first known text dealing with this gap was published as a blog post by Popov[19] providing a sketch of a reduction of context-free grammars to regexes with DEFINE rules and named subroutine calls. Popov's claim of the equivalence of expressive power of regexes with subroutine calls and context-free languages was later formally proved by Hruša[12] while using numbered-only subroutine calls. This paper gives an alternative proof to the Hruša's while using the matching relation and regexes with DEFINE rules and named subroutine calls. We hope that our approach is more straightforward and extensible in future research.

[1] As shown by several discussions at Stack Overflow, for example, `https://stackoverflow.com/q/35449863`, `https://stackoverflow.com/q/2974210`, or `https://stackoverflow.com/q/4840988`.

To the author's knowledge, there is no peer-reviewed publication on the expressive power of practical regular expressions with both subroutine calls and lookaround assertions. The problem seems to have a solution due to Popov's[19] sketch (or, more precisely, an idea) of a reduction from context-sensitive grammars to regexes[2]. However, we show a counterexample. In addition, we present proof that the expressive power of practical regular expressions with lookaround assertions and subroutine calls is greater than the expressive power of expressions with only subroutine calls.

This paper focuses on the expressive power of regex with subroutine calls (subpattern recursion) and lookaround assertions. Our contributions are the following:

- We give a formalisation of practical regular expressions by extending the notion of the matching relation. In particular, this paper gives the formalisation of constructs named parenthesised expression, DEFINE rule, and both numbered and named subroutine calls. Our formalisation mimics the syntax and semantics of Perl-Compatible Regular Expressions (PCRE2, as documented on the manual page[10] and as we experimentally verified). At the same time, it also works for Perl regular expressions[18] and Ruby Regexp class[20].
- We prove that the expressive power of regex with concatenation, alternative, DEFINE rule, and subroutine call is equal to the class of context-free languages. This proof is based on the matching relation and works for regex with numbered and named subroutines.
- We show that adding lookaround assertions to regexes with subroutine calls extends their expressive power beyond context-free languages. In addition, we also show that the equivalence of the expressive power of these regexes to the class of context-sensitive languages remains an open problem.

This paper is structured as follows. In section 2, we give notational preliminaries. Section 3 contains the formalisation of practical regular expressions with subroutine calls. In section 4, we present proof that the expressive power of regexes with subroutine calls is equal to context-free languages: a conversion between such a regex and context-free grammar. Section 5 contains proof that adding lookaround assertion extends the expressive power of practical regular expressions with subroutine call. In section 6, we conclude the paper and discuss future work.

2 Preliminaries

The set of natural numbers is denoted by $\mathbb{N}$ and is without zero. The mathematical symbols $\emptyset$, $\cup$, $\cap$, $\setminus$, $\wedge$, $\forall$, and $\nexists$ denote the empty set, set union, set intersection, set difference, logical conjunction, universal quantifier, and negated existential quantifier, respectively. If $\mathcal{V}$ and $\mathcal{X}$ are sets, $\mathcal{V}$ being a subset (or a strict subset) of $\mathcal{X}$ is denoted by $\mathcal{V} \subseteq \mathcal{X}$ (or $\mathcal{V} \subsetneq \mathcal{X}$, respectively). An *alphabet*, denoted by $\mathcal{A}$, is a finite nonempty set whose elements are called *symbols*. A *string* over $\mathcal{A}$ is a finite sequence of elements of $\mathcal{A}$. The empty sequence is called the *empty string* and is denoted by ε. *The set of all strings* over $\mathcal{A}$ is denoted by $\mathcal{A}^*$ and *the set of all nonempty strings* over $\mathcal{A}$ is denoted by $\mathcal{A}^+$. The *length* of a string $\boldsymbol{y}$ is the length of the sequence associated with

[2] The reduction of context-sensitive grammars to regexes seem to be trusted: for example, a Stack Exchange contributor claims that regexes with subroutine calls and lookaround assertions can express any context-sensitive language using Popov's argument: `https://cs.stackexchange.com/q/143221`.

$\boldsymbol{y}$ and is denoted by $|\boldsymbol{y}|$. By $\boldsymbol{y}[i]$, where $i \in \mathbb{N} \wedge i \in \{1, \ldots, |\boldsymbol{y}|\}$, we denote the symbol at the index i of $\boldsymbol{y}$. The concatenation of strings $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ is denoted by $\boldsymbol{y}_1\boldsymbol{y}_2$. Thus, $\boldsymbol{y} = \boldsymbol{y}[1]\boldsymbol{y}[2]\ldots\boldsymbol{y}[|\boldsymbol{y}|]$. The substring of $\boldsymbol{y}$ that starts at the index i and ends at the index g is denoted by $\boldsymbol{y}[i..g]$; that is, $\boldsymbol{y}[i..g] = \boldsymbol{y}[i]\boldsymbol{y}[i+1]\ldots\boldsymbol{y}[g]$. A *language* over an alphabet $\mathcal{A}$ is a set of strings over $\mathcal{A}$, denoted by $\mathcal{L} \subseteq \mathcal{A}^*$. The *concatenation of languages* $\mathcal{L}_1, \mathcal{L}_2$ is denoted by $\mathcal{L}_1 \cdot \mathcal{L}_2$ and is defined as $\mathcal{L}_1 \cdot \mathcal{L}_2 = \{\boldsymbol{y} = \boldsymbol{y}_1\boldsymbol{y}_2 : \boldsymbol{y}_1 \in \mathcal{L}_1 \wedge \boldsymbol{y}_2 \in \mathcal{L}_2\}$. The *closure of a language* $\mathcal{L}$ is denoted by $\mathcal{L}^*$ and is defined as $\bigcup_{g\geq 0} \mathcal{L}^g$ where $\mathcal{L}^0 = \{\varepsilon\}$, $\mathcal{L}^1 = \mathcal{L}$, and for $g > 1$: $\mathcal{L}^g = \mathcal{L} \cdot \mathcal{L}^{g-1}$.

The following grammar-related notions follow the conventions of Hopcroft et al.[11] and Mateescu et al.[15]. A *grammar* is a quadruple $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ where $\mathcal{A} \cap \mathcal{V} = \emptyset$, $S \in \mathcal{V}$, and $\mathcal{R}$ is a set of pairs $(\boldsymbol{v}_1, \boldsymbol{v}_2)$ where $\boldsymbol{v}_1 \in (\mathcal{A} \cup \mathcal{V})^* \cap \mathcal{V}^+$ and $\boldsymbol{v}_2 \in (\mathcal{A} \cup \mathcal{V})^*$. The sets $\mathcal{V}$ (nonterminals), $\mathcal{A}$, and $\mathcal{R}$ are finite. We use the following naming and typographic conventions: $a \in \mathcal{A}$, $N \in \mathcal{V}$, $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}^*$, and $\boldsymbol{v} \in (\mathcal{A} \cup \mathcal{V})^*$ (bold italic sans serif for a string that can contain a nonterminal). The members of the set $\mathcal{R}$ are called *productions* and are written with $\to$ as a delimiter of the left- and right-hand side. Multiple productions with the same left-hand side can be contracted: for instance, if $\boldsymbol{v} \to \boldsymbol{v}_1, \boldsymbol{v} \to \boldsymbol{v}_2 \in \mathcal{R}$ then we can write $\boldsymbol{v} \to \boldsymbol{v}_1 \mid \boldsymbol{v}_2 \in \mathcal{R}$. A *derivation step* in grammar $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ is denoted by $\underset{\mathfrak{G}}{\Rightarrow}$ and defined as follows. If $\boldsymbol{v}_1 \in (\mathcal{A} \cup \mathcal{V})^+$, $\boldsymbol{p}, \boldsymbol{s}, \boldsymbol{v}_2 \in (\mathcal{A} \cup \mathcal{V})^*$, and $\boldsymbol{v}_1 \to \boldsymbol{v}_2 \in \mathcal{R}$ then $\boldsymbol{p}\boldsymbol{v}_1\boldsymbol{s} \underset{\mathfrak{G}}{\Rightarrow} \boldsymbol{p}\boldsymbol{v}_2\boldsymbol{s}$. The transitive closure of $\underset{\mathfrak{G}}{\Rightarrow}$ is denoted by $\underset{\mathfrak{G}}{\Rightarrow}^+$, and the reflective and transitive closure is denoted by $\underset{\mathfrak{G}}{\Rightarrow}^*$. If $S \underset{\mathfrak{G}}{\Rightarrow}^* \boldsymbol{v}$ then the string $\boldsymbol{v}$ is called a *sentential form* and $S \underset{\mathfrak{G}}{\Rightarrow}^* \boldsymbol{v}$ is called a *derivation* of $\boldsymbol{v}$ in $\mathfrak{G}$. The *language generated by grammar* $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ is denoted by $\mathrm{L}(\mathfrak{G})$ and is defined as $\mathrm{L}(\mathfrak{G}) = \{\boldsymbol{x} \in \mathcal{A}^* : S \underset{\mathfrak{G}}{\Rightarrow}^* \boldsymbol{x}\}$.

