Exact Learning of Regular Pattern Languages from One Positive Example Using a Linear Number of Membership Queries

Satoshi Matsumoto, Tomoyuki Uchida, *Member, IAENG,* Takayoshi Shoudai, Yusuke Suzuki and Tetsuhiro Miyahara

***Abstract*—A query learning model is an established mathematical model of learning via queries in computational learning theory. 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 constant strings. In a query learning model, it is known that the class of languages of regular patterns is identifiable from one positive example using a polynomial number of membership queries. In this paper, we show that the class is identifiable from one positive example using a linear number of membership queries. This result means that the number of membership queries is reduced to be linear with respect to the length of the positive example.**

***Index Terms*—pattern language, membership query, query learning, computational learning theory**

I. Introduction

The query learning model of Angluin [1] is an established mathematical model of learning via queries in computational learning theory. In this learning model, a learning algorithm accesses oracles, which answer specific types of queries, and collects information about a target. The query learning model is regarded as a mathematical model of a data mining strategy using queries for large databases (e.g.,[2]). Querying whether or not a target 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 contained in a target.

Angluin [3] presented a *pattern* as a string consisting of constant symbols and variable symbols. Particularly, a pattern π is said to be *regular* if each variable symbol occurs in π at most once. For example, let $\pi_1 = xaybaz$ and $\pi_2 = xaybyz$ be two strings consisting of two constant symbols a, b and variable symbols x, y, z. The string π_1 is a regular pattern but π_2 is not regular. 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 strings consisting of constant symbols. There is considerable research [1], [4], [5], [6] about learning algorithms for identifying a target pattern language 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 to query 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 string that belongs to the target language before starting the process of asking queries. Matsumoto and Shinohara [5] showed a query learning algorithm for identifying a target pattern language from one positive example w using a polynomial number of membership queries with respect to the length of w. Moreover, they presented a non-trivial subclass of regular pattern languages that is identified from one positive example using a linear number of membership queries.

In this paper, we show that the full class of regular pattern languages is identified from one positive example using a linear number of membership queries. The proposed query learning algorithm consists of the following two phases, called ShrinkString and IdentifyVariables, while shifting the index in a given positive example from the start to the end of it. The ShrinkString phase is to recursively shrink the given positive example by asking membership queries, until it is confirmed that the resulting example is a shortest one. The IdentifyVariables phase is to determine whether or not the target pattern has a variable symbol at the index by asking membership queries. By shrinking constant symbols at the position of the rightmost variable symbol in the substring up to the index in both the ShrinkString and IdentifyVariables phases, we update a shortest example. From this algorithm, we expand a target class that is exactly learnable from one positive example using a linear number of membership queries to the class of regular pattern languages. That is, this result shows that a target 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 [5].

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

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).

S. Matsumoto is with the Faculty of Science, Tokai University, 4-1-1 Kitakaname, Hiratsuka, Japan, e-mail: matsumoto@tsc.u-tokai.ac.jp.
T. Uchida, Y. Suzuki and T. Miyahara are with Hiroshima City University, e-mail:{uchida, y-suzuki, miyares19}@hiroshima-cu.ac.jp.
T. Shoudai is with Kyushu International University, e-mail: shoudai@cb.kiu.ac.jp.

This paper is organized as follows. In Sec. II, we introduce a regular pattern and its language. Moreover, we briefly introduce the query learning model proposed by Angluin [1]. In Sec. III, by presenting a query learning algorithm, we show that the class of regular pattern languages is exactly learnable from only one positive example using a linear number of membership queries. We conclude this paper and give future work in Sec. IV.

II. Preliminaries

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

A. 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 $\Sigma \cup X$. 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 $(\Sigma \cup X)^+$ 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. Namely, for a pattern π, $\theta(\pi)$ is a pattern obtained from π by replacing variable symbols with patterns according to θ. For a pattern π, the *pattern language* of π, denoted by $L(\pi)$, is the set of all strings w in Σ^+ such that w is obtained from π by replacing all variable symbols in π with strings in Σ^+, that is, $L(\pi) = \{w \in \Sigma^+ \mid w = \theta(\pi) \text{ for some substitution } \theta\}$. We define $\mathcal{RPL} = \{L(\pi) \mid \pi \in \mathcal{RP}\}$.

