Computing the Shortest String and the Edit-Distance for Parsing Expression Languages

Hyunjoon Cheon and Yo-Sub Han(✉)

Department of Computer Science, Yonsei University, 50, Yonsei-ro, Seodaemun-gu, Seoul 03722, Republic of Korea
{hyunjooncheon,emmous}@yonsei.ac.kr

Abstract. A distance between two languages is a useful tool to measure the language similarity, and is closely related to the intersection problem as well as the shortest string problem. A parsing expression grammar (PEG) is an unambiguous grammar such that the choice operator selects the first matching in PEG while it can be ambiguous in a context-free grammar. PEGs are also closely related to top-down parsing languages. We consider two problems on parsing expression languages (PELs). One is the r-shortest string problem that decides whether or not a given PEL contains a string of length shorter than r. The other problem is the edit-distance problem of PELs with respect to other language families such as finite languages or regular languages. We show that the r-shortest string problem and the edit-distance problem with respect to finite languages are NEXPTIME-complete, and the edit-distance problem with respect to regular languages is undecidable. In addition, we prove that it is impossible to compute a length bound $\mathcal{B}(G)$ of a PEG G such that $L(G)$ has a string w of length at most $\mathcal{B}(G)$.

Keywords: Formal languages · Parsing expression grammars · Edit-distance

1 Introduction

Perl-compatible regular expressions (PCREs) are popular tools for information retrieval and data processing. From a formal language viewpoint, the expressive power of PCREs is interesting. A simple PCRE can define a context-sensitive language. For example, the PCRE `(a*)b\1b\1` represents a context-sensitive language $a^nba^nba^n$, which is not context-free. This implies that there are no simple matching algorithms for PCREs.

Ford [4] proposed parsing expression grammars as a recognition-based formal grammar that is a generalization of TMG recognition schema [2]. Parsing expression grammars (PEGs) are intuitive for pattern matching and have a simpler and efficient algorithm for matching than the algorithms for PCRE. Unlike other grammars such as regular expressions or context-free grammars (CFGs)

N. Jonoska and D. Savchuk (Eds.): DLT 2020, LNCS 12086, pp. 43–54, 2020.
https://doi.org/10.1007/978-3-030-48516-0_4

that match the input string if the grammars generate the whole input string, PEGs regard the string matching if PEGs recognize a prefix (not necessarily the whole string) of the input string deterministically. Thus, a PEG itself is unambiguous—each rule in the grammar has a strict order and the grammar tries to match the input according to its rule orders. This makes PEGs useful for parsing a string since it has a lookahead that can verify whether or not a prefix of the remaining input matches the given expression. Based on this property, the Packrat parsing algorithm [3] for PEG recognizes a prefix of its input string in polynomial time in the input size and the grammar size. This efficient matching algorithm makes PEGs to be alternatives of PCREs. IBM recently proposed Rosie pattern language (RPL) [1] based on PEG for pattern matching.

Medeiros et al. [11] designed a PEG construction from a PCRE and evaluated the pattern matching performance between PEGs and PCREs. Loff et al. [10] discussed a few computational aspects of PEGs. They showed that a PEG accepts a pair of an input and its output for any computable functions. They also proposed a new computational machine, scaffolding automata (SAs), that operates using a set of states and an auxiliary DAG structure, and proved the equivalence between SAs and the reversal of parsing expression languages (PELs). Koga [7] examined the context-freeness of a PEL, and showed that it is undecidable whether or not a PEL L belongs to a subfamily of context-free languages (CFLs).

For the edit-distance problems of languages, Mohri [12] showed that the edit-distance between two regular languages can be solved in polynomial-time while the same problem between two CFLs is undecidable. Konstantinidis [8] suggested an algorithm for computing the edit-distance of a given finite automaton (FA) and obtained an upper bound of the distance. Povarov [16] studied the neighborhood language according to the Hamming distance [5] that counts the number of different symbols between two strings of the same length. An r-neighborhood L_r of a language L according to a distance metric d is a set of strings whose distance from a string in L is at most r. From an FA with n states, we can construct an NFA with $n(r+1)$ states for r-Hamming-neighborhood language of $L(A)$. Under the similar construction, we can also construct an NFA for r-edit-distance neighborhood with the same bound. Han et al. [6] considered the edit-distance problem between a regular language and a CFL. They presented a construction that accepts an alignment between a pair of strings from each language, and designed an algorithm that computes the edit-distance between a regular language and a CFL in polynomial time based on the construction. Ng et al. [13] studied the edit-distance neighborhood of a regular languages. They showed that, for an n state FA A, there exists a DFA with at most $(r+2)^n - 1$ states that accepts an r-edit-distance neighborhood of $L(A)$.