We relate language classes of practical regular expressions to the Chomsky hierarchy ([7]). A *context-sensitive grammar* is a grammar $(\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ where every member of $\mathcal{R}$ is of the form $\boldsymbol{p}N\boldsymbol{s} \to \boldsymbol{p}\boldsymbol{v}\boldsymbol{s}$ where $\boldsymbol{v} \neq \varepsilon$, and $\boldsymbol{p}, \boldsymbol{s} \in (\mathcal{A} \cup \mathcal{V})^*$. A *context-free grammar* is a grammar $(\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ where all members of $\mathcal{R}$ are of the form $N \to \boldsymbol{v}$.

A derivation $S \underset{\mathfrak{Gl}}{\Rightarrow}^* \boldsymbol{v}$ in a context-free grammar $\mathfrak{G}$ is called *the leftmost* if at each derivation step we replace the leftmost nonterminal (in a sentential form) by the right-hand side of one of its productions. A context-free grammar $(\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ is in *Greibach normal form* if every production is of the form $N \to a\boldsymbol{v}$. [9],[14, lecture 21]

A *context-sensitive language* is a language that is generated by some context-sensitive grammar. Context-free languages are defined likewise. The class of context-free languages is denoted by $\mathbb{L}_{\mathrm{CF}}$.

2.1 Regular expressions

The set of all classical regular expressions over an alphabet $\mathcal{A}$ is denoted by $\mathbb{E}_{\mathrm{C},\mathcal{A}}$. The syntax of classical regular expressions is given as follows (as defined in the literature[11,13] and adapted to conform conventions used in tools and libraries; operators are ordered by their precedence from the highest):

1. $\emptyset$ and ε are regular expressions,
2. for $a \in \mathcal{A}$, a is a regular expression,
3. for $\boldsymbol{r} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $(\boldsymbol{r})$ (parenthesised expression) is a regular expression,
4. for $\boldsymbol{r} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $\boldsymbol{r}^*$ (iteration, Kleene star) is a regular expression.
5. for $\boldsymbol{r_1}, \boldsymbol{r_2} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $\boldsymbol{r_1} \cdot \boldsymbol{r_2}$ or $\boldsymbol{r_1}\boldsymbol{r_2}$ (concatenation) is a regular expression,
6. for $\boldsymbol{r_1}, \boldsymbol{r_2} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $\boldsymbol{r_1} \mid \boldsymbol{r_2}$ (alternative) is a regular expression.

The semantics of classical regular expressions (where the language matched by RE $\boldsymbol{r}$ is denoted by $\mathrm{L}(\boldsymbol{r})$) is given as follows[11]:

- $\mathrm{L}(\emptyset) = \emptyset$,
- $\mathrm{L}(\varepsilon) = \{\varepsilon\}$,
- for $a \in \mathcal{A}$, $\mathrm{L}(a) = \{a\}$,
- for $\boldsymbol{r} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $\mathrm{L}((\boldsymbol{r})) = \mathrm{L}(\boldsymbol{r})$,
- for $\boldsymbol{r} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $\mathrm{L}(\boldsymbol{r}^*) = \Big(\mathrm{L}(\boldsymbol{r})\Big)^*$,
- for $\boldsymbol{r_1}, \boldsymbol{r_2} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $\mathrm{L}(\boldsymbol{r_1 r_2}) = \mathrm{L}(\boldsymbol{r_1}) \cdot \mathrm{L}(\boldsymbol{r_2})$,
- for $\boldsymbol{r_1}, \boldsymbol{r_2} \in \mathbb{E}_{\mathrm{C},\mathcal{A}}$, $\mathrm{L}(\boldsymbol{r_1} \mid \boldsymbol{r_2}) = \mathrm{L}(\boldsymbol{r_1}) \cup \mathrm{L}(\boldsymbol{r_2})$.

The set of all practical regular expressions with operations iteration, concatenation, alternative, DEFINE rule, lookaround assertion, subroutine calls, and numbered and named parenthesised subexpressions over alphabet $\mathcal{A}$ is denoted by $\mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ where the set of labels of named parenthesised expressions is denoted by $\mathcal{X}$ ($\mathcal{X} \cap \mathcal{A} = \emptyset$). The set of all regexes without lookaround assertions is denoted by $\mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$. Each practical regular expression consists of characters that may occur in the input string (i.e., $a \in \mathcal{A}$) and metacharacters that cannot occur in the input[3]: $(,), ?, =, <, > \notin \mathcal{A}$. For brevity, we write parentheses that denote a parenthesised expression with their assigned number, e.g., $({}_l)_l$. Our syntax closely follows the flavour PCRE2:

- the empty string, a character, iteration, concatenation, and alternative are defined the same way as for classical regular expressions,
- $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}} \wedge l \in \mathbb{N} : ({}_l\boldsymbol{r})_l \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$,
- $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}} \wedge l \in \mathbb{N} \wedge N \in \mathcal{X} : ({}_l?{<}N{>}\boldsymbol{r})_l \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ (parenthesised expression named N and numbered l),
- $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}} : (?(\mathrm{DEFINE})\boldsymbol{r}) \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ (DEFINE rule),
- $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}} : (?{=}\boldsymbol{r}) \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ (lookahead),
- $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}} : (?{<}{=}\boldsymbol{r}) \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ (lookbehind),
- $l \in \mathbb{N} : (?l) \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ (numbered subroutine call),
- $N \in \mathcal{X} : (?\mathrm{P}{>}N) \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ (named subroutine call).

The numbering of parenthesised expressions, both named and unnamed, is unique. (Note that neither parentheses around lookahead, lookbehind, nor subroutine call delimit a parenthesised expression.)

The semantics of regexes with numbered backreferences and lookaround assertions was defined using the matching relation by Chida and Terauchi[5,6]. We closely follow their definition[4]. A matching relation $\rightsquigarrow$ is of the form $(\boldsymbol{r}, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}$ where $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$, $\boldsymbol{x} \in \mathcal{A}^*$, $i \in \mathbb{N} \wedge i \leq |\boldsymbol{x}|$, and $\mathcal{R} = \{i : i \in \mathbb{N} \wedge i \leq |\boldsymbol{x}| + 1\}$ (matching result). The rules for deriving the matching relation for practical regular expressions with lookaround assertions are as follows:

$$\frac{}{(\emptyset, \boldsymbol{x}, i) \rightsquigarrow \emptyset}$$

$$\frac{}{(\varepsilon, \boldsymbol{x}, i) \rightsquigarrow \{i\}}$$

[3] For brevity, we deviate from the way real-world engines treat metacharacters: they can occur in the input and can be matched in a regex when following a special escaping metacharacter (some flavours can match some metacharacters even without escaping), mostly the backslash. We refer the reader to Friedl[8].

[4] We omit capturing environment as this paper does not deal with backreferences.

$$\frac{a \in \mathcal{A} \wedge i \leq |\boldsymbol{x}| \wedge \boldsymbol{x}[i] = a}{(a, \boldsymbol{x}, i) \rightsquigarrow \{i+1\}}, \frac{a \in \mathcal{A} \wedge (i > |\boldsymbol{x}| \vee \boldsymbol{x}[i] \neq a)}{(a, \boldsymbol{x}, i) \rightsquigarrow \emptyset}$$

$$\frac{(\boldsymbol{r}, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R} \wedge \forall i_h \in \mathcal{R} \setminus \{i\} : (\boldsymbol{r}^*, \boldsymbol{x}, i_h) \rightsquigarrow \mathcal{R}_h}{(\boldsymbol{r}^*, \boldsymbol{x}, i) \rightsquigarrow \{i\} \cup \bigcup_{1 \leq h \leq |\mathcal{R} \setminus \{i\}|} \mathcal{R}_h} \tag{1}$$

$$\frac{(\boldsymbol{r}_1, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R} \wedge \forall (i_h) \in \mathcal{R} : (\boldsymbol{r}_2, \boldsymbol{x}, i_h) \rightsquigarrow \mathcal{R}_h}{(\boldsymbol{r}_1 \boldsymbol{r}_2, \boldsymbol{x}, i) \rightsquigarrow \bigcup_{1 \leq h \leq |\mathcal{R}|} \mathcal{R}_h} \tag{2}$$

$$\frac{(\boldsymbol{r}_1, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}_1 \wedge (\boldsymbol{r}_2, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}_2}{(\boldsymbol{r}_1 \mid \boldsymbol{r}_2, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}_1 \cup \mathcal{R}_2} \tag{3}$$

$$\frac{(\boldsymbol{r}, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}{((\boldsymbol{r}), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}} \tag{4}$$

$$\frac{(\boldsymbol{r}, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}{((?{=}\boldsymbol{r}), \boldsymbol{x}, i) \rightsquigarrow \{i \wedge \mathcal{R} \neq \emptyset\}} \tag{5}$$

$$\frac{\boldsymbol{y} \in \mathcal{A}^* \wedge (\boldsymbol{y}, \boldsymbol{x}[i - |\boldsymbol{y}|..i-1], 1) \rightsquigarrow \mathcal{R}}{((?{<}{=}\boldsymbol{y}), \boldsymbol{x}, i) \rightsquigarrow \{i \wedge \mathcal{R} \neq \emptyset\}} \tag{6}$$