Example 1: Let $\Sigma = \{a, b\}$ and $X = \{x, y, z, \dots\}$. Let $\pi_1 = abxby$ be a regular pattern on $\Sigma \cup X$. Then, we have $L(\pi_1) = \{ababa, ababb, abbba, abbbb, abaaba, ababba, \dots\}$.

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 a symbol in $\Sigma \cup X$ or the empty string ε 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.

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

If a pattern π contains a variable symbol, we denote by $rmvs(\pi)$ the index of the rightmost variable in π, that is, $rmvs(\pi) = \max\{i \mid \pi[i] \text{ is a variable symbol}\}$. Otherwise, we define $rmvs(\pi) = 0$. Note that $0 \leq rmvs(\pi) \leq |\pi|$.

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

B. Learning Model

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

Let $L_* \in \mathcal{L}$ be a target. Let $r_* \in R$ be one of the representations of L_*. i.e., $L_* = L(r_*)$. A 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 string $w \in \Sigma^+$. The output is "yes" if $w \in L(r_*)$ holds and "no" otherwise. We denote by **MQ** the oracle that answers membership queries. For a target L_* and a 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 $abxby$ be a target regular pattern. $\mathbf{MQ}(abaaba)$ 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 $L(r) = L(r_*)$. In the next section, we will present a learning algorithm that exactly identifies any target $L_* \in \mathcal{RPL}$ using only one positive example and a linear number of membership queries, that is, the algorithm outputs a regular pattern $\pi \in \mathcal{RP}$ satisfying $L(\pi) = L_*$.

III. Learning Algorithm

We consider the case that Σ is a set of one constant symbol. Let L_* be a target regular pattern language. The shortest positive example is obtained from a given positive example by asking a linear number of membership queries. Then, we can identify a regular pattern π such that $L(\pi) = L_*$ holds, by replacing the first constant symbol of the shortest positive example with a variable symbol. Thus, we assume that Σ is a set of two or more constant symbols. Let π_* be a target regular pattern, that is, $L(\pi_*) = L_*$.

Let π be a regular pattern, w a string in $L(\pi)$ and i an integer with $1 \leq i \leq |\pi|$. We say that a triple (π, w, i) satisfies Condition 1 if the following conditions hold: (1) there exists a substitution θ_i such that $w = \theta_i(\pi)$ and $w[1:i] = \theta_i(\pi[1:i])$ hold, and (2) for each integer j with $i < j \leq |w|$, there exists no substitution θ_j such that $w = \theta_j(\pi)$ and $w[1:j] = \theta_j(\pi[1:i])$ hold.

Example 5: For the regular pattern $\pi_4 = axybaza$, the triplet $(\pi_4, ababababbba, 5)$ satisfies Condition 1 (see Fig. 1). But the triplet $(\pi_4, abababbbba, 6)$ does not satisfy Condition 1 (see Fig. 2).

Lemma 1: Let w be a string in $L(\pi_*)$. Let i be a positive integer with $2 \leq i \leq |\pi_*|$. Let k be a nonnegative integer

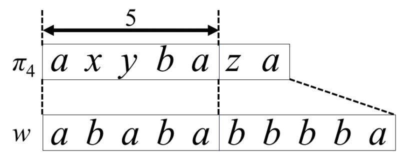

Fig. 1. The triplet $(\pi_4, ababababbba, 5)$ satisfies Condition 1. There exists a substitution θ such that $ababa = \theta(\pi_4[1:5])$ and $ababababbba = \theta(\pi_4)$. And for each integer j with $5 < j \le 10$, there exists no substitution θ_j such that $w[1:j] = \theta_j(\pi_4[1:5])$ and $w = \theta_j(\pi_4)$, where $w = ababababbba$.

