IEICE TRANS. INF. & SYST., VOL.E103–D, NO.3 MARCH 2020

PAPER Special Section on Foundations of Computer Science —Frontiers of Theory of Computation and Algorithm—

An Efficient Learning Algorithm for Regular Pattern Languages Using One Positive Example and a Linear Number of Membership Queries

Satoshi MATSUMOTO[†a)], **Tomoyuki UCHIDA**[††], **Takayoshi SHOUDAI**[†††], **Yusuke SUZUKI**[††], ***and*** **Tetsuhiro MIYAHARA**[††], ***Members***

SUMMARY A regular pattern is a string consisting of constant symbols and distinct variable symbols. The language of a regular pattern is the set of all constant strings obtained by replacing all variable symbols in the regular pattern with non-empty strings. The present paper deals with the learning problem of languages of regular patterns within Angluin's query learning model, which is an established mathematical model of learning via queries in computational learning theory. The class of languages of regular patterns was known to be identifiable from one positive example using a polynomial number of membership queries, in the query learning model. In present paper, we show that the class of languages of regular patterns is identifiable from one positive example using a linear number of membership queries, with respect to the length of the positive example.

key words: *pattern language, membership query, query learning, computational learning theory*

1. Introduction

A *pattern*, introduced by Angluin [3], is a string consisting of constant symbols and variable symbols. A *regular pattern* is a pattern in which each variable symbol occurs at most once. The *pattern language* of a pattern π is the set of all strings w consisting of constant symbols such that w is obtained from π by replacing all variable symbols with non-empty strings consisting of constant symbols. For example, let $\pi_1 = xaybaz$ and $\pi_2 = xaybyz$ be two strings consisting of constant symbols a, b and variable symbols x, y, z. The string π_1 is a regular pattern. The string π_2 is a pattern but not a regular pattern. For the regular pattern $\pi_3 = axabacy$, the string $w_1 = abbabacbacb$ is included in the pattern language of π_3, because w_1 is obtained from π_3 by replacing variable symbols x and y with non-empty strings bb and $bacb$, respectively.

In this paper, we deal with the learning problem of languages of regular patterns within Angluin's query learning model [1], which is an established mathematical model of learning via queries in computational learning theory. In the query learning model, a learning algorithm collects information about a language to be learned, called a target language, by accessing oracles that answer specific types of queries. The query learning model is a mathematical model of a data mining strategy using queries for large databases (e.g., [2]). A query as to whether or not an example exists in the database, which is called a membership query in the query learning model, is frequently performed in data mining using databases. Hence, to extract characteristic features from large databases, data mining algorithms that identify features using fewer membership queries are required. From this motivation, in this paper, we consider a query learning algorithm that uses a linear number of membership queries with respect to the length of a given string, called a positive example, contained in a target language.

As related work, there are many researches [1], [4]–[6] about the learning problem of pattern languages in the same query learning model. Angluin [1] showed that the class of pattern languages is not identifiable using polynomial numbers of membership queries and restricted equivalence queries. Here, a restricted equivalence query is a call to an oracle as to whether or not the pattern language derived by a pattern given as input is equal to the target language. Because of this result, Marron [4] presented the query learning setting in which the learner initially receives a positive example of the target language before starting the process of asking queries.

We discuss a previous work on the learnability of regular pattern languages from one positive example using membership queries. Let π_* be a regular pattern that generates the target regular pattern language. Matsumoto and Shinohara [5] presented a query learning algorithm for identifying π_* from one positive example w using $O(|w|^2)$ membership queries, where $|w|$ is the length of w. Their algorithm is as follows. (1) By removing some constant symbols from a given positive example w, a shortest positive example w' with $|w'| = |\pi_*|$ is obtained using $O(|w|^2)$ membership queries. (2) The target regular pattern π_* is identified by determining whether or not the i-th symbol of π_* is a constant symbol using a membership query for w'_i for each i $(1 \leq i \leq |w'|)$, where w'_i is a string obtained from w' by replacing the i-th symbol of w' with another symbol.

Moreover, Matsumoto and Shinohara [5] reduced the number of membership queries to be linear for a non-trivial

Manuscript received April 26, 2019.
Manuscript revised September 26, 2019.
Manuscript publicized December 23, 2019.

[†]The author is with School of Science, Tokai University, Hiratsuka-shi, 259–1292 Japan.
[††]The authors are with Graduate School of Information Sciences, Hiroshima City University, Hiroshima-shi, 731–3194 Japan.
[†††]The author is with Faculty of Contemporary Business, Kyushu International University, Kitakyushu-shi, 805–8512 Japan.
a) E-mail: matsumoto@tsc.u-tokai.ac.jp

DOI: 10.1587/transinf.2019FCP0009

subclass of regular pattern languages. They introduced a *critical pattern*, which is a special type of a regular pattern. They showed that their learning algorithm obtains a shortest positive example w' of a regular pattern π_* from a given positive example w using $O(|w|)$ membership queries, when π_* is not a critical pattern. They showed that a non-trivial subclass of regular pattern languages is identified from one positive example using a linear number of membership queries, with respect to the length of the positive example. The non-trivial subclass of regular pattern languages is a class of the languages of regular patterns that are not critical patterns.

In this paper, we show that the full class of regular pattern languages is identifiable from one positive example using a linear number of membership queries, with respect to the length of the positive example. We present a query learning algorithm that is to identify a target regular pattern language from a given positive example using a linear number of membership queries. To identify a target regular pattern π_* from a given positive example w using $O(|w|)$ membership queries, we propose a concept, called the *left non-redundant positive example* of π_* at a position i ($1 \leq i \leq |\pi_*|$). For a target pattern π_*, the proposed algorithm determines whether or not the i-th symbol of π_* is a constant symbol using $O(|w|)$ membership queries by making a left non-redundant positive example of π_* at i instead of making a shortest positive example of π_* obtained by the previous algorithm in [5]. By using our proposed algorithm, even if a target regular pattern π_* is a critical pattern, we exactly identify π_* from one positive example using a linear number of membership queries. That is, this result shows that a learnable class is extended from the subclass of regular pattern languages in [5] to the full class of regular pattern languages in the same query learning setting as in [5]. Matsumoto and Shinohara [5] also considered the learnability of the class of pattern languages generated by patterns such that each variable symbol occurs at most k times in each pattern for a positive integer k. But in this paper we focus on the learnability of the class of regular pattern languages.

As other related work, there are query learning algorithms for identifying the classes of other languages, including the regular languages [7], the erasing pattern languages [8], the languages derived from elementary formal systems [9], [10], the tree languages derived from tree patterns [11], the tree languages derived from primitive formal ordered tree systems [12], and the sets of binary decision diagrams [13], [14].

This paper is the full version of the paper [15], with complete definitions and proofs, discussions about the efficiency of the proposed algorithm, and experimental results of the proposed and previous algorithms.

This paper is organized as follows. In Sect. 2, we introduce a regular pattern, its language and the query learning model of Angluin [1]. In Sect. 3, by presenting a query learning algorithm, we show that the full class of regular pattern languages is exactly learnable from only one positive example using a linear number of membership queries, with respect to the length of the positive example. In Sect. 4, we discuss the efficiency of the proposed algorithm by comparing with the previous algorithm in [5] with respect to the number of membership queries. In Sect. 5, we conclude this paper and discuss future work.

2. Preliminaries

In this section, we introduce a regular pattern and its language. Then, we introduce the query learning model proposed by Angluin [1].

2.1 Regular Pattern and Its Language

Let Σ be a nonempty finite set of constant symbols. Let X be an infinite set of variable symbols such that $\Sigma \cap X = \emptyset$ holds. Then, a *string* on $\Sigma \cup X$ is a sequence of symbols in $\Sigma \cup X$. Particularly, the string having no symbol is called the empty string and is denoted by ε. We denote by $(\Sigma \cup X)^*$ the set of all strings on $\Sigma \cup X$ and by $(\Sigma \cup X)^+$ the set of all strings on $\Sigma \cup X$ except ε, i.e., $(\Sigma \cup X)^+ = (\Sigma \cup X)^* \setminus \{\varepsilon\}$. A *pattern* on $\Sigma \cup X$ is a string in $(\Sigma \cup X)^+$. Note that the empty string ε is not a pattern on $\Sigma \cup X$. We denote by Σ^+ the set of all strings on Σ except ε. A string in Σ^+ is called a *constant string*. Then, a pattern π on $\Sigma \cup X$ is said to be *regular* if each variable symbol in X appears at most once in π. The set of all regular patterns on $\Sigma \cup X$ is denoted by $\mathcal{RP} \subsetneq (\Sigma \cup X)^+$. Hereafter, we omit Σ and X if they are obvious from the context. A *substitution* θ is a mapping from $(\Sigma \cup X)^+$ to Σ^+ such that (1) θ is a homomorphism with respect to string concatenation, denoted by '$\cdot$', that is, for two patterns $\pi_1, \pi_2 \in (\Sigma \cup X)^+$, $\theta(\pi_1 \cdot \pi_2) = \theta(\pi_1) \cdot \theta(\pi_2)$ holds, and (2) for each constant symbol $a \in \Sigma$, $\theta(a) = a$ holds. The notation $\{x_1 := w_1, \cdots, x_n := w_n\}$ denotes a substitution that replaces each variable symbol x_i with a constant string w_i for i with $1 \leq i \leq n$, where $x_1, \cdots, x_n$ are mutually distinct variable symbols. For a pattern π, $\theta(\pi)$ denotes the constant string obtained from π by replacing variable symbols with constant strings according to θ. For a pattern π, the *pattern language* of π, denoted by $L(\pi)$, is the set of all constant strings w in Σ^+ such that w is obtained from π by replacing all variable symbols in π with constant strings, that is, $L(\pi) = \{w \in \Sigma^+ \mid w = \theta(\pi)$ for some substitution $\theta\}$. We define $\mathcal{RPL} = \{L(\pi) \mid \pi \in \mathcal{RP}\}$. A language in $\mathcal{RPL}$ is called a *regular pattern language*.

Example 1: Let $\Sigma = \{a, b\}$ and $X = \{x, y, z, \ldots\}$. Let $\pi_1 = abxby$ be a regular pattern on $\Sigma \cup X$. Consider substitutions $\theta_1 = \{x := a, y := a\}$, $\theta_2 = \{x := a, y := aba\}$ and $\theta_3 = \{x := bbab, y := aab\}$. Then, we have $\theta_1(\pi_1) = ababa$, $\theta_2(\pi_1) = abababa$, $\theta_3(\pi_1) = abbbabbaab$, and $L(\pi_1) = \{ababa, ababb, abbba, abbbb, abaaba, ababba, \ldots\}$.

Next, we prepare some notations on strings. For a string $w \in (\Sigma \cup X)^*$, the length of w, denoted by $|w|$, is the number of symbols composing w, e.g., $|\varepsilon| = 0$ and $|abcxay| = 6$. For a string $w \in (\Sigma \cup X)^+$ and a positive integer i with $1 \leq i \leq |w|$, we denote by $w[i]$ the i-th symbol

of w. For two positive integers i, j with $1 \leq i \leq j \leq |w|$, we denote by $w[i:j]$ the substring $w[i]w[i+1]\cdots w[j]$. Note that $w[i:i] = w[i]$. Let w be a pattern on $\Sigma \cup X$, a the empty string ε or a symbol in $\Sigma \cup X$ and i a positive integer with $1 \leq i \leq |w|$. Then, we denote by $w.rep(i,a)$ the pattern obtained from w by replacing the i-th symbol of w with a. If a pattern π contains a variable symbol, we denote by $rmvs(\pi)$ the position of the rightmost variable symbol in π, that is, $rmvs(\pi) = \max\{i \in \{1,\ldots,|\pi|\} \mid \pi[i] \text{ is a variable symbol}\}$. Otherwise, we define $rmvs(\pi) = 0$. Note that $0 \leq rmvs(\pi) \leq |\pi|$.

