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

Hyunjoon Cheon Yo-Sub Han

Yonsei University, Republic of Korea

Developments in Language Theory 2021, August 16–20.

Overview

Backgrounds

Main results

Conclusion

Overview

Turing machines

Definition

A (nondeterministic) Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ consists of

- a finite set Q of states,
- a finite alphabet Σ for input,
- a finite tape alphabet $\Gamma \supseteq \Sigma$,
- a set $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ of transitions,
- an initial state q_0, an accepting state q_a and a rejecting state q_r.

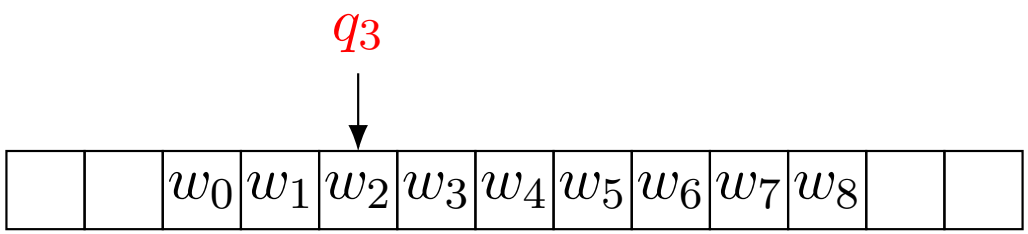
Turing machines

Configurations

Definition

A configuration $c \in (Q \cup \Gamma)^*$ of a TM M is a string that represents the current state, head position and the tape. A configuration c is *accepting* (*rejecting*) if c is on q_a (q_r, resp.)

Example



$$c = w_0 w_1 q_3 w_2 w_3 w_4 w_5 w_6 w_7 w_8$$

Turing machines

Languages

Definition

- A TM M *accepts* (*rejects*) a string $w \in \Sigma^*$ if, starting from $q_0 w$, M has a transition path that reaches an accepting (rejecting, resp.) configuration by following δ.
- $L(M) = \{w \mid M \text{ accepts } w\}$.
- M *halts on* w if M reaches either q_a or q_r on w.

Parsing expressions

Definition

A parsing expression (PE) e is generated on the following grammar:

$$e \to \lambda \mid \sigma \in \Sigma \mid A \in V \mid (e) \mid ee \mid e/e \mid !e.$$

The key differences from CFG productions are

1. ordered choices (e/e), which *sequentially* try each rule and
2. not-predicates ($!e$), which consume λ iff e does not match.

Parsing expression grammars

Definition

A parsing expression grammar (PEG) $G = (V, \Sigma, R, S)$ consists of

- a finite set V of variables,
- a finite alphabet Σ,
- a mapping $R : V \to e$ and
- a starting variable $S \in V$.

- . $\iff X \leftarrow \sigma_1/\sigma_2/\cdots/\sigma_n$ (a character)
- $e^* \iff X \leftarrow eX/\lambda$ (repetition)
- $\&e \iff !!e$ (and-predicate)

Parsing expression languages

Definition

Given a PEG G, its language (parsing expression language, PEL) $L(G)$ is the set of strings that G *recognizes* their prefix.

Example

$S \leftarrow \&(ab)$ *consumes* only an empty string λ (when it *recognizes* ab), but it can *recognizes* any string in $L(ab.^*)$.

How PEGs recognize a string

The parsing mechanism is exactly the same to that of recursive descendant parsers (RDPs).

x

procedure READ(x)
  if x is the next character **then**
    consume x
  else
    error
  end if
end procedure

How PEGs recognize a string

The parsing mechanism is exactly the same to that of recursive descendant parsers (RDPs).

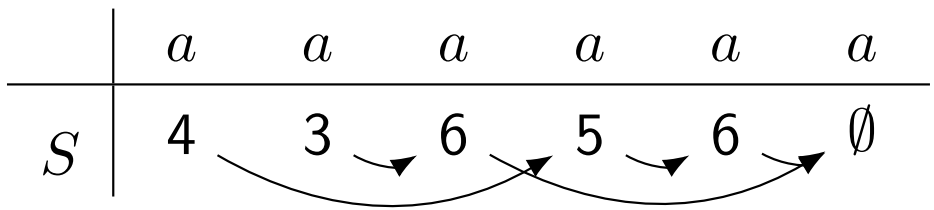
$S \to aSa/aa$

procedure PARSE-S
  try
    READ(a)
    PARSE-S
    READ(a)
  end try
  try
    READ(a)
    READ(a)
  end try
  error
end procedure

How PEGs recognize a string

Linear-time parsing algorithm

By memoizing parsing results for each variable-position pair, one can achieve linear-time performance on any given strings [Ford02]. Note that $G = [S \leftarrow aSa/aa]$ consumes only the first four a's in a^6.



PE characteristics

- Deterministic: There is at most one recognizing path for any given string. For a PE aa/aab and a string $w = aab$, the PE *always* consumes the first two symbols (aa).
- Greedy: PE reads as many as it can. For example, a PE a^*a represents an empty language, since the PE cannot consume a right after consuming *every* a.