The language of a regex $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ is $\mathrm{L}(\boldsymbol{r}) = \{\boldsymbol{x} : (\boldsymbol{r}, \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R} \wedge |\boldsymbol{x}| + 1 \in \mathcal{R}\}$. We also say that a string $\boldsymbol{x} \in \mathrm{L}(\boldsymbol{r})$ *matches* a regex $\boldsymbol{r}$ (similarly, $\mathrm{L}(\boldsymbol{r})$ is the language matched by $\boldsymbol{r}$). The class of all languages that can be matched by the regexes of $\mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ is denoted by $\mathbb{L}_{\mathrm{E_{LS}}}$. The class of all languages that can be matched by the regexes of $\mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ is denoted by $\mathbb{L}_{\mathrm{E_S}}$.

3 Formalizing expressions with subroutine calls and lookaround assertions

We now formally define the semantics of practical regular expressions with named parenthesised expressions, DEFINE rules, and numbered and named subroutine calls. Our definition is an extension of the matching relation in the previous section. In this section, the following notation is used: $i, l \in \mathbb{N}; N \in \mathcal{X}; \boldsymbol{r}, \boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3 \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}; \boldsymbol{x} \in \mathcal{A}^*$.

The subroutine call attempts to match a given parenthesised expression at a current position. To be able to use the subexpression given the parenthesis number, the partial function $\sigma_{\boldsymbol{r}}$ is computed before the matching of regex $\boldsymbol{r}$ starts; it is defined as follows:

$$\sigma_{\boldsymbol{r}} : \mathbb{N} \to \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}} \wedge \boldsymbol{r} = \boldsymbol{r}_1({}_l\boldsymbol{r}_2)_l\boldsymbol{r}_3 \text{ implies } \sigma_{\boldsymbol{r}}(l) = \boldsymbol{r}_2$$

If no confusion can arise, we use σ for simplicity. In addition to being unambiguous, the numbering of parenthesised expressions (both named and unnamed) is not important for our results. Our definition conforms to some flavours of practical regular expressions.[5]

Matching a numbered subroutine call means matching the subpattern given in l-th parentheses from the current position. The subroutine call can be located anywhere related to the referred subpattern (i.e., both forward and recursive calls are valid).

$$\frac{(\sigma(l), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}{((?l), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}$$

[5] Namely PCRE2 and Perl. Both PCRE2 and Perl even support enabling duplicate parenthesis numbers (it is not the default). Our definition needs to be modified for the Ruby Regexp class: numbers cannot be used when at least one named expression is present.

Matching a named subroutine call uses the parenthesised expression assigned to the given name. Named parenthesised expressions can be identified by either their name or number. Thus, a name is just an alias for the parenthesis number. The partial function $\nu_{\boldsymbol{r}}$ is computed before the matching of $\boldsymbol{r}$ starts. If the name is used for multiple parenthesised expressions, $\nu_{\boldsymbol{r}}$ assigns the name to the leftmost parentheses.

$\nu_{\boldsymbol{r}} : \mathcal{X} \to \mathbb{N} \wedge \boldsymbol{r} = \boldsymbol{r}_1({}_l?{<}N{>}\boldsymbol{r}_2)_l\boldsymbol{r}_3$ implies $\nu_{\boldsymbol{r}}(N) = l \wedge \nexists l' < l : \boldsymbol{r} = \boldsymbol{r}'_1({}_{l'}?{<}N{>}\boldsymbol{r}'_2)_{l'}\boldsymbol{r}'_3$

If no confusion can arise, we use ν for simplicity.

$$\frac{(\sigma(\nu(N)), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}{((?\mathrm{P}{>}N), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}} \tag{7}$$

This matching relation for named subroutine call closely mimics the semantics of PCRE2, Perl, and Ruby.

Any regex can be wrapped in a DEFINE rule. In addition to the possibility of extending ν or σ, the DEFINE rule does not affect the matching.

$$\frac{}{((?(\mathrm{DEFINE})\boldsymbol{r}), \boldsymbol{x}, i) \rightsquigarrow \{i\}} \tag{8}$$

DEFINE rules with the above-defined semantics are supported by PCRE2 and Perl.

4 Expressive power of subroutine call

We give a rigorous proof of the equivalence of expressive power of context-free languages and practical regular expressions with subroutine calls. Our proof is based on the matching relation and extends Hruša's work[12], which is based on Popov's claim[19].

Theorem 1. $\mathbb{L}_{\mathrm{E_S}} = \mathbb{L}_{\mathrm{CF}}$

To prove the class equivalence, we first show that every context-free grammar can be converted to a regex with subroutine calls. Later, we show that every such regex can be converted into context-free grammar.

Lemma 2. $\mathbb{L}_{\mathrm{CF}} \subseteq \mathbb{L}_{\mathrm{E_S}}$.

We show that every context-free grammar can be expressed by a practical regular expression with the following sufficient operations: concatenation, alternative, DEFINE rule with named parenthesised expression, and named subroutine call. Intuitively, such a regex contains all the building blocks of context-free grammar: concatenation, alternative, and the ability to reuse a subexpression, even recursively. The conversion is formally defined by algorithm 1 and definition 3. The restriction of conversion to Greibach normal form grammar does not change the expressive power: Any context-free grammar (and therefore any context-free language) can be expressed by a Greibach normal form grammar using a known transformation.[9][14, lecture 21]

Definition 3. *Let us have a context-free grammar* $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ *and a regex* $\boldsymbol{r} \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ *where* $\mathcal{V} = \mathcal{X}$. *The function* $\mathrm{rx} : (\mathcal{V} \cup \mathcal{A})^* \to \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ *is defined as follows: let* $\boldsymbol{v}_1, \boldsymbol{v}_2 \in (\mathcal{A} \cup \mathcal{V})^*$, $a \in \mathcal{A}$, *and* $N \in \mathcal{V}$ *then* $\mathrm{rx}(\varepsilon) = \varepsilon$, $\mathrm{rx}(a) = a$, $\mathrm{rx}(N) = (?\mathrm{P}{>}N)$, *and* $\mathrm{rx}(\boldsymbol{v}_1\boldsymbol{v}_2) = \mathrm{rx}(\boldsymbol{v}_1)\,\mathrm{rx}(\boldsymbol{v}_2)$.

Algorithm 1 Conversion of a context-free grammar to a regex

Input: a context-free grammar $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ in Greibach normal form
Output: a regex $\boldsymbol{r} \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ such that $\mathrm{L}(\mathfrak{G}) = \mathrm{L}(\boldsymbol{r})$

1. initialize $\boldsymbol{r} = \varepsilon$ and consider $\mathcal{X} = \mathcal{V}$
2. for all productions with a non-terminal N on the left-hand side ($N \to \boldsymbol{v}_{N1} \mid \cdots \mid \boldsymbol{v}_{Nm_N} \in \mathcal{R}$):
   (a) let $\boldsymbol{r}_{Ng}$ ($1 \leq g \leq m_N$) be constructed from the g-th right-hand side of the production for N ($\boldsymbol{v}_{Ng}$) by replacing the non-terminals with subroutine call: $\boldsymbol{r}_{Ng} = \mathrm{rx}(\boldsymbol{v}_{Ng})$
   (b) add a DEFINE rule with parenthesised expression named N containing the strings $\boldsymbol{r}_{Ng}$ constructed from right-hand side of these productions, i.e., let $\boldsymbol{r} = \boldsymbol{r}(?(\mathrm{DEFINE})(?{<}N{>}\boldsymbol{r}_{N1} \mid \boldsymbol{r}_{N2} \mid \cdots \mid \boldsymbol{r}_{Nm_N}))$
3. add the matching of the initial symbol, i.e., let $\boldsymbol{r} = \boldsymbol{r}(?\mathrm{P}{>}S)$