Example 2: Let $\Sigma = \{a,b,c\}$ and $\pi_2 = abcabc \in \Sigma^+$ be a pattern. Then, we have $\pi_2[2] = b$, $\pi_2[2:4] = bca$, $\pi_2.rep(2,a) = aacabc$, and $\pi_2.rep(2,\varepsilon) = acabc$.

Example 3: Let $\Sigma = \{a,b,c\}$ and $X = \{x,y,z,\ldots\}$. Let $\pi_2 = abcabc$, $\pi_3 = axyaxa$, and $\pi_4 = axybaza$ be patterns on $\Sigma \cup X$. Then, we have $rmvs(\pi_2) = 0$, $rmvs(\pi_3) = 5$ and $rmvs(\pi_4) = 6$.

2.2 Learning Model

Let $\mathcal{L}$ be a class consisting of sets of constant strings such that each set in $\mathcal{L}$ has its own representation of finite length. Let R be the set of representations for all sets of constant strings in $\mathcal{L}$. For each representation $r \in R$, we denote by $L(r)$ the set of constant strings that is represented by r. For example, for a set L of constant strings, a regular pattern π is a representation of L if $L = L(\pi)$ holds.

Let $L_* \in \mathcal{L}$ be a learning target. A constant string $w \in \Sigma^+$ is said to be a *positive example* of L_* if $w \in L_*$ holds. In the query learning model presented by Angluin [1], learning algorithms can access *oracles* that will answer queries about the target L_*. In this paper, we consider the *membership query* defined as follows. The input is a constant string $w \in \Sigma^+$. The output is "yes" if $w \in L_*$ holds and "no" otherwise. We denote by **MQ** the oracle that answers membership queries. For a constant string $w \in \Sigma^+$, a notation $\mathbf{MQ}(w)$ denotes the answer of **MQ** for the membership query in the case that the input of **MQ** is w.

Example 4: Let $L(abxby)$ be a learning target. $\mathbf{MQ}(abaaba)$ is "yes", and $\mathbf{MQ}(abaaa)$ is "no".

A learning algorithm $\mathcal{A}$ is said to *exactly identify a target* $L_* \in \mathcal{L}$ if $\mathcal{A}$ outputs a representation $r \in R$ satisfying $L(r) = L_*$.

3. An Efficient Query Learning Algorithm for Regular Pattern Languages

In this section, we present a learning algorithm *LearningStringPattern* (Algorithm 1) that exactly identifies any target language in $\mathcal{RPL}$. Let L_* be a target in $\mathcal{RPL}$ and π_* a regular pattern in $\mathcal{RP}$ such that $L_* = L(\pi_*)$ holds. We show in Theorem 1 that Algorithm *LearningStringPattern* outputs a regular pattern $\pi \in \mathcal{RP}$ with $L(\pi) = L(\pi_*)$ from one positive example w using $O(|w|)$ membership queries, where $|\Sigma| \geq 2$.

Algorithm 1 *LearningStringPattern*

Input: A constant string w in $L(\pi_*)$
Output: A pattern π with $L(\pi) = L(\pi_*)$

```
1: i := 1, k := 0, vSet := ∅;
2: while i ≤ |w| do
3:     w := ShrinkString(w, i, k);
4:     k := IdentifyVariable(w, i, k);
5:     if k ≠ 0 and k ∉ vSet then
6:         vSet := vSet ∪ {k};
7:     end if
8:     i := i + 1;
9: end while
10: π := w;
11: for all i ∈ vSet do
12:     Let x be a new variable symbol that does not appear in π.
13:     π := π.rep(i, x);
14: end for
15: output π;
```

In the case of $|\Sigma| = 1$, we can easily show that there exists a learning algorithm that exactly identifies any target language in $\mathcal{RPL}$ from one positive example using a linear number of membership queries. The learning algorithm is as follows. Suppose that $\Sigma = \{a\}$. Let w be a constant string on Σ. We can see that $\mathbf{MQ}(w)$ is "yes" if and only if $w \in L(\pi_*)$ and $|w| \geq |\pi_*|$. If a constant string w is given as a positive example of L_*, by asking $|w|-|\pi_*|+1$ membership queries for $aa\cdots a$ of length k downwardly from $k = |w|-1$ to $|\pi_*|-1$, we obtain the shortest positive example $aa\cdots a$ of length $|\pi_*|$. If the answer of the first membership query of them is "no", because the constant string w is already the shortest positive example, we ask the next membership query for $aa\cdots a$ of length $|\pi_*|+1$. If the answer of the next membership query is "no", the target pattern π_* is w itself. On the other hand, if the answer of the next membership query is "yes", for the shortest positive example $aa\cdots a$ of length $|\pi_*|$, by replacing the first symbol a with a variable symbol $x \in X$, we get the regular pattern $\pi = xa\cdots a$ of length $|\pi_*|$. Because $|\Sigma| = 1$, we easily see that $L(\pi) = L(\pi_*)$ holds.

Hereafter, in the case of $|\Sigma| \geq 2$, by presenting Algorithm *LearningStringPattern* (Algorithm 1), we give the following theorem.

Theorem 1: Let L_* be a target in $\mathcal{RPL}$ and π_* a regular pattern in $\mathcal{RP}$ such that $L_* = L(\pi_*)$ holds. Algorithm *LearningStringPattern* outputs a regular pattern π with $L(\pi) = L(\pi_*)$ from one positive example w using $O(|w|)$ membership queries, where $|\Sigma| \geq 2$.

In order to prove this theorem, we give several definitions and lemmas in the following subsections. Finally, we give a proof of the theorem at the last of this section.

3.1 Left Non-Redundant Positive Example

The next concept plays an important role in Lemmas 1–3.

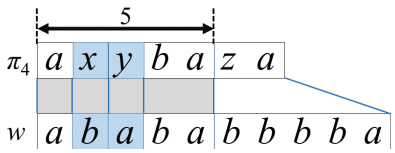

Fig. 1 The constant string *abababbbba* is a left non-redundant positive example of $L(\pi_4)$ at 5.

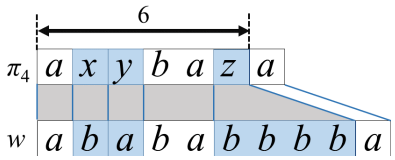

Fig. 2 The constant string *abababbbba* is not a left non-redundant positive example of $L(\pi_4)$ at 6.

Left Non-Redundant Positive Example: Let π be a regular pattern, w a positive example of $L(\pi)$, and i a positive integer with $1 \leq i \leq |\pi|$. We say that w is a *left non-redundant positive example of* $L(\pi)$ *at* i if there is no positive integer j with $i < j \leq |w|$ such that for some substitution φ, both $w = \varphi(\pi)$ and $w[1:j] = \varphi(\pi[1:i])$ hold. We denote by $L^{lnr}(\pi, i)$ the set of all left non-redundant positive examples of $L(\pi)$ at i.

We explain the meaning of a left non-redundant positive example of $L(\pi)$ at i. Let w be a left non-redundant positive example of $L(\pi)$ at i. For any substitution θ with $w = \theta(\pi)$, $w[1:i] = \theta(\pi[1:i])$ holds and each variable symbol in the left side of $\pi[i]$ is replaced with a constant symbol. Moreover, for any k $(1 \leq k \leq i)$, the constant string obtained from w by removing the constant symbol at k is not a positive example of π. In this sense, we say that w is a non-redundant positive example of $L(\pi)$ at i.

Example 5: Let $\pi_4 = axybaza$ be a regular pattern. The constant string *abababbbba* is a left non-redundant positive example of $L(\pi_4)$ at 5 (Fig. 1). The reason is as follows. If $abababbbba = \theta(\pi_4)$, then θ is a substitution such that by applying it to π_4, x and y are replaced with b and a respectively. For each $k = 1, 2, 3, 4, 5$, $abababbbba.rep(k, \varepsilon)$ is not a positive example of $L(\pi_4)$. And there is no positive integer j with $5 < j \leq 10 = |abababbbba|$ such that for some substitution φ, both $abababbbba = \varphi(\pi_4)$ and $abababbbba[1:j] = \varphi(\pi_4[1:5]) = \varphi(axyba)$ hold. However, the constant string *abababbbba* is a not left non-redundant positive example of $L(\pi_4)$ at 6 (Fig. 2), since there exist an integer $j = 9$ and a substitution $\theta = \{x := b, y := a, z := bbbb\}$ such that $abababbbba[1:j] = abababbbb = \theta(\pi_4[1:6]) = \theta(axybaz)$ and $abababbbba = \theta(\pi_4)$ hold.

3.2 Outline of Algorithm *LearningStringPattern*

Let π_* be a target regular pattern. Algorithm *LearningStringPattern* (Algorithm 1) calls two procedures *ShrinkString* (Procedure 2) and *IdentifyVariable* (Procedure 3) at lines 3 and 4, respectively. We regard $L^{lnr}(\pi_*, 0)$ as $L(\pi_*)$. For a positive integer i $(1 \leq i \leq |\pi_*|)$, given a positive example w in $L^{lnr}(\pi_*, i-1)$, Procedure *ShrinkString* makes a next positive example in $L^{lnr}(\pi_*, i)$ from w, and then Procedure *IdentifyVariable* decides whether or not the i-th symbol of the new positive example corresponds to a variable symbol. Finally, we obtain a shortest positive example w in $L^{lnr}(\pi_*, |\pi_*|)$ and all indexes of variable symbols in w as a set *vSet*.

One of the ideas of our algorithm is the following proposition, whose proof is obvious.

Proposition 1: Let $\pi_* = s_1 x_1 s_2 x_2 \cdots s_n x_n s_{n+1}$ be a regular pattern in $\mathcal{RP}$ where $s_1, s_2, \ldots, s_{n+1} \in \Sigma^*$, and $x_1, \ldots, x_n \in X$ for some integer n. Let w be a constant string in $L^{lnr}(\pi_*, i-1)$ for a positive integer i $(1 \leq i \leq |\pi_*|)$, and $x_1, \ldots, x_r$ $(0 \leq r \leq n)$ the variable symbols appearing in $\pi_*[1:i-1]$. For convenience, we regard $\pi_*[i:0]$ as ε. Because $w \in L^{lnr}(\pi_*, i-1)$ holds, any substitution θ with $\theta(\pi_*) = w$ contains $\{x_1 := a_1, \ldots, x_{r-1} := a_{r-1}, x_r := a_r, x_{r+1} := s'_{r+1}, \ldots, x_n := s'_n\}$ as a subset, where $a_1, \ldots, a_r \in \Sigma$ and $s'_{r+1}, \ldots, s'_n \in \Sigma^+$. Then the next two statements hold.