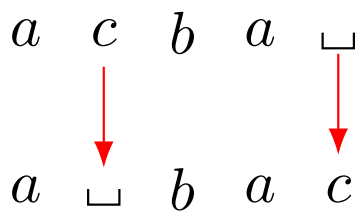
Edit-distance

Definition

The edit-distance $d(x, y)$ of two strings x and y is the number of minimum edits to transform x into y.

Example

$d(acba, abac) = 2$



Edit-distance

Extensions

Definition

Given a string w and a language L, the edit-distance between them is

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

Definition

Given two languages L and M, the edit-distance between them is

$$d(L, M) = \min_{x \in L} d(w, M).$$

Overview

Problems

Problem (r-shortest string (r-SS) problem)

Given a language L and a nonnegative integer r, r-SS on L is to decide whether or not there exists a string $w \in L$, $|w| \leq r$.

Problem (r-edit-distance (r-ED) problem)

Given a string w, a language L and a nonnegative integer r, r-ED between w and L is to decide whether or not $d(w, L) \leq r$.

Example

$G = [S \leftarrow aSa/aa]$ satisfies 2-SS ($aa \in L(G)$) but not 1-SS.

Summary

- r-SS on a PEL is NEXPTIME-complete.
- r-ED between a finite language and a PEL is NEXPTIME-complete.
- r-ED between an infinite language and a PEL is undecidable.

r-SS NEXPTIME

Proof

Lemma

Given a PEG G, r-SS on $L(G)$ is NEXPTIME.

Proof.

Guess $w \in \Sigma^{\leq r}$ and test acceptance. If there is a string accepted, return YES. □

Bounded halting problem

Theorem

Given a TM M, an input $w \in \Sigma^*$ and a nonnegative integer k, it is NEXPTIME-complete *to decide whether or not M halts on w in at most k steps.*

r-SS NEXPTIME-hardness

Lemma

Given a PEG G, r-SS on $L(G)$ is NEXPTIME-hard.

Idea

- Reduction from the bounded halting problem: If M halts on w with at most k steps, there must be a finite sequence of configurations (a computation) $c_1\# \cdots \#c_n\#$ where $1 \leq n \leq k$.
- This is valid if 1) $c_1 \equiv q_0 w$, 2) c_n is on either q_a or q_r and 3) every step follows δ.

r-SS NEXPTIME-hardness

Design PEG rules

1. I recognizes the initial configuration $q_0w\#$, but does not consume it.
2. F consumes a final configuration and no more.
3. D *recursively* recognizes valid transitions and consumes the first configuration. (l is the length of a configuration)
4. M recognizes a computation.

$$I \leftarrow \&(q_0w\#)$$

r-SS NEXPTIME-hardness

Design PEG rules

1. I recognizes the initial configuration $q_0w\#$, but does not consume it.
2. F consumes a final configuration and no more.
3. D *recursively* recognizes valid transitions and consumes the first configuration. (l is the length of a configuration)
4. M recognizes a computation.

$$F \leftarrow \&(Q_f\#)(\Gamma \cup Q)^l\#!.$$
$$Q_f \leftarrow (q_a/q_r)\Gamma^*/\Gamma Q_f$$

r-SS NEXPTIME-hardness

Design PEG rules

1. I recognizes the initial configuration $q_0w\#$, but does not consume it.
2. F consumes a final configuration and no more.
3. D *recursively* recognizes valid transitions and consumes the first configuration. (l is the length of a configuration)
4. M recognizes a computation.

$$
\begin{aligned}
&D \leftarrow F \\
&\quad /\sigma' p \sigma \& (.^{l-2} q \sigma' \gamma) D && \text{(if } (p, \sigma, q, \gamma, L) \in \delta) \\
&\quad /p \sigma \& (.^{l-1} \gamma q) D && \text{(if } (p, \sigma, q, \gamma, R) \in \delta) \\
&\quad /\sigma \& (.^{l} \sigma) D
\end{aligned}
$$

r-SS NEXPTIME-hardness

Design PEG rules

1. I recognizes the initial configuration $q_0w\#$, but does not consume it.
2. F consumes a final configuration and no more.
3. D *recursively* recognizes valid transitions and consumes the first configuration. (l is the length of a configuration)
4. M recognizes a computation.

$$M \leftarrow ID$$

r-SS NEXPTIME-hardness

Fixing l

- Idea: M's head moves at most k times.
- We can pad the input with the blank symbols (B) to maintain the same length for each configuration; $l = 2k + 1$.

$$I \leftarrow B^k q_0 w B^{k-|w|} \#$$

r-SS NEXPTIME-hardness

Reduce rules

I and D has rules of $.^{O(l)} = .^{O(k)}$ that uses *exponential* space to describe.
We can reduce them into polynomial size with the X_i gadgets.