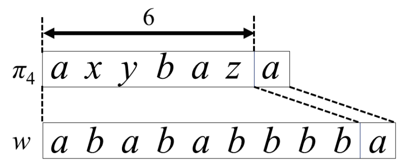

Fig. 2. The triplet $(\pi_4, ababababbba, 6)$ does not satisfy (2) of Condition 1, since there exists a substitution θ such that $ababababbb = \theta(\pi_4[1:6])$ and $ababababbba = \theta(\pi_4)$ hold.

with $0 \le k \le |\pi_*|$. We assume that $(\pi_*, w, i-1)$ satisfies Condition 1, and $k = rmvs(\pi_*[1:i-1])$. Let s be the output of Procedure *ShrinkString* when w, i, and k are given as input. Then the triplet (π_*, s, i) satisfies Condition 1.

Proof: By the assumption, the triplet $(\pi_*, w, i-1)$ satisfies Condition 1. There exists a substitution θ_{i-1} such that $w[1:i-1] = \theta_{i-1}(\pi_*[1:i-1])$ and $w = \theta_{i-1}(\pi_*)$. For each integer j with $i \le j \le |w|$, there exists no substitution θ'_j such that $w[1:j] = \theta'_j(\pi_*[1:i-1])$ and $w = \theta'_j(\pi_*)$. We consider the following two cases.

1) In the case that $\pi_*[i]$ is a variable symbol. Let w_1 be the string obtained from w after the loop of lines 1–3. Then, $\mathbf{MQ}(w_1.rep(i,\varepsilon))$ is "no". We show that for each integer j with $i < j \le |w_1|$, there exists no substitution $\theta'_{1,j}$ such that $w_1[1:j] = \theta'_{1,j}(\pi_*[1:i])$ and $w_1 = \theta'_{1,j}(\pi_*)$. Suppose that there exists a substitution $\theta'_{1,j}$ such that $w_1[1:j] = \theta'_{1,j}(\pi_*[1:i])$ and $w_1 = \theta'_{1,j}(\pi_*)$ for some integer j with $i < j \le |w_1|$. Since $\pi_*[i]$ is a variable symbol, by the substitutions θ_{i-1} and $\theta'_{1,j}$, there exists a substitution θ'_1 such that $w_1[1:i-1] = \theta'_1(\pi_*[1:i-1])$, $w_1[i:j] = \theta'_1(\pi_*[i])$ and $w_1 = \theta'_1(\pi_*)$. Since $|w_1[i:j]| > 1$, $\mathbf{MQ}(w_1.rep(i,\varepsilon))$ is "yes". This is a contradiction. This satisfies (2) of Condition 1. Since $w_1 \in L(\pi_*)$, there exists a substitution θ_1 such that $w_1[1:i] = \theta_1(\pi_*[1:i])$ and $w_1 = \theta_1(\pi_*)$. This satisfies (1) of Condition 1. Therefore, the triplet (π_*, w_1, i) satisfies Condition 1.
In the case of $k = 0$, it is clear that $s = w_1$. Thus, The triplet $(\pi_*, s, 1)$ satisfies Condition 1. We consider the case of $k > 0$, that is, a variable symbol appears in $\pi_*[1:i-1]$. Note that $0 < k < i$. Let w_2 be the string obtained after the loop of lines 5–7. Then, we show that w_1 and w_2 are the same. We assume that w_2 is different from w_1. By the loop of lines 5–7, $\mathbf{MQ}(w_1.rep(k,\varepsilon))$ is "yes". Let $w'_1 = w_1.rep(k,\varepsilon)$. Since $w'_1 \in L(\pi_*)$, there exists a substitution θ'_1 such that $w'_1[\ell : |w'_1|] = \theta'_1(\pi_*[i:|\pi_*|])$ for some integer ℓ with $i \le \ell$. Since $w'_1[\ell : |w'_1|] = w_1[\ell+1 : |w_1|]$, there exists a substitution θ''_1 such that $w_1[1:i-1] = \theta''_1(\pi_*[1:i-1])$, $w_1[i:\ell+1] = \theta''_1(\pi_*[i])$ and $w_1 = \theta''_1(\pi_*)$. Therefore, we have $|w_1[i:\ell+1]| > 1$. $\mathbf{MQ}(w_1.rep(i,\varepsilon))$ is "yes". This is a contradiction. Thus, w_1 and w_2 are the same. The triplet (π_*, w_2, i) satisfies Condition 1. Since $s = w_2$, the triplet (π_*, s, i) satisfies Condition 1.