We consider two decision problems on PEGs: the r-shortest string problem and the r-edit-distance problem. The r-shortest string problem determines whether or not a given PEL has a string whose length is at most length $r \geq 0$ string. The r-edit-distance problem decides whether or not the edit-distance between one PEL and another language is at most $r \geq 0$. We show that the r-shortest string problem and the edit-distance problem with respect to finite

languages are NEXPTIME-complete. Moreover, we demonstrate that the r-edit-distance problem with respect to regular languages is undecidable. In addition, we prove that it is impossible to compute a length bound $\mathcal{B}(G)$ of a PEG G such that $L(G)$ has a string of length at most $\mathcal{B}(G)$.

2 Preliminaries

A Turing machine (TM) M is specified by a tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where Q is a set of states, Σ is an input alphabet, Γ is a tape alphabet, $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ is a set of transitions, and three states $q_0, q_a, q_r \in Q$ are an initial, an accepting and a rejecting state, respectively. The blank tape symbol is denoted by $B \in \Gamma$.

A TM M accepts (rejects) a string $w \in \Sigma^*$ if, starting from the initial state q_0, M has a transition path that reaches an accepting state q_a (a rejecting state q_r, respectively) by following δ. We define the language $L(M)$ of M to be

$$L(M) = \{w \mid M \text{ accepts } w\}.$$

We assume that M halts on w if M reaches either q_a or q_r on w.

A configuration C of M is a string over $Q \cup \Gamma$, where C consists of one state symbol and tape symbols. The state symbol represents both the current state (by the symbol itself) and the head position (by its position). The initial configuration with the input string w is always $q_0 w$ and the head is at the first symbol of w. We say a configuration C is accepting (rejecting) if C is on the state q_a (q_r, respectively).

2.1 Parsing Expression Grammars

A parsing expression is an expression generated from the following grammars.

$$e \to \lambda \mid a \in \Sigma \mid A \in V \mid (e) \mid ee \mid e/e \mid \&e \mid !e,$$

where λ denotes the empty string, Σ is an alphabet and V is a set of variables. Given a parsing expression e and a string $w = xy$, we say that $(e, w) \to y$ if e recognizes a prefix x of w and $(e, w) \to f$ if e fails to recognize w. A PEG G is specified by a tuple (V, Σ, R, S), where V is a set of variables, Σ is an input alphabet, R is a set of rules and $S \in V$ is the starting variable. A rule $(A, e) \in R$ is a pair of a variable and a parsing expression—we sometimes denote it by $A \leftarrow e$. PEGs look similar to CFGs yet the main difference is that PEGs do not have the union operation that can nondeterministically choose either of productions. On the other hand, PEGs have a choice operator ($/$) (e.g., A/B) that first tries to match its left side rule (A) and, if it fails, tries its right side rule (B) in order. This choice operator makes a PEG unambiguous. The expressions of the forms $\&e$ or $!e$ are called *and-* and *not-predicate* that recognizes the empty string λ if e recognizes (cannot recognize, respectively) the input string. In other words, these expressions work as unbounded lookahead.

Given a PEG $G = (V, \Sigma, R, S)$, we define the PEL $L(G)$ to be

$$L(G) = \{xy \mid (S, xy) \to y, y \in \Sigma^*\}.$$

This definition allows to contain any strings whose prefix is recognized by the grammar.