Lemma 4. *Let us have a Greibach normal form grammar $\mathfrak{G}$ and a practical regular expression $\boldsymbol{r}$ such that $\boldsymbol{r}$ is constructed by the algorithm 1 from $\mathfrak{G}$. For any possible derivation step in $\mathfrak{G}$, a possible step exists in the matching relation for $\boldsymbol{r}$.*

Proof. Let the grammar be $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$. Without loss of generality, we can assume that only the leftmost derivations are used to derive any sentential form. Let $\boldsymbol{p}N\boldsymbol{s} \underset{\mathfrak{Gl}}{\Longrightarrow} \boldsymbol{p}\boldsymbol{v}\boldsymbol{s}$ be a derivation step (recall that this derivation step is possible only if $N \to \boldsymbol{v} \in \mathcal{R}$) where $\boldsymbol{p} \in \mathcal{A}^*$, $\boldsymbol{s} \in \mathcal{V}^*$ and $\boldsymbol{v} \in (\mathcal{V} \cup \mathcal{A})^+$. Recall that due to algorithm 1, $\boldsymbol{r}$ is in the following form:

$$(?(\mathrm{DEFINE})\ldots)\cdots(?(\mathrm{DEFINE})(?{<}N{>}\cdots \mid \mathrm{rx}(\boldsymbol{v}) \mid \cdots))\cdots(?\mathrm{P}{>}S) \tag{9}$$

The following steps of the matching relation show that an expression of the form $\boldsymbol{p}(?\mathrm{P}{>}N)\,\mathrm{rx}(\boldsymbol{s})$ (from step 2a follows that $\mathrm{rx}(\boldsymbol{p}) = \boldsymbol{p}$) can always be resolved by the concatenation of $\boldsymbol{p}$, $\mathrm{rx}(\boldsymbol{v})$, and $\mathrm{rx}(\boldsymbol{s})$.

$$\dfrac{\dfrac{\dfrac{\dots}{(\mathrm{rx}(\boldsymbol{v}_{N1}),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hN1}} \cdots \dfrac{(\mathrm{rx}(\boldsymbol{v}),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hNg}}{(\mathrm{rx}(\boldsymbol{v}),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hNg}} \cdots \dfrac{\dots}{(\mathrm{rx}(\boldsymbol{v}_{Nm_N}),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hNm_N}}}{(\mathrm{rx}(\boldsymbol{v}_{N1}) \mid \cdots \mid \mathrm{rx}(\boldsymbol{v}) \mid \cdots \mid \mathrm{rx}(\boldsymbol{v}_{Nm_N}),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hN} = \bigcup_{1\leq h'\leq m_N} \mathcal{R}_{hNh'}}}{\dfrac{\dfrac{\dots}{(\boldsymbol{p},\boldsymbol{x},i) \rightsquigarrow \mathcal{R}'} \quad \dfrac{\dfrac{(\sigma(l),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hN}}{(\sigma(\nu(N)),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hN}}}{(?\mathrm{P}{>}N),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_{hN}} \quad \dfrac{\dots}{\forall i_{hN} \in \mathcal{R}_{hN} : (\mathrm{rx}(\boldsymbol{s}),\boldsymbol{x},i_{hN}) \rightsquigarrow \mathcal{R}_{h_{\boldsymbol{s}}N}}}{\forall i_h \in \mathcal{R}' : ((?\mathrm{P}{>}N)\,\mathrm{rx}(\boldsymbol{s}),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_h = \bigcup_{1\leq h_{\boldsymbol{s}}\leq|\mathcal{R}_{hN}|} \mathcal{R}_{h_{\boldsymbol{s}}N}}}{(\boldsymbol{p}(?\mathrm{P}{>}N)\,\mathrm{rx}(\boldsymbol{s}),\boldsymbol{x},i) \rightsquigarrow \bigcup_{1\leq h\leq|\mathcal{R}'|} \mathcal{R}_h}$$

Note that $\mathcal{R}' = \{i + |\boldsymbol{p}|\}$ if $\boldsymbol{x}[i..i + |\boldsymbol{p}| - 1] = \boldsymbol{p}$ and $\mathcal{R}' = \emptyset$ otherwise. Furthermore, the matching result for $\mathrm{rx}(\boldsymbol{v})$, $\mathcal{R}_{hNg}$, is involved in the expression matching. □

The following lemma holds due to lemma 4.

Lemma 5. *Let us have a Greibach normal form grammar $\mathfrak{G}$ and a practical regular expression $\boldsymbol{r}$ such that $\boldsymbol{r}$ is constructed by the algorithm 1 from $\mathfrak{G}$. Then $\mathrm{L}(\mathfrak{G}) \subseteq \mathrm{L}(\boldsymbol{r})$.*

Lemma 6. *Let us have a Greibach normal form grammar $\mathfrak{G}$ and a practical regular expression $\boldsymbol{r}$ such that $\boldsymbol{r}$ is constructed by the algorithm 1 from $\mathfrak{G}$. Then $\mathrm{L}(\boldsymbol{r}) \subseteq \mathrm{L}(\mathfrak{G})$.*

Proof. Let the grammar be $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ and $\boldsymbol{x} \in \mathrm{L}(\mathfrak{G})$. The general form of a regex constructed by the algorithm is (9) while all subpatterns $\mathrm{rx}(\boldsymbol{v})$ of $\boldsymbol{r}$ are of

the form $a_1\,\mathrm{rx}(\boldsymbol{s})$ where $a_1 \in \mathcal{A}$ and $\boldsymbol{s} \in \mathcal{V}^*$ (due to the normal Greibach form). Such a subpattern can exist only if $N \to a_1\boldsymbol{s} \in \mathcal{R}$ (due to step 2a). Following the matching relation, the initial step in matching, that is, (?P>S) which applies some subpattern of named parentheses S, $a_2\,\mathrm{rx}(\boldsymbol{s'})$, is possible only when $\boldsymbol{x}[1] = a_2$; such a subpattern is constructed only if $S \to a_2\boldsymbol{s'} \in \mathcal{R}$. Suppose, for contradiction, that $a_2\,\mathrm{rx}(\boldsymbol{s'})$ matches $\boldsymbol{x}$ and $\boldsymbol{x} \notin \mathrm{L}(\mathfrak{G})$. As $\boldsymbol{s'} = N_1N_2\ldots N_{|\boldsymbol{s'}|}$, the only possibility is that any of (?P>N_g) matches a substring $\boldsymbol{y}$ of $\boldsymbol{x}$ and $N_g \underset{\mathfrak{G}}{\Rightarrow}^* \boldsymbol{y}$ is not possible. However, due to the algorithm 1, the named parenthesised expressions in $\boldsymbol{r}$ contain only subpatterns corresponding to the right-hand sides of the productions in $\mathfrak{G}$. Due to the Greibach normal form, $\boldsymbol{r}$ can only match $\boldsymbol{x}$ using subpatterns in the same order as productions of $\mathfrak{G}$ are applied when generating $\boldsymbol{x}$. In any situation when a subroutine call (?P>N_g) occurs, $\mathfrak{G}$ can use any production for N_g, because the application of productions in context-free grammar is not restricted by their order or context. □

The validity of lemma 2 was shown by lemmas 5 and 6: any context-free language can be expressed by a regex with subroutine calls.

Lemma 7. $\mathbb{L}_{\mathrm{E_S}} \subseteq \mathbb{L}_{\mathrm{CF}}$.

We show that an equivalent context-free grammar can express any practical regular expression with operations concatenation, alternative, DEFINE rule with named parenthesised expression, and named subroutine call. We begin by showing that removing the other valid operations in $\mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ does not change the expressive power.

Lemma 8 (Redundancy of Kleene star). *Every regex of the form $\boldsymbol{r}^* \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ can be expressed as*

$$(?(\mathrm{DEFINE})(?<N>\varepsilon \mid \boldsymbol{r}(?\mathrm{P}>N)))(?\mathrm{P}>N) \tag{10}$$

where $N \notin \mathcal{X}$.