1. If $w.rep(i, \varepsilon) \in L(\pi_*)$ holds, then there exist $n - r + 1$ constant strings $s''_r, s''_{r+1}, \ldots, s''_n$ such that for the substitution $\theta' = \{x_1 := a_1, \ldots, x_{r-1} := a_{r-1}, x_r := s''_r, x_{r+1} := s''_{r+1}, \ldots, x_n := s''_n\}$, $\theta'(\pi_*) = w.rep(i, \varepsilon)$ holds (Fig. 3).
2. If $w.rep(i, a) \in L(\pi_*)$ holds for a constant symbol $a \in \Sigma \setminus \{w[i]\}$, then there exist $n - r + 1$ constant strings $s''_r, s''_{r+1}, \ldots, s''_n$ such that for the substitution

Procedure 2 *ShrinkString*

Input: A constant string w in $L(\pi_*)$, a positive integer i and a nonnegative integer k
Output: A constant string w in $L(\pi_*)$
1: **if MQ**$(w.rep(i, \varepsilon))$ = "yes" **then**
2: $w := w.rep(i, \varepsilon)$;
3: **while MQ**$(w.rep(i, \varepsilon))$ = "yes" **do**
4: $w := w.rep(i, \varepsilon)$;
5: **end while**
6: **if** $k > 0$ **then**
7: **while MQ**$(w.rep(k, \varepsilon))$ = "yes" **do**
8: $w := w.rep(k, \varepsilon)$;
9: **end while**
10: **end if**
11: **end if**
12: **output** w;

Procedure 3 *IdentifyVariable*

Input: A constant string w in $L(\pi_*)$, a positive integer i and a nonnegative integer k
Output: a nonnegative integer $k = rmvs(\pi_*[1:i])$
1: Let a be a constant symbol in $\Sigma \setminus \{w[i]\}$;
2: **if MQ**$(w.rep(i, a))$ = "yes" **then**
3: $w' := w.rep(i, a)$;
4: /* If $k = 0$, then **MQ**$(w'.rep(k, \varepsilon))$ is not called. */
5: **if** $k = 0$ **or MQ**$(w'.rep(k, \varepsilon))$ = "no" **then**
6: $k := i$;
7: **end if**
8: **end if**
9: **output** k;

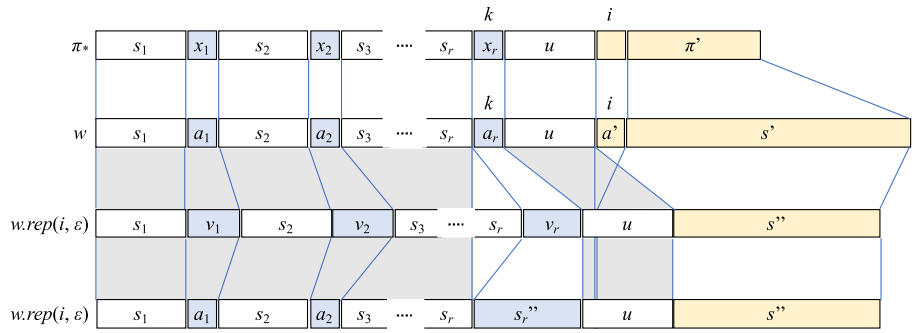

Fig. 3 Proposition 1 (1). Let $\pi_* = s_1x_1s_2x_2\cdots s_nx_ns_{n+1}$. Let r be the number of variables in $\pi_*[1 : i-1]$, u a prefix of s_{r+1}, and k the position of the rightmost variable symbol in $\pi_*[1 : i]$, that is, $k = rmvs(\pi_*[1 : i])$ and $x_r = \pi_*[k]$. We assume that $w \in L^{lnr}(\pi_*, i-1)$ (depicted in the 2nd line) and $w.rep(i,\varepsilon) \in L(\pi_*)$, i.e., there exists a substitution $\varphi = \{x_1 := v_1, \ldots, x_n := v_n\}$ $(v_1, \ldots, v_n \in \Sigma^+)$ such that $\varphi(\pi_*) = w.rep(i,\varepsilon)$ holds (as depicted in the 3rd line). Then, there exists a substitution $\theta' = \{x_1 := a_1, \ldots, x_{r-1} := a_{r-1}, x_r := s''_r, x_{r+1} := s''_{r+1}, \ldots, x_n := s''_n\}$ such that $\theta'(\pi_*) = w.rep(i,\varepsilon)$ holds, where $a_1, \ldots, a_{r-1} \in \Sigma$ and $s''_r, \ldots, s''_n \in \Sigma^+$ (depicted in the bottom line).

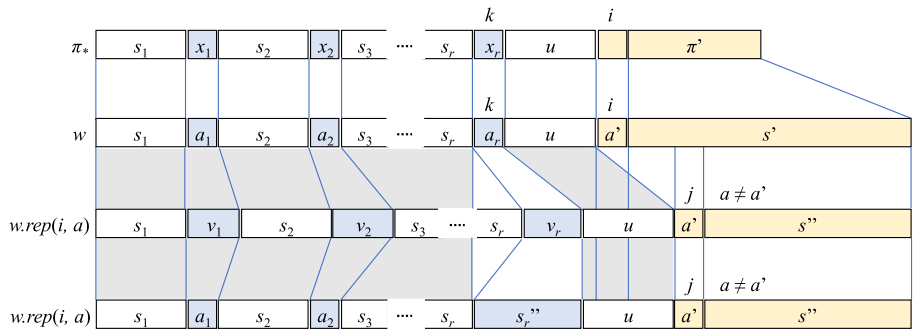

Fig. 4 Proposition 1 (2). Similarly to Proposition 1 (1) (Fig. 3), if $w \in L^{lnr}(\pi_*, i)$ and $w.rep(i,a) \in L(\pi_*)$ hold, then there exists a substitution $\theta' = \{x_1 := a_1, \ldots, x_{r-1} := a_{r-1}, x_r := s''_r\} \cup \psi$ such that $\theta'(\pi_*) = w.rep(i,a)$ holds, where ψ is a substitution for $x_{r+1}, \ldots, x_n$.

$\theta' = \{x_1 := a_1, \ldots, x_{r-1} := a_{r-1}, x_r := s''_r, x_{r+1} := s''_{r+1}, \ldots, x_n := s''_n\}$, $\theta'(\pi_*) = w.rep(i,a)$ holds (Fig. 4).

In Fig. 3, let $\pi_* = s_1x_1s_2x_2\ldots s_nx_ns_{n+1}$, r the number of variable symbols in $\pi_*[1 : i-1]$, and k the position of the rightmost variable symbol in $\pi_*[1 : i]$, that is, $k = rmvs(\pi_*[1 : i])$ and $x_r = \pi_*[k]$. We describe the regular pattern π_*, the constant string w, and the strings $w.rep(i,\varepsilon)$ with two possible substitutions. Since $w.rep(i,\varepsilon) \in L(\pi_*)$ holds, there exists a substitution φ such that $\varphi(\pi_*) = w.rep(i,\varepsilon)$ holds as shown at the third line in the figure. From the precondition of Proposition 1 (1), since the constant string w has the substitution θ such that $\theta(\pi_*) = w$ holds, we construct the new substitution θ' by combining θ and φ as shown at the forth line.

We assume that $w \in L^{lnr}(\pi_*, i-1)$ holds. If $w.rep(i,\varepsilon) \notin L(\pi_*)$, then $w \in L^{lnr}(\pi_*, i)$ holds. Otherwise, that is, $w.rep(i,\varepsilon) \in L(\pi_*)$, $w' \in L^{lnr}(\pi_*, i-1)$ and $|w'| < |w|$ hold, where w' is the constant string obtained from $w.rep(i,\varepsilon)$ by iteratively deleting the symbol at the position k of the rightmost variable in $\pi_*[1 : i]$ while the constant string after the deletions is in $L(\pi_*)$, that is, $k = rmvs(\pi_*[1 : i])$, $w' = w.rep(i,\varepsilon).\underbrace{rep(k,\varepsilon).\cdots.rep(k,\varepsilon)}_{\ell \text{ times } (\ell \geq 0)} \in L(\pi_*)$ and $w'.rep(k,\varepsilon) \notin L(\pi_*)$. By iteratively applying the above process to a constant string in $L^{lnr}(\pi_*, i-1)$, we finally get the constant string in $L^{lnr}(\pi_*, i)$. For example, we consider the case of $\pi_* = xabacy$ and $w = aabacbacc \in L^{lnr}(\pi_*, 4)$. Note that $rmvs(xabacy[1 : 5]) = rmvs(xabac) = 1$. Since $w.rep(5,\varepsilon) = aababacc \in L(\pi_*)$, we have $w.rep(5,\varepsilon).rep(1,\varepsilon).rep(1,\varepsilon) = babacc \in L^{lnr}(\pi_*, 5)$. By modifying this iteration efficiently, we propose Procedure *ShrinkString* for computing the constant string in $L^{lnr}(\pi_*, i)$. We will give the correctness of the algorithm in Lemma 1. As for Proposition 1 (2), we describe π_*, w, and $w.rep(i,a)$ with images of two possible substitutions in Fig. 4. Similarly to Proposition 1 (1), we construct the new substitution θ' that satisfies the conclusion of Proposition 1 (2).

Another important idea of our algorithm is the following proposition.

Proposition 2: We assume that $w \in L^{lnr}(\pi_*, i)$ and $w.rep(i,a) \in L(\pi_*)$ hold for a constant symbol $a \in \Sigma \setminus \{w[i]\}$, and that k is a positive integer with $1 \leq k < i$. Then, the symbol $\pi_*[i]$ is a variable symbol if and only if the constant string w' that is obtained from $w.rep(i,a)$ by deleting the k-th symbol of $w.rep(i,a)$, i.e., $w' = (w.rep(i,a)).rep(k,\varepsilon)$, is not a positive example of $L(\pi_*)$.

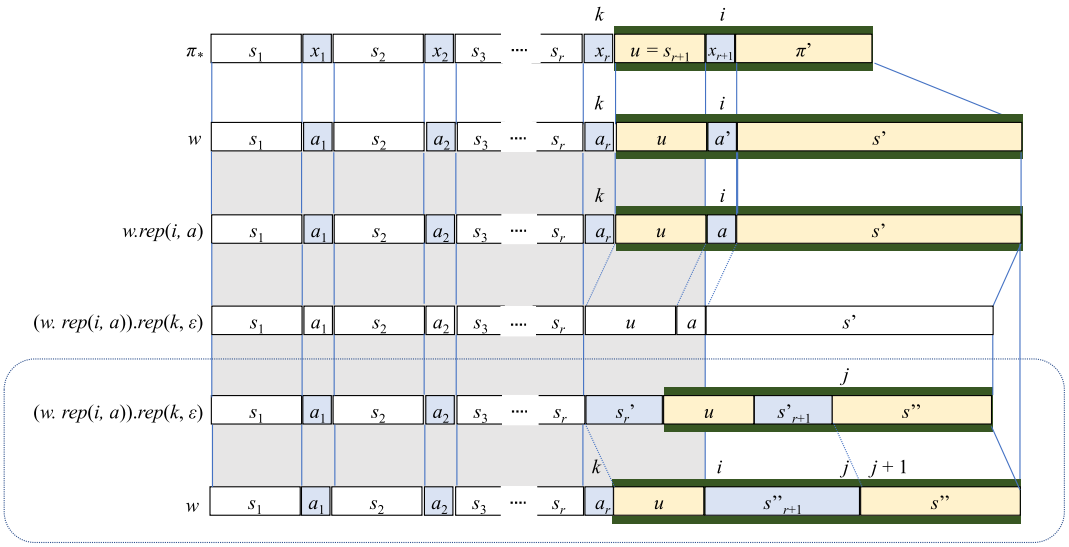

Fig. 5 The '*only if*' part of Proposition 2. We assume that $w \in L^{lnr}(\pi_*, i)$ and $w.rep(i, a) \in L(\pi_*)$ hold for $a \neq a' = w[i]$. If $\pi_*[i] \in X$ and $(w.rep(i, a)).rep(k, \varepsilon) \in L(\pi_*)$ hold, a substitution like depicted in the bottom line exists. It contradicts that $w \in L^{lnr}(\pi_*, i)$ holds.