Example 1. The following PEG G over $\Sigma = \{a, b, c\}$ with a single rule $S \leftarrow \&a/!(bc)$ recognizes the language $\{(a + ba + bb + c)\Sigma^*\}$ by the following steps:

- The strings in $\{a\Sigma^*\}$ (e.g., abc) match the and-predicate. We do not try matching the second expression.
- The strings in $\{bc\Sigma^*\}$ (e.g., bca) *do not match* both expressions so G fails to recognize. (The second expression $!(bc)$ fails to match since bc matches those strings.)
- Other strings in $\{(ba + bb + c)\Sigma^*\}$ do not match the and-predicate but match the not-predicate.

Example 1 illustrates the role of & and ! in PEGs. Table 1 shows a few more PEG examples and languages (Σ matches any symbol).

Table 1. Example PEGs and their languages

PEG	Language
$S \leftarrow a$	$\{a\Sigma^*\}$
$S \leftarrow a/ab$	$\{a\Sigma^*\}$
$S \leftarrow \&(ab)$	$\{ab\Sigma^*\}$
$S \leftarrow a!\Sigma$	$\{a\}$

It is known that the emptiness, the universality of a PEL and the equivalence between two PELs are all undecidable [4]. It follows that the intersection emptiness between a regular language and a PEL is also undecidable: if the regular language is Σ^*, their intersection emptiness shows that the PEL is empty. Furthermore, PEGs have a linear-time parsing algorithm that uses Packrat parsing method. If the grammar G, however, is not fixed, the membership test $w \in L(G)$ can be done in $O(|G| \cdot |w|)$ time [3].

Note that PEGs may seem similar to conjunctive grammars [14]. However, a big difference is that PEGs are always unambiguous whereas conjunctive grammars can be ambiguous [4,15]. Also, in parsing, the conjunction operation in conjunctive grammars must consume the matching substring but the and-predicate in PEGs only verifies the matching string and consumes no symbols.

2.2 Edit-Distance

The edit-distance (or the Levenshtein distance) [9] between two strings x and y, denoted by $d(x, y)$, is the minimum number of edit operations—insertion, deletion and substitution—that transform x into y, where

- insertion adds a symbol into x
- deletion removes a symbol from x and
- substitution replaces a symbol from x with another symbol.

Then, we define the edit-distance between a string and a language, and between two languages as follows:

$$d(w, L) = \min_{x \in L} d(w, x) \qquad d(L_1, L_2) = \min_{x \in L_1} d(x, L_2).$$

3 The r-Shortest String Problem

The r-shortest string problem is to decide whether or not a language L contains a string of length of at most $r \geq 0$. For CFGs, this problem is straightforward. However, for PEGs, because of the ordered choice and the predicates in PEGs, the problem is not trivial. When recognizing a given input string, we need to check both the predicates and the corresponding rules that actually process the input string. Thus, we need to verify every rule and predicate, and find the common string for every rule. For instance, a PEG G of

$$S \leftarrow \&(10)A, \quad A \leftarrow 1(1/01)$$

cannot recognize the string 11 as the and-predicate does not match 11. Moreover, the r-shortest string problem is a special version of the r-edit-distance problem for two languages (the formal definition is in Definition 2), where one language is $\{\lambda\}$.

Definition 1 (r-shortest string (r-SS) problem). *Given a language L and an integer $r \geq 0$, the r-SS problem on L is to decide whether or not there exists a string $w \in L$ whose length is at most r.*

We prove that the r-SS problem on PELs is NEXPTIME-complete. We start with a simple NEXPTIME algorithm.

Lemma 1. *Given a PEG $G = (V, \Sigma, R, S)$ and an integer $r \geq 0$, The r-SS problem on $L(G)$ is in* NEXPTIME.

Proof. Consider the following algorithm.

1. Nondeterministically choose a string $w \in \Sigma^{\leq r}$,
2. Decide whether or not $w \in L(G)$.

We can guess the string w on the first step of the algorithm in at most r computation steps. The following membership test takes in quadratic time to the grammar size $|G|$ and the input size $|w| \leq r$ [3]. Thus, the entire algorithm is in NEXPTIME. □

Next we show that the r-SS problem on PEL is NEXPTIME-hard using the bounded halting problem.