Proof. The matching relation can be derived as follows:

$$\dfrac{\dfrac{}{(\varepsilon,\boldsymbol{x},i) \rightsquigarrow \{i\}} \quad \dfrac{(\boldsymbol{r},\boldsymbol{x},i) \rightsquigarrow \mathcal{R}' \wedge \forall i_h \in \mathcal{R}' : ((?\mathrm{P}>N),\boldsymbol{x},i_h) \rightsquigarrow \mathcal{R}_h}{(\boldsymbol{r}(?\mathrm{P}>N),\boldsymbol{x},i) \rightsquigarrow \mathcal{R} = \bigcup_{1\le h\le|\mathcal{R}'|}\mathcal{R}_h}}{(\varepsilon \mid \boldsymbol{r}(?\mathrm{P}>N),\boldsymbol{x},i) \rightsquigarrow \mathcal{R} \cup \{i\}}$$

$$\dfrac{\dfrac{(?(\mathrm{DEFINE})(?<N>\varepsilon \mid \boldsymbol{r}(?\mathrm{P}>N))),\boldsymbol{x},i) \rightsquigarrow \{i\}}{((?\mathrm{P}>N),\boldsymbol{x},i) \rightsquigarrow \mathcal{R} \cup \{i\}} \quad \dfrac{(\sigma(\nu(N)),\boldsymbol{x},i) \rightsquigarrow \mathcal{R} \cup \{i\}}{((?\mathrm{P}>N),\boldsymbol{x},i) \rightsquigarrow \mathcal{R} \cup \{i\}}}{((10),\boldsymbol{x},i) \rightsquigarrow \mathcal{R} \cup \{i\}}$$

It is clear that the matching result is the same as that of the Kleene star (1). Although the matching relation of the Kleene star (1) excludes i from rematching $\boldsymbol{r}^*$, this exclusion does not affect the positions in its matching result. □

The following two redundancies of a standalone parenthesised expression are straightforward and thus are left without proof. Each occurrence of parenthesised expression put inside a DEFINE rule retains its parenthesis number and name, and thus it does not affect any subroutine call.

Lemma 9 (Redundancy of named parenthesised expression). *A regex in the form $(?<N>\boldsymbol{r}) \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ (named parenthesised expression outside the DEFINE rule) can be expressed as*

$$(?(\mathrm{DEFINE})(?<N>\boldsymbol{r}))(?\mathrm{P}>N)$$

Lemma 10 (Redundancy of numbered parenthesised expression). *A regex in the form* $({}_l\boldsymbol{r})_l \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ *(numbered parenthesised expression outside the DEFINE rule) can be expressed as*

$$(?(\mathrm{DEFINE})({}_l?{<}N{>}\boldsymbol{r})_l)(?\mathrm{P}{>}N)$$

where $N \notin \mathcal{X}$.

The redundancy of the numbered subroutine call (and its replacement with the named subroutine call) is straightforward and thus left without proof.

Lemma 11 (Redundancy of numbered subroutine call). *Let* $\boldsymbol{r} \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ *be a regex with operations concatenation, alternative, DEFINE rule with named parenthesised expression, named subroutine call, and numbered subroutine call. To construct a regex* $\boldsymbol{r}' \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ *such that* $\mathrm{L}(\boldsymbol{r}) = \mathrm{L}(\boldsymbol{r}')$ *and* $\boldsymbol{r}'$ *does not contain numbered subroutine call,* $\boldsymbol{r}'$ *is the same as* $\boldsymbol{r}$ *with the following modifications: any occurrence of* $(?l)$ *from* $\boldsymbol{r}$ *is replaced with* $(?\mathrm{P}{>}N)$ *in* $\boldsymbol{r}'$ *where* l *refers to* $(?(\mathrm{DEFINE})({}_l?{<}N{>}\boldsymbol{r}_l)_l)$.

The conversion of a regex to a context-free grammar is formally defined in algorithm 2, which is inspired by Thompson's[23] pattern matching algorithm as presented by Hopcroft et al.[11, theorem 3.7] and closure properties of context-free languages studied by Scheinberg[21] as presented by Kozen[14].

Algorithm 2 Conversion of a regex to a context-free grammar

Input: a practical regular expression $\boldsymbol{r} \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ with operations concatenation, alternative, DEFINE rule with named parenthesised expression, and named subroutine call
Output: a context-free grammar $\mathfrak{G}_{\boldsymbol{r}} = (\mathcal{V}_{\boldsymbol{r}}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}}, S_{\boldsymbol{r}})$ such that $\mathrm{L}(\boldsymbol{r}) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}})$

1. construct grammars for elementary expressions
   - $\mathfrak{G}_{\emptyset} = (\{S_{\emptyset}\}, \mathcal{A}, \emptyset, S_{\emptyset})$
   - $\mathfrak{G}_{\varepsilon} = (\{S_{\varepsilon}\}, \mathcal{A}, \{S_{\varepsilon} \to \varepsilon\}, S_{\varepsilon}\})$
   - $\mathfrak{G}_{a} = (\{S_a\}, \mathcal{A}, \{S_a \to a\}, S_a) : a \in \mathcal{A}$
   - $\mathfrak{G}_{(?\mathrm{P}>N)} = (\{S_{(?\mathrm{P}>N)}, N\}, \mathcal{A}, \{S_{(?\mathrm{P}>N)} \to N\}, S_{(?\mathrm{P}>N)}) : N \in \mathcal{X}$
2. iteratively construct grammars for operations in a regex (suppose $\mathfrak{G}_{\boldsymbol{r}_1} = (\mathcal{V}_{\boldsymbol{r}_1}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}_1}, S_{\boldsymbol{r}_1})$ and $\mathfrak{G}_{\boldsymbol{r}_2} = (\mathcal{V}_{\boldsymbol{r}_2}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}_2}, S_{\boldsymbol{r}_2})$ are already constructed for $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, respectively)
   - $\mathfrak{G}_{\boldsymbol{r}_1\boldsymbol{r}_2} = (\mathcal{V}_{\boldsymbol{r}_1} \cup \mathcal{V}_{\boldsymbol{r}_2} \cup \{S_{\boldsymbol{r}_1\boldsymbol{r}_2}\}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}_1} \cup \mathcal{R}_{\boldsymbol{r}_2} \cup \{S_{\boldsymbol{r}_1\boldsymbol{r}_2} \to S_{\boldsymbol{r}_1}S_{\boldsymbol{r}_2}\}, S_{\boldsymbol{r}_1\boldsymbol{r}_2}) : S_{\boldsymbol{r}_1\boldsymbol{r}_2} \notin \mathcal{V}_{\boldsymbol{r}_1} \cup \mathcal{V}_{\boldsymbol{r}_2}$
   - $\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2} = (\mathcal{V}_{\boldsymbol{r}_1} \cup \mathcal{V}_{\boldsymbol{r}_2} \cup \{S_{\boldsymbol{r}_1|\boldsymbol{r}_2}\}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}_1} \cup \mathcal{R}_{\boldsymbol{r}_2} \cup \{S_{\boldsymbol{r}_1|\boldsymbol{r}_2} \to S_{\boldsymbol{r}_1} \mid S_{\boldsymbol{r}_2}\}, S_{\boldsymbol{r}_1|\boldsymbol{r}_2}) : S_{\boldsymbol{r}_1|\boldsymbol{r}_2} \notin \mathcal{V}_{\boldsymbol{r}_1} \cup \mathcal{V}_{\boldsymbol{r}_2}$
   - $\mathfrak{G}_{(?(\mathrm{DEFINE})({}_l?{<}N{>}\boldsymbol{r}_1)_l)} = (\mathcal{V}_{\boldsymbol{r}_1} \cup \{S_{(?(\mathrm{DEFINE})({}_l?{<}N{>}\boldsymbol{r}_1)_l)}, N_l\}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}_1} \cup \{N_l \to S_{\boldsymbol{r}_1}, S_{(?(\mathrm{DEFINE})({}_l?{<}N{>}\boldsymbol{r}_1)_l)} \to \varepsilon\}, S_{(?(\mathrm{DEFINE})({}_l?{<}N{>}\boldsymbol{r}_1)_l)}) : S_{(?(\mathrm{DEFINE})({}_l?{<}N{>}\boldsymbol{r}_1)_l)}, N_l \notin \mathcal{V}_{\boldsymbol{r}_1}$
3. having grammar $\mathfrak{G}_{\boldsymbol{r}} = (\mathcal{V}_{\boldsymbol{r}}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}}, S_{\boldsymbol{r}})$, for every nonterminal $N \in \mathcal{X} \cap \mathcal{V}_{\boldsymbol{r}}$: if $N_{\nu(N)} \in \mathcal{V}_{\boldsymbol{r}}$ then replace all occurrences of N in the right-hand sides of productions $\mathcal{R}_{\boldsymbol{r}}$ with $N_{\nu(N)}$
4. return $\mathfrak{G}_{\boldsymbol{r}}$

Lemma 12. *Let* $\boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$. *Let* $\mathfrak{G}_{\boldsymbol{r}_1}, \mathfrak{G}_{\boldsymbol{r}_2}$ *be the grammars constructed by the algorithm 2 from* $\boldsymbol{r}_1, \boldsymbol{r}_2$, *respectively. If* $\mathrm{L}(\boldsymbol{r}_1) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1})$ *and* $\mathrm{L}(\boldsymbol{r}_2) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_2})$, *then* $\mathrm{L}(\boldsymbol{r}_1 \mid \boldsymbol{r}_2) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2})$ *and* $\mathrm{L}(\boldsymbol{r}_1\boldsymbol{r}_2) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1\boldsymbol{r}_2})$.