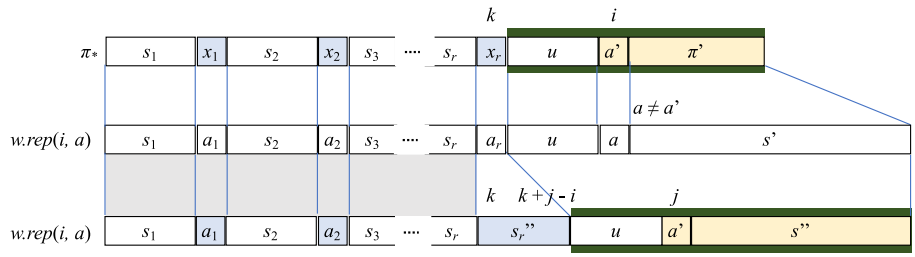

Fig. 6 The '*if*' part of Proposition 2. We assume that $w \in L^{lnr}(\pi_*, i)$ and $w.rep(i, a) \in L(\pi_*)$ hold. If $\pi_*[i] \notin X$ holds, because a is different from $w[i]$, there exists a positive integer j $(i < j)$ such that the pattern $\pi_*[i : |\pi_*|]$ matches $(w.rep(i, a))[j : |w|]$. From Proposition 1 (2), we can represent the substitution like the bottom line. Therefore, $(w.rep(i, a)).rep(k, \varepsilon) \in L(\pi_*)$ holds.

We describe the '*only if*' part and the '*if*' part in Fig. 5 and Fig. 6, respectively. When the symbol $\pi_*[i]$ is a variable symbol, i.e., $\pi_*[i] = x_{r+1}$, if $w' \in L(\pi_*)$ holds, there exists a positive integer j $(i < j)$ such that the pattern $\pi_*[i + 1 : |\pi_*|]$ matches $w[j + 1 : |w|]$. It contradicts that w is a left non-redundant positive example of $L(\pi_*)$ at i (Fig. 5). When $\pi_*[i]$ is not a variable symbol, since a is different from $w[i]$, there exists a positive integer j $(i < j)$ such that the pattern $\pi_*[i : |\pi_*|]$ matches $(w.rep(i, a))[j : |w|]$. Therefore $w' \in L(\pi_*)$ (Fig. 6). The exact proof of the proposition is given as a part of Lemma 2 together with the correctness of Procedure *IdentifyVariable* (Procedure 3).

3.3 Correctness of Algorithm *LearningStringPattern*

In this section, we show that our learning algorithm exactly identifies the target pattern by using a linear number of membership queries.

Lemma 1: Let L_* be a target in $\mathcal{RPL}$ and π_* a regular pattern in $\mathcal{RP}$ such that $L_* = L(\pi_*)$ holds. Let w be a constant string in $L(\pi_*)$, i a positive integer with $2 \leq i < |\pi_*|$, $k = rmvs(\pi_*[1 : i - 1])$ the nonnegative integer, and s the output of Procedure *ShrinkString* for inputs w, i, and k. If w is a left non-redundant positive example of $L(\pi_*)$ at $i - 1$, then the constant string s is a left non-redundant positive example of $L(\pi_*)$ at i.

Proof. The constant string w is a left non-redundant positive example of $L(\pi_*)$ at $i - 1$, i.e., there is no positive integer j with $i - 1 < j \leq |w|$ such that for some substitution φ, both $w = \varphi(\pi)$ and $w[1 : j] = \varphi(\pi[1 : i - 1])$ hold. Thus, for any substitution θ_1 with $w = \theta_1(\pi_*)$, $w[1 : i-1] = \theta_1(\pi_*[1 : i-1])$ holds.

Then, at line 1 of Procedure *ShrinkString*, if $\mathbf{MQ}(w.rep(i, \varepsilon))$ is "no", for any substitution θ' with $w = \theta'(\pi_*)$, we have $\theta'(\pi_*[i]) = w[i]$ regardless of whether $\pi_*[i]$ is a variable symbol or not. Therefore, w is a left non-redundant positive example of $L(\pi_*)$ at i. Otherwise we consider the following two cases:

1. In the case that $\pi_*[i]$ is a variable symbol: Let w_1 be the constant string obtained from w after the while-loop of lines 3–5. It is clear that $w[1 : i - 1] = w_1[1 : i - 1]$ holds. Then, $\mathbf{MQ}(w_1.rep(i, \varepsilon))$ is "no".
   We will prove that there is no positive integer j with

$$\begin{array}{lll} abbabacbacb & \Rightarrow & a\underline{b}babacbacb.rep(2,\varepsilon) = ababacbacb \in L(axabacy) \\ & \Rightarrow & a\underline{b}abacbacb.rep(2,\varepsilon) = aabacbacb \notin L(axabacy) \end{array} \Big\} \text{ The lines 1–5}$$

Fig. 7 A running example of Procedure *ShrinkString* when the constant string $w = abbabacbacb$, $i = 2$, and $k = 0$ are given as inputs. The procedure outputs the constant string *ababacbacb*.

$$\begin{array}{lll} ababacbacb & \Rightarrow & ababa\underline{c}bacb.rep(6,\varepsilon) = abababacb \in L(axabacy) \\ & \Rightarrow & ababa\underline{b}acb.rep(6,\varepsilon) = ababaacb \notin L(axabacy) \\ & \Rightarrow & a\underline{b}ababacb.rep(2,\varepsilon) = aababacb \in L(axabacy) \\ & \Rightarrow & a\underline{a}babacb.rep(2,\varepsilon) = ababacb \in L(axabacy) \\ & \Rightarrow & a\underline{b}abacb.rep(2,\varepsilon) = aabacb \notin L(axabacy) \end{array} \begin{array}{l} \Big\} \text{ The lines 1–5} \\ \Big\} \text{ The loop of lines 7–9} \end{array}$$

Fig. 8 A running example of Procedure *ShrinkString* when the constant string $w = ababacbacb$, $i = 6$, and $k = 2$ are given as inputs. The procedure outputs the constant string *ababacb*.

$i < j \leq |w_1|$ such that for some substitution θ, both $w_1 = \theta(\pi_*)$ and $w_1[1:j] = \theta(\pi_*[1:i])$ hold. Suppose that there exist a positive integer j ($i < j \leq |w_1|$) and a substitution φ_1 such that $w_1 = \varphi_1(\pi_*)$ and $w_1[1:j] = \varphi_1(\pi_*[1:i])$ hold. Since $\pi_*[i]$ is a variable symbol, from the substitutions θ_1 and φ_1, we define a new substitution ψ_1 for $\pi_* = \pi_*[1:i-1] \cdot \pi_*[i] \cdot \pi_*[i+1:|\pi_*|]$ in the following way:

$$\begin{aligned} \psi_1(\pi_*[1:i-1]) &:= \theta_1(\pi_*[1:i-1]) \\ &= w[1:i-1] \\ &= w_1[1:i-1], \\ \psi_1(\pi_*[i]) &:= w_1[i:j], \\ \psi_1(\pi_*[i+1:|\pi_*|]) &:= \varphi_1(\pi_*[i+1:|\pi_*|]) \\ &= w_1[j+1:|w_1|]. \end{aligned}$$

Then, since $\psi_1(\pi_*) = w_1$ and $|w_1[i:j]| > 1$ hold, $\mathbf{MQ}(w_1.rep(i,\varepsilon))$ must be "yes" at line 3 of Procedure *ShrinkString*. This is a contradiction. Thus w_1 is a left non-redundant positive example of $L(\pi_*)$ at i.

We will prove that $s = w_1$ holds. If $k = 0$, then $s = w_1$ obviously. Thus s is a left non-redundant positive example of $L(\pi_*)$ at i. We consider the case of $k > 0$, i.e., a variable symbol appears in $\pi_*[1:i-1]$. Note that $0 < k < i$. Suppose that $s \neq w_1$ holds. We note that from Proposition 1 (1), it is sufficient to consider the variable symbol $\pi_*[k]$ only. By the while-loop of lines 7–9, $\mathbf{MQ}(w_1.rep(k,\varepsilon))$ is "yes". Let $w_1' = w_1.rep(k,\varepsilon)$. Since $w_1' \in L(\pi_*)$ holds, there exist a positive integer ℓ ($i \leq \ell$) and a substitution φ_1' such that $w_1'[\ell+1:|w_1'|] = \varphi_1'(\pi_*[i+1:|\pi_*|])$ holds. Since $w_1'[\ell+1:|w_1'|] = w_1[\ell+2:|w_1|]$ holds, $\varphi_1'(\pi_*[i+1:|\pi_*|]) = w_1[\ell+2:|w_1|]$ holds. Then we define a new substitution ψ_1' for $\pi_* = \pi_*[1:i-1] \cdot \pi_*[i] \cdot \pi_*[i+1:|\pi_*|]$ in the following way:

$$\begin{aligned} \psi_1'(\pi_*[1:i-1]) &:= \theta_1(\pi_*[1:i-1]) \\ &= w_1[1:i-1], \\ \psi_1'(\pi_*[i]) &:= w_1[i:\ell+1], \\ \psi_1'(\pi_*[i+1:|\pi_*|]) &:= \varphi_1'(\pi_*[i+1:|\pi_*|]) \\ &= w_1[\ell+2:|w_1|]. \end{aligned}$$

Since $\psi_1'(\pi_*) = w_1$ and $|w_1[i:\ell+1]| > 1$ hold, $\mathbf{MQ}(w_1.rep(i,\varepsilon))$ must be "yes". This is a contradiction. Thus $s = w_1$ holds. Therefore s is a left non-redundant positive example of $L(\pi_*)$ at i.