Theorem 1 (Bounded halting problem [17]). *Given a TM M, an input $w \in \Sigma^*$ and an integer $k \geq 0$, it is* NEXPTIME-complete *to decide whether or not M halts on w in at most k steps.*

Before the main proof, consider the following PEG that generates a string of an exponential length with respect to the grammar size.

Example 2. The following PEG G of $n+1$ variables recognizes 0^{2^n} and, thus, $L(G) = \{0^{2^n}\Sigma^*\}$.

$$\begin{aligned} S &\leftarrow A_{n-1}A_{n-1} \\ A_{n-1} &\leftarrow A_{n-2}A_{n-2} \\ &\vdots \\ A_1 &\leftarrow A_0A_0 \\ A_0 &\leftarrow 0 \end{aligned}$$

By substituting A_0 rule to recognize a set of symbols, say $A_0 \leftarrow \Sigma$, we can design a PEG for recognizing Σ^n using $O(\log n)$ rules.

Now, we are ready to show that the r-SS problem on PELs is NEXPTIME-hard.

Lemma 2. *Given a PEG G and an integer $r \geq 0$, the r-SS problem on $L(G)$ is* NEXPTIME-hard.

Proof. We prove the hardness by a poly-time reduction from the bounded halting problem to the r-SS problem. Given a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, an input w and a nonnegative integer k, if M halts on w in at most k steps, we must have a finite computation of

$$C_1\#C_2\#C_3\#\ldots\#C_n\#,$$

where each C_i is a configuration over $\Sigma \cup Q$ and $1 \leq n \leq k$. Such a halting computation is valid if and only if:

1. every configuration C_i is valid (has only one state and contains only tape symbols),
2. the initial configuration is $C_1 = q_0w$,
3. the final configuration is on either the state q_a or q_r and
4. every computation step must follow the TM transitions δ.

We construct a PEG G that accepts every halting computation

$$C = C'_1\#C'_2\#C'_3\#\ldots\#C'_n\#$$

of M on input w, where $C_1' = B^k q_0 w B^k$ with the blank symbol B. Because C_1' has at least k symbols on both sides of its head, as long as we simulate at most k steps, we can assume that every configuration C_i has the same number $l = 2k + |C_1| = 2k + |w| + 1$ of symbols.

The following is a fragment of PEG G for M over the alphabet $\widetilde{\Sigma} = \Gamma \cup Q \cup \{\#\}$. We omit the polynomial size rules for the strings in the form of A^k, where A is a set of symbols for simplicity. (The construction is similar to Example 2)

$$
\begin{aligned}
S &\leftarrow \&(B^k q_0 w B^k \#) D \\
F &\leftarrow \&(Q_f \#)(\Gamma \cup Q)^l \# ! \widetilde{\Sigma} \\
D &\leftarrow F \\
&\quad / cpa \&(\widetilde{\Sigma}^{l-2} qcb) D && (p, a, q, b, L) \in \delta, c \in \Gamma \\
&\quad / pa \&(\widetilde{\Sigma}^{l-1} bq) D && (p, a, q, b, R) \in \delta \\
&\quad / a \&(\widetilde{\Sigma}^{l} a) D && a \in \Gamma \cup \{\#\} \\
Q_f &\leftarrow (q_a / q_r) \Gamma^* / \Gamma Q_f
\end{aligned}
$$

The constructed PEG G has three main rules: S, F and D. The rule S for the valid condition 2 checks whether or not the initial configuration is exactly $B^k q_0 w B^k$. Since we can design a PEG that accepts the prefix B^k in the size of $O(\log k)$, this rule does not violate the polynomial bound of reduction.

The rule F corresponding to the valid condition 3 determines whether or not a length l configuration is a halting configuration. F checks that the configuration has exactly one state, which is either q_a or q_r, and $l - 1$ tape symbol sequence ending with $\#$.

The rule D corresponding to the valid condition 4 represents the TM transitions. D first checks that the current configuration is a halting configuration by delegating checking to F. If the current configuration is not a halting configuration, it enumerates possible TM transitions between the current configuration and the next configuration.