Proof. For the alternative of regexes $\boldsymbol{r}_1, \boldsymbol{r}_2$, following the matching relation (3), $\boldsymbol{r}_1 \mid \boldsymbol{r}_2$ matches some $\boldsymbol{x}$ if at least one of $\boldsymbol{r}_1, \boldsymbol{r}_2$ matches $\boldsymbol{x}$. Clearly, $\boldsymbol{x} \in \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2})$ if $\boldsymbol{x} \in \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1}) \vee \boldsymbol{x} \in \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_2})$. Following the well-known construction of a context-free grammar for the union of languages[21], $\mathrm{L}(\boldsymbol{r}_1 \mid \boldsymbol{r}_2) \subseteq \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2})$ because $\mathcal{V}_{\boldsymbol{r}_1} \cap \mathcal{V}_{\boldsymbol{r}_2}$ can be nonempty. Suppose, for contradiction, that $\boldsymbol{x} \notin \mathrm{L}(\boldsymbol{r}_1 \mid \boldsymbol{r}_2) \wedge \boldsymbol{x} \in \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2})$ while

$\mathrm{L}(\boldsymbol{r}_1) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1})$ and $\mathrm{L}(\boldsymbol{r}_2) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_2})$. The only way it can happen is $S_{\boldsymbol{r}_1|\boldsymbol{r}_2} \underset{\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2}}{\Longrightarrow}{}^{+}$ $\boldsymbol{p}_1 N_1 \boldsymbol{s}_1 \underset{\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2}}{\Longrightarrow} \boldsymbol{p}_1 \boldsymbol{v}_1 \boldsymbol{s}_1 \underset{\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2}}{\Longrightarrow}{}^{*} \boldsymbol{p}_2 N_2 \boldsymbol{s}_2 \underset{\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2}}{\Longrightarrow} \boldsymbol{p}_2 \boldsymbol{v}_2 \boldsymbol{s}_2 \underset{\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2}}{\Longrightarrow}{}^{*} \boldsymbol{x}$ where $N_1 \to \boldsymbol{v}_1$ and $N_2 \to \boldsymbol{v}_2$ are not from the same grammar $\mathfrak{G}_{\boldsymbol{r}_1}, \mathfrak{G}_{\boldsymbol{r}_2}$; in other words, $N_2 \in \mathcal{V}_{\boldsymbol{r}_1} \cap \mathcal{V}_{\boldsymbol{r}_2} \wedge N_2 \to \boldsymbol{v}_2 \notin \mathcal{R}_{\boldsymbol{r}_1} \cap \mathcal{R}_{\boldsymbol{r}_2}$. All grammars from step 2 introduce unique nonterminals that cannot appear in both $\mathcal{V}_{\boldsymbol{r}_1}$ and $\mathcal{V}_{\boldsymbol{r}_2}$. The nonterminals of grammars $\mathfrak{G}_\emptyset$, $\mathfrak{G}_\varepsilon$, and $\mathfrak{G}_a$ clearly cannot have different right-hand sides of productions in different grammars $\mathfrak{G}_{\boldsymbol{r}_1}, \mathfrak{G}_{\boldsymbol{r}_2}$. Both nonterminals introduced by $\mathfrak{G}_{(?\mathrm{P}>N)}$ can appear in both $\mathcal{V}_{\boldsymbol{r}_1}$ and $\mathcal{V}_{\boldsymbol{r}_2}$, however, due to step 3, every N from $\mathfrak{G}_{(?\mathrm{P}>N)}$ is replaced by a single N_l that rewrites to a unique nonterminal determined by a single DEFINE rule. Thus, it is not possible to achieve $N_2 \in \mathcal{V}_{\boldsymbol{r}_1} \cap \mathcal{V}_{\boldsymbol{r}_2} \wedge N_2 \to \boldsymbol{v}_2 \notin \mathcal{R}_{\boldsymbol{r}_1} \cap \mathcal{R}_{\boldsymbol{r}_2}$.

Similar arguments can be used to prove that $\mathrm{L}(\boldsymbol{r}_1\boldsymbol{r}_2) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1\boldsymbol{r}_2})$. □

Lemma 13. *If $\mathfrak{G}_{\boldsymbol{r}} = (\mathcal{V}_{\boldsymbol{r}}, \mathcal{A}, \mathcal{R}_{\boldsymbol{r}}, S_{\boldsymbol{r}})$ is constructed from $\boldsymbol{r}$ by algorithm 2 then $\mathrm{L}(\boldsymbol{r}) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}})$ for any $\boldsymbol{r} \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ (with operations concatenation, alternative, DEFINE rule with named parenthesised expression, and named subroutine call) and $\mathfrak{G}_{\boldsymbol{r}}$ is context-free.*

Proof. The grammars are clearly correct for the cases of elementary expressions $\emptyset, \varepsilon, a \in \mathcal{A}$. Assume that for $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, $\mathrm{L}(\boldsymbol{r}_1) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_1})$ and $\mathrm{L}(\boldsymbol{r}_2) = \mathrm{L}(\mathfrak{G}_{\boldsymbol{r}_2})$, respectively. For a subroutine call $(?\mathrm{P}{>}N)$, the matching relation (7) is

$$\frac{\frac{\frac{(\boldsymbol{r}_1, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}{(\sigma(l), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}}{(\sigma(\nu(N)), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}}{((?\mathrm{P}{>}N), \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}}$$

where $\mathcal{R}$ contains all i' such that $\boldsymbol{x}[i..i'-1]$ matches $\boldsymbol{r}_1$ and l identifies the leftmost parenthesised expression named N. Production $S_{(?\mathrm{P}>N)} \to N$ of $\mathfrak{G}_{(?\mathrm{P}>N)}$ is effectively $S_{(?\mathrm{P}>N)} \to N_{\nu(N)}$ due to step 3. Nonterminal $N_{\nu(N)}$ rewrites to $S_{\boldsymbol{r}_1}$. Therefore, the grammar $\mathfrak{G}_{(?\mathrm{P}>N)}$ generates the same language as is matched by $(?\mathrm{P}{>}N)$.

The grammar $\mathfrak{G}_{(?(\mathrm{DEFINE})(_l?<N>\boldsymbol{r}_1)_l)}$ follows the matching relation for the DEFINE rule (8). The correctness of both $\mathfrak{G}_{\boldsymbol{r}_1|\boldsymbol{r}_2}$ and $\mathfrak{G}_{\boldsymbol{r}_1\boldsymbol{r}_2}$ follows from lemma 12. Therefore, step 2 constructs the correct grammars.

All grammars add only productions with a single nonterminal on the left-hand side; therefore, all grammars constructed by the algorithm 2 are context-free. □

Lemma 2 is proved by lemmas 5 and 6; also, lemma 7 is proved by lemmas 8, 9, 10, 11, and 13. Therefore, theorem 1 is proved.

5 Expressive power of subroutine call combined with lookaround assertions

We show that lookaround assertion combined with subroutine call has greater expressive power than subroutine call alone. We use an example of such regex inspired by Popov's blog post[19] and arguments by Scheinberg[21].

Theorem 14. $\mathbb{L}_{\mathrm{E_S}} \subsetneq \mathbb{L}_{\mathrm{E_{LS}}}$

Proof. We show a regex $\boldsymbol{r} \in \mathbb{E}_{\mathrm{LS},\mathcal{A},\mathcal{X}}$ that matches a language that is not context-free (the equality of $\mathbb{L}_{\mathrm{CF}}$ and $\mathbb{L}_{\mathrm{E_S}}$ is shown by theorem 1). Language $\mathcal{L} = \{\mathtt{a}^g\mathtt{b}^g\mathtt{c}^g : g \in \mathbb{N}\}$ is a well-known language that is not context-free[11, example 7.19]. We show that for $\boldsymbol{r} = (?{=}(?{<}N_1{>}\mathtt{a}(\varepsilon \mid (?\mathrm{P}{>}N_1))\mathtt{b})\mathtt{c})\mathtt{a}\mathtt{a}^*(?{<}N_2{>}\mathtt{b}(\varepsilon \mid (?\mathrm{P}{>}N_2))\mathtt{c})$, $\mathcal{L} = \mathrm{L}(\boldsymbol{r})$. Let $\boldsymbol{r}_1 = (?{<}N_1{>}\mathtt{a}(\varepsilon \mid (?\mathrm{P}{>}N_1))\mathtt{b})\mathtt{c}, \boldsymbol{r}_2 = \mathtt{a}\mathtt{a}^*, \boldsymbol{r}_{N_2} = \mathtt{b}(\varepsilon \mid (?\mathrm{P}{>}N_2))\mathtt{c}$, and thus $\boldsymbol{r} = (?{=}\boldsymbol{r}_1)\boldsymbol{r}_2(?{<}N_2{>}\boldsymbol{r}_{N_2})$. Let $\boldsymbol{r}_{N_1} = \mathtt{a}(\varepsilon \mid (?\mathrm{P}{>}N_1))\mathtt{b}$. The matching relation for $\boldsymbol{r}_1$ can be derived as follows:

$$\dfrac{\dfrac{\dfrac{\dots}{(\boldsymbol{r}_{N_1}, \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R}}}{((?{<}N_1{>}\boldsymbol{r}_{N_1}), \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R}} \quad \dfrac{\boldsymbol{x}[i] = \mathtt{c}}{\forall i \in \mathcal{R} : (\mathtt{c}, \boldsymbol{x}, i) \rightsquigarrow \{i+1 : \boldsymbol{x}[i] = \mathtt{c}\}}}{((?{<}N_1{>}\boldsymbol{r}_{N_1})\mathtt{c}, \boldsymbol{x}, 1) \rightsquigarrow \bigcup_{i\in\mathcal{R}}\{i+1 : \boldsymbol{x}[i] = \mathtt{c}\}}$$