2) In the case that $\pi_*[i]$ is a constant symbol. At first, we consider the case $k = 0$. Since $\pi_*[1:i]$ consists of constant symbols, it is clear that (π_*, w, i) satisfies Condition 1. Next, we consider the case $k > 0$. Let w_2 be the string obtained after the loop of lines 5–7. $\mathbf{MQ}(w_2.rep(k,\varepsilon))$ is "no". We assume that there exists a substitution $\theta'_{2,j}$ such that $w_2[1:j] = \theta'_{2,j}(\pi_*[1:i])$ and $w_2 = \theta'_{2,j}(\pi_*)$ for some integer j with $i < j \le |w_2|$. Since $\pi_*[i]$ is a constant symbol, by the substitution θ_{i-1} and $\theta'_{2,j}$, there exists a substitution θ'_2 such that $w_2[1:k-1] = \theta'_2(\pi_*[1:k-1])$, $w_2[k:m] = \theta'_2(\pi_*[k])$ and $w_2 = \theta'_2(\pi_*)$ for some integer $m > k$. Since $|w_2[k:m]| > 1$, $\mathbf{MQ}(w_2.rep(k,\varepsilon))$ is "yes". This is a contradiction. Thus, for each integer j with $i < j \le |w_2|$, there exists no substitution $\theta_{2,j}$ such that $w_2[1:j] = \theta_{2,j}(\pi_*[1:i])$ and $w_2 = \theta_{2,j}(\pi_*)$. This satisfies (2) of Condition 1. Since $w_2 \in L(\pi_*)$, there exists a substitution θ_2 such that $w_2[1:i] = \theta_2(\pi_*[1:i])$ and $w_2 = \theta_2(\pi_*)$. This satisfies (1) of Condition 1. The triplet (π_*, w_2, i) satisfies Condition 1. Since $s = w_2$, the triplet (π_*, s, i) satisfies Condition 1.

From the above, the triplet (π_*, s, i) satisfies Condition 1. ■

Procedure 1 *ShrinkString*

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

Example 6: Let $\pi_* = axabacy$ and $i = 2$. Then, we have $rmvs(\pi_*[1]) = 0$. Procedure *ShrinkString* outputs a string $ababacbacb$ when it takes $w = abbabacbacb$, $i = 2$, and $k = 0$ as input (see Fig. 3). In case $i = 6$, we have $rmvs(\pi_*[5]) = 2$. And the procedure outputs a string $ababacb$ when it takes $w = ababacbacb$, $i = 6$, and $k = 2$ as input (see Fig. 4).

Lemma 2: Let w be a string in $L(\pi_*)$. Let i be a positive integer with $2 \le i \le |\pi_*|$. Let k be a nonnegative integer with $0 \le k \le |\pi_*|$. We assume that (π_*, w, i) satisfies Condition 1, and $k = rmvs(\pi_*[1:i-1])$. Given w, i and k as input, Procedure *IdentifyVariables* correctly outputs the value of $rmvs(\pi_*[1:i])$.