The valid condition 1 holds because the initial configuration always has exactly one (q_0) state surrounded by $l - 1$ tape symbols and the rule D ensures that, if the previous configuration is valid for rule 1, then the next one is also valid. The rule Q_f decides whether or not the current configuration sequence, not necessarily l-length, has exactly one final state and a sequence of tape symbols.

The constructed grammar G can recognize a halting computation of a $(2k + |w| + 2)n$ length string, where n is the number of computation steps to halt. Thus, the $(r = (2k + |w| + 2)k)$-SS problem is equivalent to deciding the existence of a halting computation of at most k steps. This completes a polynomial time reduction. □

By Lemmas 1 and 2, we establish the following statement.

Theorem 2. *Given a PEG G and an integer $r \geq 0$, the r-SS problem on $L(G)$ is* NEXPTIME-complete.

When r is in unary representation or a fixed constant, we can obtain a better result as follows:

Corollary 1. *When r is given in a unary representation, the r-SS problem is* NP-complete. *If r is a fixed constant, then we can solve the problem in polynomial time.*

Proof. The proof is similar to the proof for Lemma 2. The bounded halting problem is NP-complete when the input r is unary [17]. If we regard r to be a fixed constant, then the algorithm in the proof of Lemma 2 becomes polynomial. □

4 The r-Edit-Distance Problem

Next, we examine the r-edit-distance problem for PELs. This problem is crucial for verifying whether or not a string is in a given error bound from a PEL, or for measuring the similarity between a PEL and another language.

Definition 2 (r-edit-distance problem (r-ED)). *Given a language L, a string w and an integer $r \geq 0$, the r-ED problem between w and L is to decide whether or not $d(w, L) \leq r$.*

As the r-SS problem is a simple version of the r-ED problem, it is immediate that the r-ED problem is "harder" than r-SS problem.

Lemma 3. *Given a language L and an integer $r \geq 0$, the r-SS problem for L is mapping reducible to the r-ED between λ and L.*

Proof. If L has a string of length $n \leq r$, the edit-distance between the empty string λ and the string must be $n \leq r$. On the other hand, if L has no strings of length $n \leq r$, then the edit-distance between the empty string and L must be greater than r. □

Corollary 2. *Given a PEG G, a string w and an integer $r \geq 0$, the r-ED between w and $L(G)$ is* NEXPTIME-hard.

Now, we show that the r-ED problem on PELs is in NEXPTIME.

Lemma 4. *Given a PEG G, a string w and an integer $r \geq 0$, the r-ED between w and $L(G)$ is in* NEXPTIME.

Proof. Similar to the r-SS problem, the following algorithm shows that the r-ED problem is in NEXPTIME.

1. Choose $i \leq r$ nondeterministically.
2. Nondeterministically choose a valid edit operation on w and apply it.
3. Repeat the previous step i times to make the resulting string x has edit-distance of at most r.
4. Decide whether or not $x \in L(G)$.

Note that the length of x is between $\max(0, |w| - r)$ and $|w| + r$, which is exponential to the input size. □

Combining Corollary 2 and Lemma 4, we obtain the following statement.

Theorem 3. *Given a PEG G, a string w and an integer $r \geq 0$, the r-ED between w and $L(G)$ is* NEXPTIME-complete.

We can also solve the r-ED problem between a PEL and a finite language.

Theorem 4. *Given a PEG G, an acyclic DFA A and an integer $r \geq 0$, the r-ED problem between $L(G)$ and $L(A)$ is* NEXPTIME-complete.

Proof. It is easy to show that the problem is NEXPTIME-hard since the r-ED between a PEL and a single string is already NEXPTIME-complete.

The following is a naive NEXPTIME algorithm for the r-ED problem.

1. Nondeterministically choose a string $w \in L(A)$.
2. Apply the NEXPTIME algorithm for the r-ED problem between the given PEG G and the string w.

Since A is acyclic, $|w| \leq |A|$, we can guess w in polynomial time to the input size. Thus, the second step is in NEXPTIME bound with respect to the original input size. □

For the general case when we consider the r-ED problem for a PEG G and an arbitrary infinite language L, we should ensure that $L(G)$ is nonempty. If not, the problem becomes undecidable since $d(L(G), \Sigma^*) \leq 0$, which is equivalent to $L(G) \cap \Sigma^* \neq \emptyset$, decides whether or not $L(G)$ is empty [4]. Therefore, from now on, we assume that a PEL is a nonempty language. Now consider when L is regular. We show that this problem is undecidable even with a fixed r.