Therefore, $\mathrm{L}(\boldsymbol{r}_1) = \mathrm{L}(\boldsymbol{r}_{N_1}) \cdot \{\mathtt{c}\}$. Let us derive the matching relation for $\boldsymbol{r}_{N_1}$:

$$\dfrac{\boldsymbol{x}[1] = \mathtt{a}}{(\mathtt{a}, \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R}_{\mathtt{a}}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dots}{(\boldsymbol{r}_{N_1}, \boldsymbol{x}, 2) \rightsquigarrow \mathcal{R}_{N_1\prime}}}{(\varepsilon, \boldsymbol{x}, 2) \rightsquigarrow \{2\}} \quad \dfrac{}{((?\mathrm{P}{>}N_1), \boldsymbol{x}, 2) \rightsquigarrow \mathcal{R}_{N_1\prime}}}{(\varepsilon \mid (?\mathrm{P}{>}N_1), \boldsymbol{x}, 2) \rightsquigarrow \mathcal{R}_{N_1} = \{2\} \cup \mathcal{R}_{N_1\prime}} \quad \dfrac{\boldsymbol{x}[i] = \mathtt{b}}{\forall i \in \mathcal{R}_{N_1} : (\mathtt{b}, \boldsymbol{x}, i) \rightsquigarrow \mathcal{R}_{\mathtt{b}_i}}}{((\varepsilon \mid (?\mathrm{P}{>}N_1)), \boldsymbol{x}, 2) \rightsquigarrow \mathcal{R}_{N_1} \qquad \text{if } \boldsymbol{x}[1] = \mathtt{a} : ((\varepsilon \mid (?\mathrm{P}{>}N_1))\mathtt{b}, \boldsymbol{x}, 2) \rightsquigarrow \mathcal{R} = \bigcup_{i\in\mathcal{R}_{N_1}} \mathcal{R}_{\mathtt{b}_i}}$$

$$(\mathtt{a}(\varepsilon \mid (?\mathrm{P}{>}N_1))\mathtt{b}, \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R} \text{ if } \boldsymbol{x}[1] = \mathtt{a}$$

The regex $\boldsymbol{r}_{N_1}$ matches only if a prefix of $\boldsymbol{x}$ has the form $\mathtt{a}\dots\mathtt{a}\mathtt{b}\dots\mathtt{b}$. Furthermore, because as and bs are matched only within the same parenthesised expression and the same number of times, $\mathrm{L}(\boldsymbol{r}_{N_1}) = \{\mathtt{a}^g\mathtt{b}^g : g \in \mathbb{N}\}$. Clearly, $\mathrm{L}(\boldsymbol{r}_2) = \{\mathtt{a}\}^+$. Following similar arguments as for $\mathrm{L}(\boldsymbol{r}_{N_1})$, $\mathrm{L}(\boldsymbol{r}_{N_2}) = \{\mathtt{b}^g\mathtt{c}^g : g \in \mathbb{N}\}$. The matching relation for $\boldsymbol{r}$ can be derived as follows:

$$\dfrac{\dfrac{\dfrac{\dots}{(\boldsymbol{r}_1, \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R}_1}}{((?{=}\boldsymbol{r}_1), \boldsymbol{x}, 1) \rightsquigarrow \{1 : \mathcal{R}_1 \neq \emptyset\}} \quad \dfrac{\dots}{\text{if } \mathcal{R}_1 \neq \emptyset : (\boldsymbol{r}_2(?{<}N_2{>}\boldsymbol{r}_{N_2}), \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R}}}{((?{=}(?{<}N_1{>}\mathtt{a}(\varepsilon \mid (?\mathrm{P}{>}N_1))\mathtt{b})\mathtt{c})\mathtt{a}\mathtt{a}^*(?{<}N_2{>}\mathtt{b}(\varepsilon \mid (?\mathrm{P}{>}N_2))\mathtt{c}), \boldsymbol{x}, 1) \rightsquigarrow \mathcal{R}}$$

The lookahead matches, following the matching relation (5), only if the regex $\boldsymbol{r}_1$ matches, while the current position in $\boldsymbol{x}$ is unchanged. Thus, $\boldsymbol{r}$ matches $\boldsymbol{x}$ if $\boldsymbol{x}$ starts with $\mathtt{a}^g\mathtt{b}^g\mathtt{c}$ and also has the form $\mathtt{a}^{g'}\mathtt{b}^g\mathtt{c}^g$. In other words, $\mathrm{L}(\boldsymbol{r}) = (\{\mathtt{a}^g\mathtt{b}^g\mathtt{c} : g \in \mathbb{N}\} \cdot \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*) \cap (\{\mathtt{a}^g : g \in \mathbb{N}\} \cdot \{\mathtt{b}^g\mathtt{c}^g : g \in \mathbb{N}\})$. □

5.1 Relation with context-sensitive languages

To the author's knowledge, there is no peer-reviewed or academic publication concerning the expressive power of practical regular expressions with both lookaround assertions and subroutine calls. The only known text on this topic is due to Popov[19]: an idea of what a reduction of context-sensitive grammars to regexes might look like.

Popov claims that having a context-sensitive grammar $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$, any production in the form $\boldsymbol{p}N\boldsymbol{s} \to \boldsymbol{p}\boldsymbol{v}\boldsymbol{s}$ can be converted into a DEFINE rule of the form $(?(\mathrm{DEFINE})(?{<}N{>}(?{<=}\,\mathrm{rx}(\boldsymbol{p}))\,\mathrm{rx}(\boldsymbol{v})(?{=}\,\mathrm{rx}(\boldsymbol{s}))))$. However, this alone does not work for all context-sensitive grammars. Let us attempt to formalize the conversion in algorithm 3 as a modification of algorithm 1:

Although a regex $\boldsymbol{r}$ matches any string generated by grammar $\mathfrak{G}$, it is still not correct because, in general, it can match more. To apply a production of the form

Algorithm 3 Conversion of a context-sensitive grammar to a regex ([19], incorrect)

Input: a context-sensitive grammar $\mathfrak{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$
Output: a regex $\boldsymbol{r} \in \mathbb{E}_{\mathrm{S},\mathcal{A},\mathcal{X}}$ such that $\mathrm{L}(\mathfrak{G}) = \mathrm{L}(\boldsymbol{r})$

1. initialize $\boldsymbol{r} = \varepsilon$ and consider $\mathcal{X} = \mathcal{V}$
2. for all $N \in \mathcal{V}$:
   (a) let $\boldsymbol{r}_N = \varepsilon$
   (b) for all productions with N on the left-hand side and particular left and right context ($\boldsymbol{p}N\boldsymbol{s} \to \boldsymbol{p}\boldsymbol{v}_{N1}\boldsymbol{s} \mid \cdots \mid \boldsymbol{p}\boldsymbol{v}_{Nm_N}\boldsymbol{s} \in \mathcal{R}$):
      i. let $\boldsymbol{r}_{Ng}$ $(1 \leq g \leq m_N)$ be constructed from $\boldsymbol{v}_{Ng}$ for $\boldsymbol{p}N\boldsymbol{s}$ ($\boldsymbol{v}_{Ng}$): $\boldsymbol{r}_{Ng} = \mathrm{rx}(\boldsymbol{v}_{Ng})$
      ii. if $\boldsymbol{r}_N = \varepsilon$ then $\boldsymbol{r}_N = (?<=\boldsymbol{p})\boldsymbol{r}_{Ng}(?=\boldsymbol{s})$ else $\boldsymbol{r}_N = \boldsymbol{r}_N \mid (?<=\boldsymbol{p})\boldsymbol{r}_{Ng}(?=\boldsymbol{s})$
   (c) let $\boldsymbol{r} = \boldsymbol{r}(?(\mathrm{DEFINE})(?<N>\boldsymbol{r}_N))$
3. add the matching of the initial symbol, i.e., let $\boldsymbol{r} = \boldsymbol{r}((?\mathrm{P}>S))$