2. In the case that $\pi_*[i]$ is a constant symbol: If $k = 0$, obviously w is a left non-redundant positive example of $L(\pi_*)$ at i, because $\pi_*[1:i]$ consists of constant symbols. We consider the case of $k > 0$. Let w_2 be the constant string obtained after the while-loop of lines 7–9. It is clear that $w[1:k-1] = w_1[1:k-1] = w_2[1:k-1]$ hold. Then, $\mathbf{MQ}(w_2.rep(k,\varepsilon))$ is "no".
   We will prove that w_2 is a left non-redundant positive example of $L(\pi_*)$ at i. Suppose that there exist a positive integer j ($i < j \leq |w_2|$) and a substitution φ_2 such that $w_2 = \varphi_2(\pi_*)$ and $w_2[1:j] = \varphi_2(\pi_*[1:i])$ hold. Since $\pi_*[k+1:i]$ consists of constant symbols, $w_2[k+j-i+1:j] = \varphi_2(\pi_*[k+1:i])$ holds. From the substitutions θ_1 and φ_2, we define a new substitution ψ_2 for $\pi_* = \pi_*[1:k-1] \cdot \pi_*[k] \cdot \pi_*[k+1:|\pi_*|]$ in the following way:

$$\begin{aligned} \psi_2(\pi_*[1:k-1]) &:= \theta_1(\pi_*[1:k-1]) \\ &= w_2[1:k-1], \\ \psi_2(\pi_*[k]) &:= w_2[k:k+j-i], \\ \psi_2(\pi_*[k+1:|\pi_*|]) &:= \varphi_2(\pi_*[k+1:|\pi_*|]) \\ &= w_2[k+j-i+1:|w_2|]. \end{aligned}$$

   Since $|w_2[k:k+j-i]| > 1$ holds, $\mathbf{MQ}(w_2.rep(k,\varepsilon))$ must be "yes". This is a contradiction. Therefore w_2 is a left non-redundant positive example of $L(\pi_*)$ at i. Since $s = w_2$ holds, s is a left non-redundant positive example of $L(\pi_*)$ at i.

From the above, the constant string s is a left non-redundant positive example of $L(\pi_*)$ at i. □

Example 6: Let $\pi_* = axabacy$ and $i = 2$. Then, we have $rmvs(\pi_*[1:1]) = 0$. The constant string *abbabacbacb* is a left non-redundant positive example of $L(\pi_*)$ at 1. When Procedure *ShrinkString* takes $w = abbabacbacb$, $i = 2$, and $k = 0$ as inputs, it outputs the constant string *ababacbacb* (Fig. 7). The constant string *ababacbacb* is a left non-redundant positive example of $L(\pi_*)$ at 2.

Next, in case $i = 6$, we have $rmvs(\pi_*[1:5]) = 2$. The constant string *abababacbacb* is a left non-redundant positive example of $L(\pi_*)$ at 5. When the procedure takes $w = abababacbacb$, $i = 6$, and $k = 2$ as inputs, it outputs the constant string *ababacb* (Fig. 8). The constant string

$$ababacbacb \quad \Rightarrow \quad a\underline{b}ababacbacb.rep(2,a) = aaabacbacb \in L(axabacy) \quad \text{The line 2}$$

Fig. 9 A running example of Procedure *IdentifyVariable* when the constant string $w = ababacbacb$, $i = 2$, and $k = 0$ are given as inputs. The procedure outputs a positive integer 2.

$$\begin{aligned} ababacb \quad &\Rightarrow \quad ababac\underline{b}.rep(7,a) = ababaca \in L(axabacy) \quad \text{The line 2} \\ &\Rightarrow \quad a\underline{b}abaca.rep(2,\varepsilon) = aabaca \notin L(axabacy) \quad \text{The line 5} \end{aligned}$$

Fig. 10 A running example of Procedure *IdentifyVariable* when the constant string $w = ababacbacb$, $i = 7$, and $k = 2$ are given as inputs. The procedure outputs a positive integer 7.

ababacb is a left non-redundant positive example of $L(\pi_*)$ at 6.

Lemma 2: Let L_* be a target in $\mathcal{RPL}$ and π_* a regular pattern in $\mathcal{RP}$ such that $L_* = L(\pi_*)$ holds. Let w be a constant string in $L(\pi_*)$, i a positive integer with $2 \leq i < |\pi_*|$, and $k = rmvs(\pi_*[1 : i-1])$ the nonnegative integer. If w is a left non-redundant positive example of $L(\pi_*)$ at i, then Procedure *IdentifyVariable* for inputs w, i, and k correctly computes the value of $rmvs(\pi_*[1 : i])$.

Proof. The constant string w is a left non-redundant positive example of $L(\pi_*)$ at i, i.e., there is no positive integer j with $i < j \leq |w|$ such that for some substitution θ, both $w = \theta(\pi_*)$ and $w[1 : j] = \theta(\pi_*[1 : i])$ hold. Note that for any substitution θ_1 with $w = \theta_1(\pi_*)$, $w[1 : i] = \theta_1(\pi_*[1 : i])$ holds. Then, we consider the following two cases:

1. In the case that $\pi_*[i]$ is a variable symbol: From the substitution θ_1, $\mathbf{MQ}(w.rep(i,a))$ is "yes", where $a \in \Sigma \setminus \{w[i]\}$. Let w_1 be the constant string obtained from w at line 3 of Procedure *IdentifyVariable*. It is clear that $w[1 : i-1] = w_1[1 : i-1]$ holds. If $k = 0$ holds, i.e., $rmvs(\pi_*[1 : i-1]) = 0$, obviously k is updated to be i. Otherwise $k > 0$ holds, i.e., $rmvs(\pi_*[1 : i-1]) > 0$. Suppose that $\mathbf{MQ}(w_1.rep(k,\varepsilon))$ is "yes" at line 5 of *IdentifyVariable*. Since $w_1.rep(k,\varepsilon) \in L(\pi_*)$ holds (as depicted in the fifth line of Fig. 5), there exist a positive integer j ($i < j$) and a substitution φ_1 such that $w_1[1 : j] = \varphi_1(\pi_*[1 : i])$ and $w_1 = \varphi_1(\pi_*)$ hold. Since $w[1 : i-1] = w_1[1 : i-1]$ and $w[j+1 : |w|] = w_1[j+1 : |w_1|]$ hold, we define a substitution ψ_1 for $\pi_* = \pi_*[1 : i-1] \cdot \pi_*[i] \cdot \pi_*[i+1 : |\pi_*|]$ in the following way (as depicted in the bottom line of Fig. 5):

$$\begin{aligned} \psi_1(\pi_*[1 : i-1]) &:= \theta_1(\pi_*[1 : i-1]) \\ &= w[1 : i-1], \\ \psi_1(\pi_*[i]) &:= w[i : j], \\ \psi_1(\pi_*[i+1 : |\pi_*|]) &:= \varphi_1(\pi_*[i+1 : |\pi_*|]) \\ &= w[j+1 : |w|]. \end{aligned}$$

   This contradicts that w is a left non-redundant positive example of $L(\pi_*)$ at i. Therefore $\mathbf{MQ}(w_1.rep(k,\varepsilon))$ is "no" at line 5 of *IdentifyVariable*, and then k is updated to be i.
2. In the case that $\pi_*[i]$ is a constant symbol: If $k = 0$, i.e., $rmvs(\pi_*[1 : i-1]) = 0$, then $\mathbf{MQ}(w.rep(i,a))$ is "no", and then k is not updated. We assume that $\mathbf{MQ}(w.rep(i,a))$ is "yes", where $a \in \Sigma \setminus \{w[i]\}$. Let w_2 be the constant string obtained from w at line 3 of Procedure *IdentifyVariable*. Since $a \neq w[i]$ holds, we have $a \neq \pi_*[i]$. Therefore, there exist a positive integer j ($i < j$) and a substitution φ_2 such that $w_2[1 : j] = \varphi_2(\pi_*[1 : i])$ and $w_2 = \varphi_2(\pi_*)$ hold. Since $\pi_*[i] = w[i] \neq a = w_2[i]$ holds, by using θ_1 and φ_2, we construct a substitution ψ_2 such that $w_2[1 : k-1] = \psi_2(\pi_*[1 : k-1])$, $w_2[k : k+j-i] = \psi_2(\pi_*[k])$ and $w_2 = \psi_2(\pi_*)$ hold (as depicted in the bottom line of Fig. 6). Thus $\mathbf{MQ}(w_2.rep(k,\varepsilon))$ is "yes" at line 5 of *IdentifyVariable*. Therefore k is not updated.

From the above, if $\pi_*[i]$ is a variable symbol, then k is updated. Otherwise, k is not updated. Thus, Procedure *IdentifyVariable* correctly computes the value of $rmvs(\pi_*[1 : i])$. □

Example 7: Let $\pi_* = axabacy$ and $i = 2$. Then, we have $rmvs(\pi_*[1 : 1]) = 0$. The constant string *ababacbacb* is a left non-redundant positive example of $L(\pi_*)$ at 2. When Procedure *IdentifyVariable* takes $w = ababacbacb$, $i = 2$, and $k = 0$ as inputs, it outputs a positive integer 2 (Fig. 9). It is clear that $rmvs(\pi_*[1 : 2]) = rmvs(ax) = 2$.

Next, in the case of $i = 7$, we have $rmvs(\pi_*[1 : 6]) = 2$. The constant string *ababacb* is a left non-redundant positive example of $L(\pi_*)$ at 7. When the procedure takes $w = ababacb$, $i = 7$, and $k = 2$ as inputs, it outputs a positive integer 7 (Fig. 10). It is clear that $rmvs(\pi_*[1 : 7]) = rmvs(axabacy) = 7$.

Lemma 3: Let L_* be a target in $\mathcal{RPL}$ and π_* a regular pattern in $\mathcal{RP}$ such that $L_* = L(\pi_*)$ holds. Let w_i and k_i ($1 \leq i \leq |\pi_*|$) be the constant string w and nonnegative integer k after the i-th while-loop of lines 2–9 finishes in Algorithm *LearningStringPattern*. For any positive integer i with $1 \leq i \leq |\pi_*|$, the constant string w_i is a left non-redundant positive example of $L(\pi_*)$ at i and $k_i = rmvs(\pi_*[1 : i])$ holds.

Proof. The proof is by the induction on the number of iterations $i \geq 1$ of the while-loop of lines 2–9.

We consider the case of $i = 1$. The constant string w_1 is the output of Procedure *ShrinkString* for inputs w, $i = 1$, and $k = 0$. The nonnegative integer k_1 is the output of Procedure *IdentifyVariable* for inputs w_1, $i = 1$, and $k = 0$. Thus obviously w_1 is a left non-redundant positive example of $L(\pi_*)$ at 1 and $k_1 = rmvs(\pi_*[1 : 1])$ holds.