Proof: Since (π_*, w, i) satisfies Condition 1, there exists a substitution θ_i such that $w[1:i] = \theta_i(\pi_*[1:i])$ and $w = \theta_i(\pi_*)$. For each integer j with $i < j \le |w|$, there exists no substitution θ_j such that $w[1:j] = \theta_j(\pi_*[1:i])$ and

$$\begin{array}{rll} abbabacbacb & \Rightarrow & a\underline{b}babacbacb.rep(2,\varepsilon) = ababacbacb \in L(axabacy) \\ & \Rightarrow & a\underline{b}abacbacb.rep(2,\varepsilon) = aabacbacb \notin L(axabacy) \end{array} \left.\vphantom{\begin{array}{l} a \\ a \end{array}}\right\} \text{The loop of lines 1–3}$$

Fig. 3. A running example of Procedure *ShrinkString* when a string $w = abbabacbacb$, $i = 2$, and $k = 0$ are given as input. The procedure outputs the string $ababacbacb$.

$$\begin{array}{rll} ababacbacb & \Rightarrow & ababa\underline{c}bacb.rep(6,\varepsilon) = abababacb \in L(axabacy) \\ & \Rightarrow & ababa\underline{b}acb.rep(6,\varepsilon) = ababaacb \notin L(axabacy) \end{array} \left.\vphantom{\begin{array}{l} a \\ a \end{array}}\right\} \text{The loop of lines 1–3}$$

$$\begin{array}{rll} & \Rightarrow & a\underline{b}ababacb.rep(2,\varepsilon) = aababacb \in L(axabacy) \\ & \Rightarrow & a\underline{a}babacb.rep(2,\varepsilon) = ababacb \in L(axabacy) \\ & \Rightarrow & a\underline{b}abacb.rep(2,\varepsilon) = aabacb \notin L(axabacy) \end{array} \left.\vphantom{\begin{array}{l} a \\ a \\ a \end{array}}\right\} \text{The loop of lines 5–7}$$

Fig. 4. A running example of Procedure *ShrinkString* when a string $w = ababacbacb$, $i = 6$, and $k = 2$ are given as input. The procedure outputs the string $ababacb$.

Procedure 2 *IdentifyVariables*

```
Input: A string $w$ in $L(\pi_*)$, a positive integer $i$ and a
    nonnegative integer $k$
Output: an 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 (($k > 0$ and MQ($w'.rep(k, \varepsilon)$) = “yes”)
           or $k = 0$) then $k := i$;
5: end if
6: output $k$;
```

Algorithm 3 *LearningStringPattern*

```
Input: A string $w$ in $L(\pi_*)$
Output: A pattern $\pi$ with $L(\pi) = L(\pi_*)$
1: $i := 1$, $k := 0$, $vSet := \emptyset$;
2: while $i \le |w|$ do
3:     $w :=$ ShrinkString($w, i, k$);
4:     $k :=$ IdentifyVariables($w, i, k$);
5:     if $k \neq 0$ and $k \notin vSet$ then
6:         $vSet := vSet \cup \{k\}$;
7:     end if
8:     $i := i + 1$;
9: end while
10: $\pi := w$;
11: for all $i \in vSet$ do
12:     Let $x$ be a new variable symbol that does not
        appear in $\pi$.
13:     $\pi := \pi.rep(i, x)$;
14: end for
15: output $\pi$;
```

$w = \theta_j(\pi_*)$. Then, we consider the following two cases, that is, $\pi_*[i]$ is a variable symbol, or a constant symbol.