$\boldsymbol{p}N\boldsymbol{s} \to \boldsymbol{p}\boldsymbol{v}\boldsymbol{s}$ in the generation of string $\boldsymbol{x}$, the left-hand side of the production must appear in a sentential form, that is, $S \underset{\mathfrak{G}}{\Rightarrow}^* \boldsymbol{p}'\boldsymbol{p}N\boldsymbol{s}\boldsymbol{s}' \underset{\mathfrak{G}}{\Rightarrow} \boldsymbol{p}'\boldsymbol{p}\boldsymbol{v}\boldsymbol{s}\boldsymbol{s}' \underset{\mathfrak{G}}{\Rightarrow}^* \boldsymbol{x}$. In other words, the context $(\boldsymbol{p}, \boldsymbol{s})$ of the production must already be present in the sentential form. Similarly to $\boldsymbol{p}N\boldsymbol{s} \to \boldsymbol{p}\boldsymbol{v}\boldsymbol{s}$ being part of generating a substring of $\boldsymbol{x}$, the subpattern $(?<=\boldsymbol{p})\boldsymbol{v}(?=\boldsymbol{s})$ matches a substring of $\boldsymbol{x}$. However, $(?<=\boldsymbol{p})\boldsymbol{v}(?=\boldsymbol{s})$ can match a substring of another $\boldsymbol{x}' \notin \mathrm{L}(\mathfrak{G})$ because during the matching of a regex, the order of the use of subpatterns is independent of the derivations of sentential forms by $\mathfrak{G}$. The following example illustrates this.

Example 15. Let us have $\mathfrak{G} = (\{N_1, N_2, S\}, \{\mathtt{a}, \mathtt{c}\}, \{S \to N_2N_1N_1, N_1 \to \mathtt{a}, \mathtt{a}N_1 \to \mathtt{aaa}, N_2\mathtt{a} \to \mathtt{caa}\}, S)$. Clearly, $\mathrm{L}(\mathfrak{G}) = \{\mathtt{caaa}, \mathtt{caaaa}\}$. After applying algorithm 3,

$$\boldsymbol{r} = (?(\mathrm{DEFINE})(?<N_1>\mathtt{a} \mid (?<=\mathtt{a})\mathtt{aa}))(?(\mathrm{DEFINE})(?<N_2>\mathtt{ca}(?=\mathtt{a}))) \cdot \\ \cdot(?(\mathrm{DEFINE})(?<S>(?\mathrm{P}>N_2)(?\mathrm{P}>N_1)(?\mathrm{P}>N_1)))(?\mathrm{P}>S)$$

and $\mathrm{L}(\mathfrak{G}) \subsetneq \mathrm{L}(\boldsymbol{r})$, as $\mathtt{caaaaa} \in \mathrm{L}(\boldsymbol{r})$: $\mathtt{caaaaa}[1..2]$ is matched by the subpattern $\mathtt{ca}(?=\mathtt{a})$, and both $\mathtt{caaaaa}[3..4]$ and $\mathtt{caaaaa}[5..6]$ are matched by the subpattern $(?<=\mathtt{a})\mathtt{aa}$. However, in $\mathfrak{G}$, there is no way to apply production $\mathtt{a}N_1 \to \mathtt{aaa}$ twice, as there must first be symbol $\mathtt{a}$ in a sentential form (which consumes one N_1).

As a result, the relation between the class of context-sensitive languages and the class of languages expressed by regexes with both subroutine calls and lookaround assertions remains an open problem.

6 Conclusions

We presented a formalisation of syntax and semantics of certain features of practical regular expressions using the matching relation: subroutine call, named parenthesised expression, and DEFINE rule. We attempted to mimic documented (and real) behaviour of certain flavours of practical regular expressions: Perl-compatible regular expressions, Perl, and Ruby Regexp class.

This paper showed the equivalence of context-free languages and languages expressed by practical regular expressions with concatenation, alternative, and subroutine call. This result applies to flavours that support subroutine calls. We presented an alternative constructive proof employing named subroutine calls, DEFINE rules, and the matching relation: a conversion between such practical regular expressions and context-free grammar.

We showed that adding zero-width lookaround assertions to practical regular expressions with operations concatenation, alternative, and subroutine call extends their expressive power beyond context-free languages. However, the relation of the language class expressed by such expressions to some non-context-free languages, particularly the class of context-sensitive languages, remains an open problem.

We hope that our results stimulate more work on the expressive power of specific combinations of operations used in practical regular expressions, such as backreferences, subroutine calls, lookaround assertions, or atomic groups.

References

1. A. V. Aho: *Algorithms for Finding Patterns in Strings*, in Algorithms and Complexity, J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier, 1990, pp. 255–300.
2. M. Berglund and B. van der Merwe: *Regular Expressions with Backreferences Re-examined*, in Proceedings of the Prague Stringology Conference 2017, Czech Technical University in Prague, 2017, pp. 30–41.
3. M. Berglund, B. Van Der Merwe, and S. van Litsenborgh: *Regular Expressions with Lookahead*. Journal of Universal Computer Science, 27(4) 2021, pp. 324–340.
4. C. Câmpeanu, K. Salomaa, and S. Yu: *A formal study of practical regular expressions*. International Journal of Foundations of Computer Science, 14(06) Dec. 2003, pp. 1007–1018.
5. N. Chida and T. Terauchi: *On lookaheads in regular expressions with backreferences*, in 7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022), A. P. Felty, ed., vol. 228 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2022, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 15:1–15:18.
6. N. Chida and T. Terauchi: *Repairing DoS vulnerability of real-world regexes*, in 2022 IEEE Symposium on Security and Privacy (SP), Los Alamitos, CA, USA, May 2022, IEEE Computer Society, pp. 1049–1066.
7. N. Chomsky: *Three models for the description of language*. IEEE Transactions on Information Theory, 2(3) Sept. 1956, pp. 113–124.
8. J. E. F. Friedl: *Mastering Regular Expressions*, O'Reilly, Sebastapol, CA, 3rd ed., 2006.
9. S. A. Greibach: *A New Normal-Form Theorem for Context-Free Phrase Structure Grammars*. Journal of the ACM, 12(1) Jan. 1965, pp. 42–52.
10. P. Hazel: *PCRE2 — Perl-compatible regular expressions (revised API)*, University of Cambridge, Cambridge CB2 3QH, England, 2022, UNIX manual page PCRE2PATTERN(3) referring to PCRE2 10.40.
11. J. E. Hopcroft, R. Motwani, and J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation*, Always Learning / Pearson, Pearson Education, Harlow, 3rd ed., 2014.
12. V. Hruša: *Regular Expressions with Subpattern Recursion*, Master's thesis, Czech Technical University in Prague, 2021.
13. S. C. Kleene: *Representation of Events in Nerve Nets and Finite Automata*, in Automata Studies. (AM-34), C. E. Shannon and J. McCarthy, eds., Princeton University Press, Dec. 1956, pp. 3–42.
14. D. C. Kozen: *Automata and Computability*, Undergraduate Texts in Computer Science, Springer, New York, Oct. 2012.
15. A. Mateescu and A. Salomaa: *Aspects of Classical Language Theory*, in Handbook of Formal Languages, G. Rozenberg and A. Salomaa, eds., Springer Berlin Heidelberg, 1997, pp. 175–251.
16. B. Melichar and J. Holub: *6D Classification of Pattern Matching Problems*, in Proceedings of the Prague Stringology Club Workshop '97, Czech Technical University in Prague, 1997.
17. A. Morihata: *Translation of regular expression with lookahead into finite state automaton*. Computer Software, 12(1) 2012, pp. 148–158.
18. *perlre — Perl regular expressions*, Perl Programmers Reference Guide, 2022, UNIX manual page PERLRE(1) referring to perl v5.36.0.
19. N. Popov: *The true power of regular expressions*. https://www.npopov.com/2012/06/15/The-true-power-of-regular-expressions.html, June 2012.

20. *class Regexp*, https://docs.ruby-lang.org/en/3.1/Regexp.html, The API documentation for Ruby 3.1.
21. S. Scheinberg: *Note on the boolean properties of context free languages.* Information and Control, 3(4) Dec. 1960, pp. 372–375.
22. M. L. Schmid: *Characterising REGEX languages by regular languages equipped with factor-referencing.* Information and Computation, 249 Oct. 2016.
23. K. Thompson: *Programming Techniques: Regular expression search algorithm.* Communications of the ACM, 11(6) June 1968, pp. 419–422.