We assume that for any i ($2 \leq i < |\pi_*|$), the statement holds after the $(i-1)$-th while-loop. By the inductive hypothesis, w_{i-1} is a left non-redundant positive example of

Table 1 A running example of Algorithm *LearningStringPattern* identifying $L(axabacy)$ when the positive example $w_0 = abbabacbacb$ is given as input.

i	**Procedure**	output	*vSet*	
1	*ShrinkString*$(abbabacbacb, 1, 0)$	$w_1 = abbabacbacb$	$\emptyset$	
	IdentifyVariable$(abbabacbacb, 1, 0)$	$k_1 = 0$	$\emptyset$	
2	*ShrinkString*$(abbabacbacb, 2, 0)$	$w_2 = ababacbacb$	$\emptyset$	(Fig. 7)
	IdentifyVariable$(ababacbacb, 2, 0)$	$k_2 = 2$	$\{2\}$	(Fig. 9)
3	*ShrinkString*$(ababacbacb, 3, 2)$	$w_3 = ababacbacb$	$\{2\}$	
	IdentifyVariable$(ababacbacb, 3, 2)$	$k_3 = 2$	$\{2\}$	
4	*ShrinkString*$(ababacbacb, 4, 2)$	$w_4 = ababacbacb$	$\{2\}$	
	IdentifyVariable$(ababacbacb, 4, 2)$	$k_4 = 2$	$\{2\}$	
5	*ShrinkString*$(ababacbacb, 5, 2)$	$w_5 = ababacbacb$	$\{2\}$	
	IdentifyVariable$(ababacbacb, 5, 2)$	$k_5 = 2$	$\{2\}$	
6	*ShrinkString*$(ababacbacb, 6, 2)$	$w_6 = ababacb$	$\{2\}$	(Fig. 8)
	IdentifyVariable$(ababacb, 6, 2)$	$k_6 = 2$	$\{2\}$	
7	*ShrinkString*$(ababacb, 7, 2)$	$w_7 = ababacb$	$\{2\}$	
	IdentifyVariable$(ababacb, 7, 2)$	$k_7 = 7$	$\{2, 7\}$	(Fig. 10)

vSet $= \{2, 7\}$	a new variable symbol	pattern
2	x_1	$ax_1abacb = ababacb.rep(2, x_1)$
7	x_2	$ax_1abacx_2 = ax_1abacb.rep(7, x_2)$

The regular pattern ax_1abacx_2 is output by Algorithm *LearningStringPattern*.

$L(\pi_*)$ at $i-1$ and $k_{i-1} = rmvs(\pi_*[1 : i-1])$ holds. Then from Lemma 1, w_i is a left non-redundant positive example of $L(\pi_*)$ at i. Since w_i is a left non-redundant positive example of $L(\pi_*)$ at i and $k_{i-1} = rmvs(\pi_*[1 : i-1])$ holds, from Lemma 2, $k_i = rmvs(\pi_*[1 : i])$ holds. Thus, for any i $(2 \leq i < |\pi_*|)$, the statement holds after the i-th while-loop.

In particular, for $i = |\pi_*| - 1$, w_i is a left non-redundant positive example of $L(\pi_*)$ at i and $k_i = rmvs(\pi_*[1 : i])$ holds. For $i = |\pi_*|$, we show that the statement holds. In the case that $\pi_*[|\pi_*|]$ is a variable symbol, obviously $|w_{|\pi_*|}| = |\pi_*|$ and $w_{|\pi_*|} \in L(\pi_*)$ hold. Thus $w_{|\pi_*|}$ is a left non-redundant positive example of $L(\pi_*)$ at $|\pi_*|$. Since $|w_{|\pi_*|}| = |\pi_*|$ and $w_{|\pi_*|} \in L(\pi_*)$ hold, we have $|\pi_*| = rmvs(\pi_*)$. In the case that $\pi_*[|\pi_*|]$ is a constant symbol, in a similar way, we show that $w_{|\pi_*|}$ is a left non-redundant positive example of $L(\pi_*)$ at $|\pi_*|$ and $k_{|\pi_*|} = rmvs(\pi_*)$ holds.

Therefore we conclude that the statement holds for all i $(1 \leq i \leq |\pi_*|)$. □

Example 8: In Table 1, when the constant string $w = abbabacbacb$ is given to Algorithm *LearningStringPattern* as a positive example, we write constant strings $w_1, \ldots, w_7$ output by Procedure *ShrinkString* and nonnegative integers $k_1, \ldots, k_7$ output by Procedure *IdentifyVariable*. At last, the constant string *ababacb* is obtained after the loop of lines 2–9 of Algorithm *LearningStringPattern*.

From Lemmas 1, 2 and 3, we prove Theorem 1 as follows.

Proof of Theorem 1. From Lemma 3, the while-loop of lines 2–9 in Algrithm *LearningStringPattern* is repeated at $|\pi_*|$ times, and $|w_{|\pi_*|}| = |\pi_*|$ holds, where $w_{|\pi_*|}$ is the constant string after the $|\pi_*|$-th while-loop of lines 2–9 finishes in Algorithm *LearningStringPattern*. From Lemmas 2 and 3, the set *vSet* in Algorithm *LearningStringPattern* equals the set $\{i \in \{1, \ldots, |\pi_*|\} \mid \pi_*[i] \in X\}$ of positive integers. Thus Algorithm *LearningStringPattern* outputs a regular pattern π with $L(\pi) = L(\pi_*)$.

At the i-th while-loop of lines 2–9, let n_i be the number of constant symbols removed in Procedure *ShrinkString*. Then, the procedure uses $n_i + 2$ membership queries. At the i-th iteration, Procedure *IdentifyVariable* uses at most two membership queries. The while-loop of lines 2–9 totally uses at most $\sum_{i=1}^{|\pi_*|}(n_i + 4)$ membership queries. Since $\sum_{i=1}^{|\pi_*|} n_i \leq |w|$ and $|\pi_*| \leq |w|$ hold,

$$\sum_{i=1}^{|\pi_*|}(n_i + 4) \leq 4 \cdot |\pi_*| + \sum_{i=1}^{|\pi_*|} n_i \leq 4 \cdot |w| + |w| = 5 \cdot |w|.$$

Thus, Algorithm *LearningStringPattern* uses $O(|w|)$ membership queries. □

4. Discussion

In this section, we discuss the efficiency of the proposed query learning algorithm *LearningStringPattern* and its applications.

4.1 Efficiency of Algorithm *LearningStringPattern*

We discuss the efficiency of *LearningStringPattern* by comparing with the previous query learning algorithm, which is presented in [5] and denoted by *prevAlgorithm*.

We briefly explain *prevAlgorithm* that exactly identifies a target regular pattern language from one positive example w using $O(|w|^2)$ membership queries. Algorithm

$$
\begin{array}{lcll}
abbabacbacb & \Rightarrow & \underline{a}bbabacbacb.rep(1,\varepsilon) & = & babacbacb \notin L(axabacy) \\
 & \Rightarrow & a\underline{b}babacbacb.rep(2,\varepsilon) & = & ababacbacb \in L(axabacy) \\
 & \Rightarrow & a\underline{b}abacbacb.rep(2,\varepsilon) & = & aabacbacb \notin L(axabacy) \\
 & \Rightarrow & ab\underline{a}bacbacb.rep(3,\varepsilon) & = & abbacbacb \notin L(axabacy) \\
 & \Rightarrow & aba\underline{b}acbacb.rep(4,\varepsilon) & = & abaacbacb \notin L(axabacy) \\
 & \Rightarrow & abab\underline{a}cbacb.rep(5,\varepsilon) & = & ababcbacb \notin L(axabacy) \\
 & \Rightarrow & ababa\underline{c}bacb.rep(6,\varepsilon) & = & abababacb \in L(axabacy) \\
 & \Rightarrow & ababa\underline{b}acb.rep(6,\varepsilon) & = & ababaacb \notin L(axabacy) \\
 & \Rightarrow & ababa\underline{a}cb.rep(7,\varepsilon) & = & ababacb \notin L(axabacy) \\
 & \Rightarrow & ababab\underline{a}cb.rep(8,\varepsilon) & = & abababab \notin L(axabacy) \\
 & \Rightarrow & abababa\underline{c}b.rep(9,\varepsilon) & = & abababac \notin L(axabacy) \\
 & \Rightarrow & abababacb
\end{array}
$$

Fig. 11 A running example of Procedure *Shrink* that is called by *prevAlgorithm* when the string $w = abbabacbacb$ is given as input. Procedure *Shrink* uses 11 (= $|w|$) membership queries and outputs the string *abababacb*, which is not a shortest positive example of π_*, where $\pi_* = axabacy$.

prevAlgorithm consists of the following two phases, called a *shrinking phase* and an *identifying phase*. In the shrinking phase, Algorithm *prevAlgorithm* makes a shortest positive example w' using Procedure *Shrink*. Procedure *Shrink* repeatedly removes $w[i]$ for all i ($1 \leq i \leq |w|$) if $w.rep(i, \varepsilon)$ is a positive example. Procedure *Shrink* uses $|w|$ membership queries. However, Procedure *Shrink* does not always output a shortest positive example. Therefore, in the shrinking phase, Algorithm *prevAlgorithm* makes a shortest positive example w' using $O(|w|^2)$ membership queries. Then, for the shortest positive example w', in the identifying phase, Algorithm *prevAlgorithm* identifies the positions at which variable symbols occur in the pattern that generates the target language using $|w'|$ membership queries. Thus, for a positive example w, Algorithm *prevAlgorithm* identifies the target pattern language using $O(|w|^2)$ membership queries.

Matsumoto and Shinohara [5] introduced a *critical regular pattern*, which is a regular pattern having a constant symbol p such that there exists a constant string w satisfying the following conditions. (1) p is a prefix of w. (2) p does not appear in $w[2 : |w|]$. (3) p appears in a constant string obtained by removing appropriate strings from $w[2 : |w|]$. For a positive example w of a critical pattern π_* as input, Procedure *Shrink* does not always output a shortest positive example of π_*.

Example 9: Since there exist the constant strings $p = abac$ and $w = abacbac$ satisfying the above three conditions, the pattern $\pi_* = axabacy$ is a critical regular pattern. For the critical regular pattern π_* and a positive example $w = abbabacbacb$, Fig. 11 shows a running example of Procedure *Shrink*. Procedure *Shrink* outputs the string *abababacb*, which is not a shortest positive example. Algorithm *prevAlgorithm* calls Procedure *Shrink* once to check whether or not w' is a shortest positive example. Then, *prevAlgorithm* calls Procedure *Shrink* three times to make $w' = ababacb$, which is a shortest positive example, from w.

If a target regular pattern is critical, to make a shortest positive example in Procedure *Shrink* of Algorithm *prevAlgorithm*, the given positive example needs to be repeatedly shrunk several times. On the other hand, Algorithm *LearningStringPattern* makes a shortest positive example by repeatedly shrinking a non-redundant positive example at the index i obtained from a given positive example w while shifting i from 1 to $|w|$. Hence, even if a target regular pattern is critical, *LearningStringPattern* makes a shortest positive example in $O(|w|)$ time. That is, *LearningStringPattern* exactly identifies any target regular pattern language more efficiently than Algorithm *prevAlgorithm*.

We implemented both algorithms *LearningStringPattern* and *prevAlgorithm* in Python and compared the numbers of membership queries of both algorithms. Experimental results show that the number of membership queries used in *LearningStringPattern* is nearly equal to that of *prevAlgorithm* except for critical regular pattern. Let π_*^1 be the critical regular pattern $xabacyabacbacdadz$. Table 2 (resp. Fig. 12) shows a running example of *LearningStringPattern* (resp. *prevAlgorithm*) that identifies the target regular pattern language $L(\pi_*^1)$ when the positive example $eabaceabacbacdadbacbacdade \in L(\pi_*^1)$ is given as input.

LearningStringPattern used 46 membership queries, but *prevAlgorithm* used 104 membership queries. The reason why the number of membership queries used in *prevAlgorithm* is larger than that of *LearningStringPattern* is that *prevAlgorithm* calls Procedure *Shrink* four times to obtain a shortest positive example. Moreover, let π_*^2 be the critical regular pattern $x_1abacx_2abacbacdadx_3abacfab acbacdadbacbacdadeaex_4$ and w the positive example $fab acfabacbacdadbacbacdadfabacfabacbacdadbacfbacdad eaebacfabacbacdadbacbacdadeaef \in L(\pi_*^2)$. Algorithm *prevAlgorithm* used 348 membership queries to exactly identify the target regular pattern language $L(\pi_*^2)$, when w is given as input (see Fig. 13). On the other hand, *LearningStringPattern* used 130 membership queries to exactly identify $L(\pi_*^2)$ when w is given as input. These experimental results indicate that, as the number of membership queries increases to obtain a shortest positive example, the difference in the number of membership queries between *LearningStringPattern* and *prevAlgorithm* increases. Hence, it is shown that *LearningStringPattern* exactly identifies more efficiently than Algorithm *prevAlgorithm*.