$$D \leftarrow F$$
$$/\sigma' p \sigma \& (.^{l-2} q \sigma' \gamma) D$$
$$/p \sigma \& (.^{l-1} \gamma q) D$$
$$/\sigma \& (.^{l} \sigma) D$$

$$X_n \leftarrow X_{n-1} X_{n-1}$$
$$\cdots$$
$$X_2 \leftarrow X_1 X_1$$
$$X_1 \leftarrow X_0 X_0$$
$$X_0 \leftarrow .$$

r-SS NEXPTIME-hardness

Reduce rules

I and D has rules of $.^{O(l)} = .^{O(k)}$ that uses *exponential* space to describe. We can reduce them into polynomial size with the X_i gadgets.

$$
\begin{aligned}
D &\leftarrow F \\
&\quad /\sigma' p \sigma \& (.^{l-2} q \sigma' \gamma) D \\
&\quad /p \sigma \& (.^{l-1} \gamma q) D \\
&\quad /\sigma \& (.^{l} \sigma) D
\end{aligned}
$$

$$
\begin{aligned}
X_n &\leftarrow X_{n-1} X_{n-1} \\
&\quad \cdots \\
X_2 &\leftarrow X_1 X_1 \\
X_1 &\leftarrow X_0 X_0 \\
X_0 &\leftarrow .
\end{aligned}
$$

r-SS NEXPTIME-hardness

Proof

- $|G|$ is polynomial to $O(|M| + |w| + |k|)$.
- If $L(G)$ has a string of at most $r = (l + 1) \cdot k$, M halts on w. Note that r has size of $O(|w| + |k|)$. □

Theorem

r-SS is NEXPTIME-complete.

r-edit-distance

Problem

Given a string w, a language L and a nonnegative integer r, r-ED between w and L is to decide whether or not $d(w, L) \leq r$.

A relation between r-SS and r-ED

Lemma

Given a language L and an integer r, the r-SS problem for L is reducible to the r-ED between λ and L.

In other words, r-ED between a string and a PEL is NEXPTIME-hard*.*

r-ED NEXPTIME

Algorithm

Lemma

Given a PEG G and a string w, r-ED between w and G is in NEXPTIME*.*

Proof.

Given a string w, we can guess an edit sequence of at most r length. Applying this sequence and testing its acceptance is done in EXPTIME nondeterministically. □

Theorem

r-ED between a string and a PEL is NEXPTIME-complete*.*

r-ED variants

PEL and finite language

Theorem

Given a PEG G and an acyclic DFA A, r-ED between $L(G)$ and $L(A)$ is NEXPTIME-complete.

Proof.

Proving NEXPTIME is similar to w-G case; Guess a $w \in L(A)$ and apply the NEXPTIME algorithm.
For NEXPTIME-hardness, We already showed NEXPTIME-hardness even when A represents a single string. □

r-ED variants

PEL and infinite language

Theorem

Given a PEG G and a DFA A, r-ED between $L(G)$ and $L(A)$ is undecidable, even for fixed r.

- $r = 0$ case is simple: we cannot decide emptiness of PEL $L(G) \cap L(A)$.
- For $r > 0$, we reduce from PCP.

r-ED variants

PEL and infinite language

Theorem

Given a PEG G and a DFA A, r-ED between $L(G)$ and $L(A)$ is undecidable, even for fixed r.

Proof.

Consider a PCP instance $P = \{(x_i, y_i) \mid 1 \leq i \leq n\}$. The following PEG G recognizes PCP solutions and $\$^{r+1}$.

$$S \leftarrow \&(X!.)\&(Y!.)...^*/\$^{r+1}!.$$
$$X \leftarrow x_1 X a_1 / \cdots / x_n X a_n / \#$$
$$Y \leftarrow y_1 X a_1 / \cdots / y_n X a_n / \#$$

r-ED variants

PEL and infinite language

Theorem

Given a PEG G and a DFA A, r-ED between $L(G)$ and $L(A)$ is undecidable, even for fixed r.

Proof.

On the other hand, the following $L(A)$ is for the strings of PCP solution form.

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

We can observe that

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

□

Undecidability of length bound

Corollary

Given a PEG G, $\mathcal{B}(G)$ denotes a bound for string length of G: there exists $w \in L(G)$ such that $|w| \leq \mathcal{B}(G)$, which is not computable.

Proof.

If $\mathcal{B}(G)$ is computable, we can decide whether or not $L(G) = \emptyset$ by testing all strings in $\Sigma^{\mathcal{B}(G)}$; which is undecidable. □

Overview

Summary

- r-SS on a PEL is NEXPTIME-complete.
- r-ED between a finite language and a PEL is NEXPTIME-complete.
- r-ED between an infinite language and a PEL is undecidable.

Future works

- For nonempty PEG G, how to compute $\mathcal{B}(G)$ efficiently?
- For an acyclic DFA A, how to compute $d(L(G), L(A))$?

Any questions?