1) In the case that $\pi_*[i]$ is a variable symbol. From the substitution θ_i, **MQ**($w.rep(i, a)$) is “yes”, where $a \in \Sigma \setminus \{w[i]\}$. Let w_1 be the string obtained from w at line 3. At first, we consider the case $k = 0$, that is, $rmvs(\pi_*[1 : i-1]) = 0$. Then, it is clear that k is updated. Next, we consider the case $k > 0$, that is, $rmvs(\pi_*[1 : i-1]) > 0$. Suppose that **MQ**($w_1.rep(k, \varepsilon)$) is “yes”. Since $w_1.rep(k, \varepsilon) \in L(\pi_*)$, there exists a substitution $\theta'_{1,j}$ such that $w_1[1 : j] = \theta'_{1,j}(\pi_*[1 : i])$ and $w_1 = \theta'_{1,j}(\pi_*)$ for some integer j with $i < j$. Since $w[1 : i-1] = w_1[1 : i-1]$ and $\pi_*[i]$ is a variable symbol, by using θ_i and $\theta'_{1,j}$, there exists a substitution θ'_1 such that $w[1 : j] = \theta'_1(\pi_*[1 : i])$ and $w = \theta'_1(\pi_*)$ for some integer j with $i < j$. This contradicts that the triplet (π_*, w, i) satisfies Condition 1. Since **MQ**($w_1.rep(k, \varepsilon)$) is “no”, k is updated.
2) In the case that $\pi_*[i]$ is a constant symbol. We assume that **MQ**($w.rep(i, a)$) is “yes”, where $a \in \Sigma \setminus \{w[i]\}$. Let w_1 be the string obtained from w at line 3. There exists a substitution θ_1 such that $w_1[1 : j] = \theta_1(\pi_*[1 : i])$ and $w_1 = \theta_1(\pi_*)$ for some integer j with $i < j$. If $rmvs(\pi_*[1 : i]) = 0$, then **MQ**($w.rep(i, a)$) is “no”. Thus, we have $rmvs(\pi_*[1 : i]) > 0$. Let $k = rmvs(\pi_*[1 : i-1])$. Since $w[i] \neq w_1[i]$, by using θ_i and θ_1, there exists a substitution θ such that $w_1[1 : k-1] = \theta(\pi_*[1 : k-1])$, $w_1[k : m] = \theta(\pi_*[k])$ and $w_1 = \theta(\pi_*)$ for some integer m with $m > k$. Thus **MQ**($w_1.rep(k, \varepsilon)$) is “yes”. 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 *IdentifyVariables* correctly outputs the value of $rmvs(\pi_*[1 : i])$. ■

Let w_0 be the string which is given to Algorithm *LearningStringPattern*, and $k_0 = 0$. Let $w_1, w_2, \ldots$ be the sequence of strings output by Procedure *ShrinkString* at line 3 of Algorithm *LearningStringPattern*. Let $k_1, k_2, \ldots$ be the sequence of nonnegative integers output by Procedure *IdentifyVariables* at line 4 of *LearningStringPattern*. At each stage $i \ge 1$, Procedure *ShrinkString* outputs the string w_i when w_{i-1}, i, and k_{i-1} are given as input, and Procedure *IdentifyVariables* outputs the nonnegative integer k_i when w_i, i, and k_{i-1} are given as input.

In Table I, we write strings $w_1, \ldots, w_7$ output by Procedure *ShrinkString* and integers $k_1, \ldots, k_7$ output by Procedure *IdentifyVariables* when a string $w_0 = abbabacbacb$ is given to Algorithm *LearningStringPattern* as a positive example. At last, the string $ababacb$ is obtained after the loop of lines 2–9.

Lemma 3: For each integer i with $1 \le i \le |\pi_*|$, the triplet (π_*, w_i, i) satisfies Condition 1, and $k_i = rmvs(\pi_*[1 : i])$.

Proof: The proof is by the induction on the number of iterations $i \ge 1$ of the loop of lines 2–9. We consider the case $i = 1$. The string w_1 is the output of Procedure *ShrinkString* when w_0, $i = 1$, and k_0 are given. It is clear that the triplet (π_*, w_1, i) satisfies Condition 1, and $k_1 = rmvs(\pi_*[1 : 1])$.

TABLE I
A RUNNING EXAMPLE OF **ALGORITHM** *LearningStringPattern* WHEN A POSITIVE EXAMPLE $w_0 = abbabacbacb$ IS GIVEN AS INPUT, WHERE $\pi_* = axabacy$.

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

$vSet = \{2, 7\}$	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*.