4.2 Practical Applications of Learning Regular Pattern Languages

Regular patterns are widely used as knowledge representa-

Table 2 A running example of Algorithm *LearningStringPattern* identifying $L(x\mathbf{abac}y\mathbf{abacbacdad}z)$ when the positive example $e\mathbf{abac}e\mathbf{abacbacdad}bacbacdade$ is given as input.

i	k	Variables	**Procedure**	w	**MQ**	Count
1	0	{}	*ShrinkString*	$\underline{e}abaceabacbacdadbacbacdade.rep(1,\varepsilon) = abaceabacbacdadbacbacdade$	*no*	1
			IdentifyVariable	$\underline{e}abaceabacbacdadbacbacdade.rep(1,a) = a\mathbf{abace}\mathbf{abacbacdad}bacbacdade$	*yes*	2
2	1	{1}	*ShrinkString*	$ea\underline{b}aceabacbacdadbacbacdade.rep(2,\varepsilon) = ebaceabacbacdadbacbacdade$	*no*	3
			IdentifyVariable	$ea\underline{b}aceabacbacdadbacbacdade.rep(2,b) = ebbaceabacbacdadbacbacdade$	*no*	4
3	1	{1}	*ShrinkString*	$eab\underline{a}ceabacbacdadbacbacdade.rep(3,\varepsilon) = eaaceabacbacdadbacbacdade$	*no*	5
			IdentifyVariable	$eab\underline{a}ceabacbacdadbacbacdade.rep(3,a) = eaaaceabacbacdadbacbacdade$	*no*	6
4	1	{1}	*ShrinkString*	$eaba\underline{c}eabacbacdadbacbacdade.rep(4,\varepsilon) = eabceabacbacdadbacbacdade$	*no*	7
			IdentifyVariable	$eaba\underline{c}eabacbacdadbacbacdade.rep(4,b) = eabbceabacbacdadbacbacdade$	*no*	8
5	1	{1}	*ShrinkString*	$eabac\underline{e}abacbacdadbacbacdade.rep(5,\varepsilon) = eabaeabacbacdadbacbacdade$	*no*	9
			IdentifyVariable	$eabac\underline{e}abacbacdadbacbacdade.rep(5,a) = eabaaeabacbacdadbacbacdade$	*no*	10
6	1	{1}	*ShrinkString*	$eabace\underline{a}bacbacdadbacbacdade.rep(6,\varepsilon) = eabacabacbacdadbacbacdade$	*no*	11
			IdentifyVariable	$eabace\underline{a}bacbacdadbacbacdade.rep(6,a) = e\mathbf{abac}a\mathbf{abacbacdad}bacbacdade$	*yes*	12
				$\underline{e}abacaabacbacdadbacbacdade.rep(1,\varepsilon) = abacaabacbacdadbacbacdade$	*no*	13
7	6	{1, 6}	*ShrinkString*	$eabaceabacbacdadbacbacdade.rep(7,\varepsilon) = eabacebacbacdadbacbacdade$	*no*	14
			IdentifyVariable	$eabace\underline{a}bacbacdadbacbacdade.rep(7,b) = eabacebbacbacdadbacbacdade$	*no*	15
8	6	{1, 6}	*ShrinkString*	$eabaceab\underline{a}cbacdadbacbacdade.rep(8,\varepsilon) = eabaceaacbacdadbacbacdade$	*no*	16
			IdentifyVariable	$eabaceab\underline{a}cbacdadbacbacdade.rep(8,a) = eabaceaaacbacdadbacbacdade$	*no*	17
9	6	{1, 6}	*ShrinkString*	$eabaceaba\underline{c}bacdadbacbacdade.rep(9,\varepsilon) = eabaceabcbacdadbacbacdade$	*no*	18
			IdentifyVariable	$eabaceaba\underline{c}bacdadbacbacdade.rep(9,b) = eabaceabbcbacdadbacbacdade$	*no*	19
10	6	{1, 6}	*ShrinkString*	$eabaceabac\underline{b}acdadbacbacdade.rep(10,\varepsilon) = eabaceababacdadbacbacdade$	*no*	20
			IdentifyVariable	$eabaceabac\underline{b}acdadbacbacdade.rep(10,a) = eabaceabaabacdadbacbacdade$	*no*	21
11	6	{1, 6}	*ShrinkString*	$eabaceabacb\underline{a}cdadbacbacdade.rep(11,\varepsilon) = eabaceabacacdadbacbacdade$	*no*	22
			IdentifyVariable	$eabaceabacb\underline{a}cdadbacbacdade.rep(11,a) = eabaceabacaacdadbacbacdade$	*no*	23
12	6	{1, 6}	*ShrinkString*	$eabaceabacba\underline{c}dadbacbacdade.rep(12,\varepsilon) = eabaceabacbcdadbacbacdade$	*no*	24
			IdentifyVariable	$eabaceabacba\underline{c}dadbacbacdade.rep(12,b) = eabaceabacbbcdadbacbacdade$	*no*	25
13	6	{1, 6}	*ShrinkString*	$eabaceabacbac\underline{d}adbacbacdade.rep(13,\varepsilon) = eabaceabacbadadbacbacdade$	*no*	26
			IdentifyVariable	$eabaceabacbac\underline{d}adbacbacdade.rep(13,a) = eabaceabacbaadadbacbacdade$	*no*	27
14	6	{1, 6}	*ShrinkString*	$eabaceabacbacd\underline{a}dbacbacdade.rep(14,\varepsilon) = eabaceabacbacadbacbacdade$	*no*	28
			IdentifyVariable	$eabaceabacbacd\underline{a}dbacbacdade.rep(14,a) = eabaceabacbacaadbacbacdade$	*no*	29
15	6	{1, 6}	*ShrinkString*	$eabaceabacbacda\underline{d}bacbacdade.rep(15,\varepsilon) = eabaceabacbacddbacbacdade$	*no*	30
			IdentifyVariable	$eabaceabacbacda\underline{d}bacbacdade.rep(15,b) = eabaceabacbacdbdbacbacdade$	*no*	31
16	6	{1, 6}	*ShrinkString*	$eabaceabacbacda\underline{d}bacbacdade.rep(16,\varepsilon) = e\mathbf{abac}eabacbacd\mathbf{abacbacdad}e$	*yes*	32
				$eabaceabacbacdab\underline{a}cbacdade.rep(16,\varepsilon) = eabaceabacbacdaacbacdade$	*no*	33
				$eabace\underline{a}bacbacdabacbacdade.rep(6,\varepsilon) = e\mathbf{abac}abacbacd\mathbf{abacbacdad}e$	*yes*	34
				$eabac\underline{a}bacbacdabacbacdade.rep(6,\varepsilon) = e\mathbf{abac}bacbacd\mathbf{abacbacdad}e$	*yes*	35
				$eabac\underline{b}acbacdabacbacdade.rep(6,\varepsilon) = e\mathbf{abac}acbacd\mathbf{abacbacdad}e$	*yes*	36
				$eabaca\underline{c}bacdabacbacdade.rep(6,\varepsilon) = e\mathbf{abac}cbacd\mathbf{abacbacdad}e$	*yes*	37
				$eabacc\underline{b}acdabacbacdade.rep(6,\varepsilon) = e\mathbf{abac}bacd\mathbf{abacbacdad}e$	*yes*	38
				$eabacb\underline{a}cdabacbacdade.rep(6,\varepsilon) = e\mathbf{abac}acd\mathbf{abacbacdad}e$	*yes*	39
				$eabaca\underline{c}dabacbacdade.rep(6,\varepsilon) = e\mathbf{abac}cd\mathbf{abacbacdad}e$	*yes*	40
				$eabacc\underline{d}abacbacdade.rep(6,\varepsilon) = e\mathbf{abac}d\mathbf{abacbacdad}e$	*yes*	41
				$eabacd\underline{a}bacbacdade.rep(6,\varepsilon) = eabacabacbacdade$	*no*	42
			IdentifyVariable	$eabacdabacbacda\underline{d}e.rep(16,a) = eabacdabacbacdaae$	*no*	43
17	6	{1, 6}	*ShrinkString*	$eabacdabacbacdad\underline{e}.rep(17,\varepsilon) = eabacdabacbacdad$	*no*	44
			IdentifyVariable	$eabacdabacbacdad\underline{e}.rep(17,a) = e\mathbf{abac}d\mathbf{abacbacdad}a$	*yes*	45
				$eabacd\underline{a}bacbacdade.rep(6,\varepsilon) = eabacabacbacdade$	*no*	46
		{1, 6, 17}		$x_1\mathbf{abac}x_2\mathbf{abacbacdad}x_3$		

			The number of membership queries
$e\mathbf{abac}e\mathbf{abacbacdad}bacbacdade$	$\stackrel{*}{\Longrightarrow}$	$e\mathbf{abac}eabacbacd\mathbf{abacbacdad}e$	26
	$\stackrel{*}{\Longrightarrow}$	$eab\mathbf{abac}d\mathbf{abacbacdad}e$	51
	$\stackrel{*}{\Longrightarrow}$	$b\mathbf{abac}d\mathbf{abacbacdad}e$	70
	$\stackrel{*}{\Longrightarrow}$	$b\mathbf{abac}d\mathbf{abacbacdad}e$	87
	$\stackrel{*}{\Longrightarrow}$	$x_1\mathbf{abac}x_2\mathbf{abacbacdad}x_3$	104

Fig. 12 A running example of Algorithm *PrevAlgorithm* identifying $L(x\mathbf{abac}y\mathbf{abacbacdad}z)$ when the positive example $e\mathbf{abac}e\mathbf{abacbacdad}bacbacdade$ is given as input. The notation $\stackrel{*}{\Longrightarrow}$ shows the flow of some positive examples made in this algorithm. Constant strings in bold face are constant strings appearing in the regular pattern.

tions in various fields. For example, a set of regular patterns in k-mmg [16] and a decision tree on regular patterns in machine learning systems BONSAI [17] and BONSAI Garden [18] were used as knowledge representations acquired from Protein Information Resource (PIR) [20] and NIH genetic sequence database GenBank [19]. Moreover, a

	The number of membership queries
$f\mathbf{abac}f\mathbf{abacbacdad}bacbacdadf\mathbf{abacfabacbacdadbacbacdadeae}bacfabacbacdadbacbacdadeaef$	
$\stackrel{*}{\Longrightarrow} fabacfabacbacdabacbacdadf\mathbf{abac}f\mathbf{abacbacdad}bacbacdade\mathbf{abacfabacbacdadbacbacdadeae}f$	80
$\stackrel{*}{\Longrightarrow} fabacf\mathbf{abac}bacd\mathbf{abacbacdad}e\mathbf{abacfabacbacdadbacbacdadeae}f$	158
$\stackrel{*}{\Longrightarrow} fab\mathbf{abac}d\mathbf{abacbacdad}e\mathbf{abacfabacbacdadbacbacdadeae}f$	211
$\stackrel{*}{\Longrightarrow} b\mathbf{abac}d\mathbf{abacbacdad}e\mathbf{abacfabacbacdadbacbacdadeae}f$	258
$\stackrel{*}{\Longrightarrow} b\mathbf{abac}d\mathbf{abacbacdad}e\mathbf{abacfabacbacdadbacbacdadeae}f$	303
$\stackrel{*}{\Longrightarrow} x_1\mathbf{abac}x_2\mathbf{abacbacdad}x_3\mathbf{abacfabacbacdadbacbacdadeae}x_4$	348

Fig. 13 A running example of Algorithm *PrevAlgorithm* identifying $L(x_1\mathbf{abac}x_2\mathbf{abacbacdad}x_3\mathbf{abacfabacbacdadbacbacdadeae}x_4)$ when the positive example $f\mathbf{abac}f\mathbf{abacbacdad}bacbacdadf\mathbf{abacfabacbacdadbacbacdadeae}bacfabacbacdadbacbacdadeaef$ is given as input.

phrase-based pattern [21], which is formulated by a regular pattern, was proposed for text categorization and sentiment analysis.