Theorem 5. *Given a nonempty PEG G, a DFA A and a* fixed *integer $r \geq 0$, it is undecidable that the r-ED problem between $L(G)$ and $L(A)$.*

Proof. It is easy to show that the case for $r = 0$ is undecidable as we cannot decide the emptiness of a PEL [4].

The case for $r > 0$, we will show a reduction from the Post Correspondence Problem (PCP), which is a well-known undecidable problem, to the r-ED problem by construct a PEG that recognizes possible solutions for the given PCP instance and a DFA for PCP solution encodings.

Consider a PCP instance $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where every x_i's and y_i's are strings over Σ. We first construct a PEG G with the starting variable S for P and its alphabet $\widetilde{\Sigma} = \Sigma \cup \{\#, \$\}$:

$$
\begin{aligned}
S &\leftarrow \&(X!\widetilde{\Sigma})\&(Y!\widetilde{\Sigma})\widetilde{\Sigma}\widetilde{\Sigma}/\$^{r+1}!\widetilde{\Sigma} \\
X &\leftarrow x_1 X a_1 / x_2 X a_2 / \ldots / x_n X a_n / \# \\
Y &\leftarrow y_1 Y a_1 / y_2 Y a_2 / \ldots / y_n Y a_n / \#
\end{aligned}
$$

where \$, #, a_i's are new symbols not in Σ. Then $L(G)$ contains an encoded PCP solutions if P has a solution. Note that this grammar is similar to one on Ford's PCP to PEL emptiness reduction [4] but always nonempty by ensuring that $L(G)$ always contains the string $\$^{r+1}$. Also G cannot recognize the string # because S requires to recognize at least 2 symbols.

If P has no match, then the first choice rule on G cannot recognize any string, and thus G can recognize only $\$^{r+1}$. On the other hand, if P has at least one match, the first one can recognize the solution as well as the second rule.

Second, we construct a DFA A for the language

$$L(A) = \{w\#\alpha \mid w \in \Sigma^*, \alpha \in \{a_i\}^*\}$$

such that A represents a superset of valid PCP solution encodings.

Considering the PEG G and the DFA A, we can see that

- $d(L(G), L(A)) = 0 \leq r$ if P has a match and
- $d(L(G), L(A)) = r + 1 \not\leq r$ if P has no match. □

Furthermore, we cannot decide the r-ED problem between a PEL and a CFL since the classes of PELs or CFLs both contain regular languages.

Corollary 3. *Given a PEG G, a CFG (or a PEG) A and a fixed integer $r \geq 0$, the r-ED problem between $L(G)$ and $L(A)$ with bound r is undecidable.*

For other representations of r, the r-ED problems have similar results to the r-SS problems.

Corollary 4. *Given a PEG G, a finite language L and a unary (or a fixed) integer $r \geq 0$, the r-ED problem between L and $L(G)$ is* NP-complete *(done in polynomial time, respectively).*

5 Undecidability of Length Bound

Our next question is what is the length bound of shortest string in a PEG.

Definition 3. *We define $\mathcal{B}(G)$ to be a string length bound of a PEG G, where there is a nonempty string $w \in L(G)$ such that $|w| \leq \mathcal{B}(G)$.*

If we can compute $\mathcal{B}(G)$, then we can use $\mathcal{B}(G)$ to find a bound, which gives rise to the shortest string as well as the edit-distance. For example, a regular language contains at least one string whose length is shorter than the number of its FA states, a context-free language contains at least one string whose length is $2^{|V|-1}$, where V is the set of variables of corresponding CFG in CNF. Unfortunately, for PEGs, we prove that we cannot find such bound.