We assume inductively that the results hold for any number of iterations of the while loop less than i. By the inductive hypothesis, the triplet $(\pi_*, w_{i-1}, i-1)$ satisfies Condition 1, and $k_{i-1} = rmvs(\pi_*[1 : i-1])$. By Lemma 1, the triplet (π_*, w_i, i) satisfies Condition 1. Since the triplet (π_*, w_i, i) satisfies Condition 1, and $k_{i-1} = rmvs(\pi_*[1 : i-1])$, by Lemma 2, we have $k_i = rmvs(\pi_*[1 : i])$. ■

Theorem 1: Algorithm *LearningStringPattern* outputs a regular pattern π with $L(\pi) = L(\pi_*)$ from one positive example w using $O(|w|)$ membership queries.

Proof: Let $\ell = |\pi_*|$. By Lemma 3, the triplet (π_*, w_ℓ, k_ℓ) satisfies Condition 1. Since $w_\ell \in L(\pi_*)$, we have $|w_\ell| \geq \ell$. We assume that $|w_\ell| > \ell$. Then, there exists a substitution θ_ℓ such that $w_\ell[1 : j] = \theta_\ell(\pi_*[1 : \ell])$ and $w_\ell = \theta_\ell(\pi_*)$ for some integer j with $\ell < j$. This contradicts that the triplet (π_*, w_ℓ, ℓ) satisfies Condition 1. Therefore, we have $|w_\ell| = \ell = |\pi_*|$ and $w_\ell \in L(\pi_*)$. By Lemma 2 and 3, the set $vSet$ in Algorithm *LearningStringPattern* equals the set $\{i \mid \pi_*[i] \in X,\ 1 \leq i \leq |\pi_*|\}$ of positive integers. Thus, Algorithm *LearningStringPattern* outputs a regular pattern π with $L(\pi) = L(\pi_*)$.

At the i-th repetition of lines 2–9, let n_i be a nonnegative integer that is the number of constant symbols removed in Procedure *ShrinkString*. Then, the procedure uses $n_i + 2$ membership queries. At the i-th repetition, Procedure *IdentifyVariables* uses at most two membership queries. The loop of lines 2–9 uses at most $\sum_{i=1}^{|\pi_*|}(n_i + 4)$ membership queries.

Since $\sum_{i}^{|\pi_*|} n_i \leq |w|$ 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. ■

IV. CONCLUSION

In this paper, we have shown 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 introduced a primitive formal ordered tree system (pFOTS) as a formal system defining ordered tree languages [11]. For a pFOTS program Γ as background knowledge, we showed in [11] 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 a query learning algorithm for exactly identifying the class of tree languages derived from a pFOTS 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.

REFERENCES

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[2] H. Mamitsuka and N. Abe. Efficient mining from large databases by query learning. In Proc. *7th International Conference on Machine Learning* (ICML'00), Morgan Kaufmann, pp.575–582, 2000.

[3] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[4] A. Marron. Learning pattern languages from a single initial example and from queries. In Proc. *Workshop on Computational Learning Theory* (COLT'88), Morgan Kaufmann, pp.345–358, 1988.

[5] S. Matsumoto and A. Shinohara. Learning pattern languages using queries. In Proc. *European Conf. on Computational Learning Theory* (EuroCOLT' 97), LNAI Vol.1208, Springer, Berlin, pp.185–197, 1997.

[6] S. Lange and R. Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8, 361–370, 1991.

[7] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[8] J. Nessel and S. Lange. Learning erasing pattern languages with queries. *Theoretical Computer Science*, 348:41–57, 2005.

[9] H. Sakamoto, K. Hirata and H. Arimura. Leaning elementary formal systems with queries. *Theoretical Computer Science*, 298:21–50, 2003.

[10] 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 Transactions on Information and Systems*, E91-D(2):222-230, 2008.

[11] 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 Transactions on Information and Systems*, E101-D(3), to appear, 2019.

[12] A. Nakamura. An efficient query learning algorithm for ordered binary decision diagrams. *Information and Computation*, 201:178–198, 2005.

[13] H. Mizumoto, S. Todoroki, Diptarama, R. Yoshinaka, and A. Shinohara. An efficient query learning algorithm for zero-suppressed binary decision diagrams. In Proc. *Machine Learning Research* (ALT2017), 76:1–12, 2017.