Active learning, which is a framework introduced by Angluin [7], applies to interactive data mining tools [22]–[24], quantum algorithms [25]–[27] and so on. Angluin showed that the class of regular languages, each of which is characterized by the set of strings accepted by a deterministic finite automaton, is learned from a Minimally Adequate Teacher (MAT) answering membership and equivalence queries. For a membership query with respect to a string w, the teacher answers "yes" if w is the target language, but "no" if not. For an equivalence query with respect to a candidate language L, the teacher answers "yes" if L equals the target Language L_*, but provides a counterexample, which is a string w_* from the symmetric difference of L_* and L, i.e., $w_* \in (L \setminus L_*) \cup (L_* \setminus L)$, if not. Active automata learning [7] can be considered to be a key technology for dealing with black-box systems, which can be observed externally but no or little knowledge about the internal workings of which is available. In active automata learning, observations for block-box systems can be realized by membership queries and equivalence queries. For the above considered application contexts, membership queries may often be realized via testing in practice, but equivalence queries are usually unrealistic. Therefore, we use only membership queries to identify a target language. Since it is known [4] that a target language cannot be identify by only membership queries, we adopt one positive example and membership query in the scenario to identify a target language. In active learning systems, minimizing the number of required membership queries is the key to learning efficiency. Since a regular pattern language is a regular language, our results may make many practical applications based on active automata learning more efficient. The class of pattern languages and that of regular languages are incomparable, because a pattern language generated by a pattern having variable symbols occurred twice or more is not a regular language. Therefore, if our result can be expanded to pattern languages, an efficient active learning system that learns more expressive knowledge may be designed.

5. Conclusion

In this paper, by providing a query learning algorithm for regular pattern languages, we have proved that the class of regular pattern languages is exactly learnable from only one positive example using a linear number of membership queries. This result shows that the number of membership queries is reduced to be linear with respect to the length of the positive example. We implemented the proposed algorithm and evaluated its efficiency.

We introduced a term tree pattern, which is a rooted ordered tree pattern that consists of ordered tree structures with edge labels and structured variables with labels, in [28]. Moreover, we introduced a primitive formal ordered tree system (pFOTS) as a formal system defining ordered tree languages [12]. For a pFOTS program Γ as background knowledge, we showed in [12] that the class of tree languages derived using Γ and one primitive graph rewriting rule is exactly learnable from one positive example using a polynomial number of membership queries. As future work, we will consider query learning algorithms for exactly identifying the class of tree languages generated by term tree patterns and the class of tree languages generated by from pFOTSs as background knowledge and primitive graph rewriting rules, from one positive example using a linear number of membership queries with respect to the number of edges of the positive example. As applications of the proposed algorithm, we are planning to design efficient interactive data mining tools acquiring knowledge that can be modeled using regular patterns from real-world databases.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable and helpful comments on our manuscript. This work was partially supported by Grant-in-Aid for Scientific Research (C) (Grant Numbers 15K00312, 15K00313, 17K00321) from Japan Society for the Promotion of Science (JSPS).

References

[1] D. Angluin, "Queries and concept learning," Machine Learning,

vol.2, no.4, pp.319–342, 1988.
[2] H. Mamitsuka and N. Abe, "Efficient mining from large databases by query learning," Proc. 7th International Conference on Machine Learning (ICML'00), pp.575–582, 2000.
[3] D. Angluin, "Finding patterns common to a set of strings," Journal of Computer and System Sciences, vol.21, no.1, pp.46–62, 1980.
[4] A. Marron, "Learning pattern languages from a single initial example and from queries," Proc. Workshop on Computational Learning Theory (COLT'88), pp.345–358, 1988.
[5] S. Matsumoto and A. Shinohara, "Learning pattern languages using queries," Proc. European Conf. on Computational Learning Theory (EuroCOLT'97), LNAI, vol.1208, pp.185–197, Springer, Berlin, 1997.
[6] S. Lange and R. Wiehagen, "Polynomial-time inference of arbitrary pattern languages," New Generation Computing, vol.8, no.4, pp.361–370, 1991.
[7] D. Angluin, "Learning regular sets from queries and counterexamples," Information and Computation, vol.75, no.2, pp.87–106, 1987.
[8] J. Nessel and S. Lange, "Learning erasing pattern languages with queries," Theoretical Computer Science, vol.348, no.1, pp.41–57, 2005.
[9] H. Sakamoto, K. Hirata, and H. Arimura, "Learning elementary formal systems with queries," Theoretical Computer Science, vol.298, no.1, pp.21–50, 2003.
[10] H. Kato, S. Matsumoto, and T. Miyahara, "Learning of elementary formal systems with two clauses using queries," IEICE Trans. Inf. & Syst., vol.E92-D, no.2, pp.172–180, 2009.
[11] S. Matsumoto, T. Shoudai, T. Uchida, T. Miyahara, and Y. Suzuki, "Learning of finite unions of tree patterns with internal structured variables from queries," IEICE Trans. Inf. & Syst., vol.E91-D, no.2, pp.222–230, 2008.
[12] T. Uchida, S. Matsumoto, T. Shoudai, Y. Suzuki, and T. Miyahara, "Exact learning of primitive formal system defining labeled ordered tree languages via queries," IEICE Trans. Inf. & Syst., vol.E101-D, no.3, pp.470–482, 2019.
[13] A. Nakamura, "An efficient query learning algorithm for ordered binary decision diagrams," Information and Computation, vol.201, no.2, pp.178–198, 2005.
[14] H. Mizumoto, S. Todoroki, Diptarama, R. Yoshinaka, and A. Shinohara, "An efficient query learning algorithm for zero-suppressed binary decision diagrams," Proc. Machine Learning Research (ALT2017), vol.76, pp.1–12, 2017.
[15] S. Matsumoto, T. Uchida, T. Shoudai, Y. Suzuki, and T. Miyahara, "Exact learning of regular pattern languages from one positive example using a linear number of membership queries," Proc. 27th International MultiConference of Engineers and Computer Scientists (IMECS2019), pp.204–209, 2019.
[16] H. Arimura, R. Fujino, T. Shinohara, and S. Arikawa, "Protein motif discovery from positive examples by minimal multiple generalization over regular patterns," Genome Informatics, vol.5, pp.39–48, 1994.
[17] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, and S. Arikawa, "Knowledge acquisition from amino acid sequences by machine learning system BONSAI," Transactions of Information Processing Society of Japan, vol.35, no.10, pp.2009–2018, 1994.
[18] T. Shoudai, M. Lappe, S. Miyano, A. Shinohara, T. Okazaki, S. Arikawa, T. Uchida, S. Shimozono, T. Shinohara, and S. Kuhara, "BONSAI Garden: parallel knowledge discovery system for amino acid sequences," Proc. Third International Conference on Intelligent Systems for Molecular Biology, pp.359–366, 1995.
[19] National Center for Biotechnology Information, NIH genetic sequence database. https://www.ncbi.nlm.nih.gov/genbank/, Sept. 2019.
[20] Protein Information Resource (PIR). https://proteininformationresource.org/pirwww/, Sept. 2019.
[21] K. Hattori, H. Yokono, and A. Aizawa, "Phrase pattern generation for text classification," Proc. 29th Annual Conference of the Japanese Society for Artificial Intelligence, 2E1-4in, 2015 (in Japanese).
[22] B. Settles, "From theories to queries: active learning in practice," Proc. Active Learning and Experimental Design Workshop in conjunction with AISTATS 2010, PMLR, pp.11–18, 2011.
[23] F. Howar and B. Steffen, "Active automata learning in practice - an annotated bibliography of the years 2011 to 2016," Proc. Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, LNCS 11026, pp.123–148, Springer, 2018.
[24] International Community interested in Grammatical Inference, https://grammarlearning.org/Active_Learning, Sept. 2019.
[25] R.A. Servedio and S.J. Gortler, "Quantum versus classical learnability," Proc. 16th Annual IEEE Conference on Computational Complexity (CoCo2001), pp.138–148, 2001.
[26] Quantum Algorithm Zoo, https://quantumalgorithmzoo.org, Sept. 2019.
[27] S. Aaronson, D. Grier, and L. Schaeffer, "A quantum query complexity trichotomy for regular languages," Electronic Colloquium on Computational Complexity (ECCC), 26:61, 2019.
[28] Y. Suzuki, T. Shoudai, T. Uchida, and T. Miyahara, "Ordered term tree languages which are polynomial time inductively inferable from positive data," Theoretical Computer Science, vol.350, no.1, pp.63–90, 2006.

Satoshi Matsumoto is an associate professor of Department of Mathematical Sciences, Tokai University, Kanagawa, Japan. He received the B.S. degree in Mathematics, the M.S. and Dr. Sci. degrees in Information Systems all from Kyushu University, Fukuoka, Japan in 1993, 1995 and 1998, respectively. His research interests include algorithmic learning theory.

Tomoyuki Uchida received the B.S. degree in Mathematics, the M.S. and Dr. Sci. degrees in Information Systems all from Kyushu University, in 1989, 1991 and 1994, respectively. Currently, he is an associate professor of Graduate School of Information Sciences, Hiroshima City University. His research interests include data mining from semistructured data, algorithmic graph theory and algorithmic learning theory.

Takayoshi Shoudai received the B.S. in 1986, the M.S. degree in 1988 in Mathematics and the Dr. Sci. in 1993 in Information Science all from Kyushu University. Currently, he is a professor of Faculty of Contemporary Business, Kyushu International University. His research interests include graph algorithms, computational learning theory, and data mining.

Yusuke Suzuki received the B.S. degree in Physics, the M.S. and Dr. Sci. degrees in Informatics all from Kyushu University, in 2000, 2002 and 2007, respectively. He is currently a research associate of Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. His research interests include machine learning and data mining.

Tetsuhiro Miyahara is an associate professor of Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. He received the B.S. degree in Mathematics, the M.S. and Dr. Sci. degrees in Information Systems all from Kyushu University, Fukuoka, Japan in 1984, 1986 and 1996, respectively. His research interests include algorithmic learning theory, knowledge discovery and machine learning.