Corollary 5. *On a PEG G, $\mathcal{B}(G)$ is not computable.*

Proof. We prove the statement by contradiction. Suppose that there exists a TM for $\mathcal{B}$. Then, by the definition of $\mathcal{B}$, for given a PEG G, $L(G)$ must contain a string w whose length is at most $\mathcal{B}(G)$. On the other hand, if $L(G)$ has at least one string, regardless of its length, $L(G)$ is not empty.

Then, we can decide whether or not $L(G)$ is nonempty by testing membership of every string in $\Sigma^{\mathcal{B}(G)}$ on G since $\mathcal{B}(G)$ is computable. If G can recognize any of those strings, G must be nonempty. This contradicts to the fact that deciding emptiness of G is not possible. □

6 Conclusions

Recently, PEGs became popular as a recognition-based formal grammar, which is always unambiguous and has a simple parsing algorithm. We have studied the shortest string problem and the edit-distance problem on PELs since we can use a language similarity metric, like edit-distance, to quantitatively verify errors between a grammar and a target string.

We have considered the r-SS problem about whether or not a PEL contains a string of length at most r and we have proved that the r-SS problem is NEXPTIME-complete. We have also examined the r-ED problem about whether or not the edit-distance between a PEL and a language is bounded up to r. The r-ED problem is decidable when we consider the edit-distance between a PEL and a finite language, and we prove that its complexity is NEXPTIME-complete. Finally, we have demonstrated that we cannot bound the length of strings in $L(G)$, where G is a PEG.

For future work, we plan to design a nontrivial algorithm that computes $\mathcal{B}(G)$ for a nonempty PEG G. We would also compute the edit-distance with the swap operation, which is another popular edit operation.

References

1. Rosie Pattern Language. https://rosie-lang.org. Accessed 7 Jan 2020
2. Birman, A., Ullman, J.D.: Parsing algorithms with backtrack. Inf. Control **23**(1), 1–34 (1973)
3. Ford, B.: Packrat parsing: simple, powerful, lazy, linear time, functional pearl. In: Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP 2002), pp. 36–47 (2002)
4. Ford, B.: Parsing expression grammars: a recognition-based syntactic foundation. In: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, pp. 111–122 (2004)
5. Hamming, R.: Error detecting and error correcting codes. Bell Syst. Tech. J. **29**, 147–160 (1950)
6. Han, Y.S., Ko, S.K., Salomaa, K.: The edit-distance between a regular language and a context-free language. Int. J. Found. Comput. Sci. **24**(07), 1067–1082 (2013)
7. Koga, T.: Context-freeness of parsing expression languages is undecidable. Int. J. Found. Comput. Sci. **29**(7), 1203–1213 (2018)

8. Konstantinidis, S.: Computing the edit distance of a regular language. Inf. Comput. **205**(9), 1307–1316 (2007)
9. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Phys. Dokl. **10**, 707–710 (1966)
10. Loff, B., Moreira, N., Reis, R.: The computational power of parsing expression grammars. In: Hoshi, M., Seki, S. (eds.) DLT 2018. LNCS, vol. 11088, pp. 491–502. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98654-8_40. Extended version is available on arXiv: https://arxiv.org/abs/1902.08272
11. Medeiros, S., Mascarenhas, F., Ierusalimschy, R.: From regexes to parsing expression grammars. Sci. Comput. Program. **93**, 3–18 (2014)
12. Mohri, M.: Edit-distance of weighted automata: general definitions and algorithms. Int. J. Found. Comput. Sci. **14**(06), 957–982 (2003)
13. Ng, T., Rappaport, D., Salomaa, K.: State complexity of neighbourhoods and approximate pattern matching. Int. J. Found. Comput. Sci. **29**(02), 315–329 (2018)
14. Okhotin, A.: Conjunctive grammars. J. Autom. Lang. Comb. **6**(4), 519–535 (2001)
15. Okhotin, A.: Unambiguous Boolean grammars. Inf. Comput. **206**(9–10), 1234–1247 (2008)
16. Povarov, G.: Descriptive complexity of the hamming neighborhood of a regular language. In: Proceedings of the 1st International Conference on Language and Automata Theory and Applications, pp. 509–520 (2007)
17. Sipser, M.: Introduction to the Theory of Computataion, 3rd edn. Cengage Learning, Boston (2013)