IPSJ Transactions on Mathematical Modeling and Its Applications Vol. 2 No. 3 127–137 (Dec. 2009)

Regular Paper

Learning of Finite Unions of Tree Patterns with Repeated Internal Structured Variables from Queries

Satoshi Matsumoto,†1 Yusuke Suzuki,†2
Takayoshi Shoudai,†3 Tetsuhiro Miyahara†2
and Tomoyuki Uchida†2

The exact learning model by Angluin (1988) is a mathematical model of learning via queries in computational learning theory. A term tree is a tree pattern consisting of ordered tree structures and repeated structured variables, which occur more than once. Thus, a term tree is suited for representing common tree structures based on tree-structured data, such as HTML and XML files on the Web. In this paper, we consider the learnability of finite unions of term trees with repeated variables in the exact learning model. We present polynomial time learning algorithms for finite unions of term trees with repeated variables by using superset and restricted equivalence queries. Moreover, we show that there exists no polynomial time learning algorithm for finite unions of term trees by using restricted equivalence, membership, and subset queries. This result indicates the hardness of learning finite unions of term trees in the exact learning model.

1. Introduction

In the field of Web mining, Web documents such as HTML and XML files have tree structures and are called tree-structured data. To extract meaningful knowledge from given data, many data mining tools require collaboration with experts or users in mining processes. Many such tools have been designed in a query learning scheme. This learning scheme is formulated as the exact learning model by Angluin [4)], which is a mathematical model of learning via queries, in computational learning theory. We are interested in clustering heterogeneous tree-structured data having no rigid structure. From this motivations, in this paper, we consider polynomial time learnabilities of finite unions of tree-structured patterns in the exact learning model.

A term tree is a rooted tree pattern consisting of an ordered tree structure, ordered children, and internal structured variables [10),11),13)]. A variable in a term tree is a list of two vertices, and it can be substituted by an arbitrary tree. Amoth, et al. [1),2)] presented into-matching semantics and introduced the class of ordered tree patterns and ordered forests with this semantics. Such an ordered tree pattern is a standard tree pattern, which is also called a first order term in formal logic. Since a term tree can have variables consisting of two internal vertices (e.g., the variable x_2 in **Fig. 1**), a term tree is more powerful than an ordered tree pattern. Arimura, et al. [6)] presented ordered gapped tree patterns and ordered gapped forests under the into-matching semantics introduced by Amoth, et al. [2)]. An ordered gapped tree pattern is not comparable to a term tree, since a gap-variable in an ordered gapped tree pattern does not exactly correspond to an internal variable in a term tree. A variable with a variable label x in a term tree t is said to be *repeated* if x occurs in t more than once. In this paper, we consider a term tree with repeated variables. Arimura, et al. [6)] discussed polynomial time learnabilities of ordered gapped forests without a repeated gap-variable in the exact learning model. In this paper, on the other hand, we examine polynomial time learnabilities of finite unions of term trees with repeated variables in the exact learning model. For a tree T representing tree-structured data, such as a collection of Web documents, string data such as tags or texts are assigned to the edges of T. Hence, we assume naturally that the cardinality of a set of edge labels is infinite. Let Λ be a set of strings used in tree-structured data. Then, our target class for learning is the class, denoted by $\mathcal{OTF}_\Lambda$, of all finite sets of term trees whose edges are all labeled with elements in Λ. The *term tree language* of a term tree t, denoted by $L_\Lambda(t)$, is the set of all labeled ordered trees that are obtained from t by substituting arbitrary labeled trees for all variables in t. The language represented by a finite set of term trees $R = \{t_1, t_2, \ldots, t_m\}$ in $\mathcal{OTF}_\Lambda$ is the finite union of m term tree languages $L_\Lambda(R) = L_\Lambda(t_1) \cup L_\Lambda(t_2) \cup \ldots \cup L_\Lambda(t_m)$.

In the exact learning model by Angluin [4)], a learning algorithm is said to *exactly learn* a target finite set R_* of term trees if it outputs a finite set R of term trees such that $L_\Lambda(R) = L_\Lambda(R_*)$ and halts, after using some queries. In this paper,

---
†1 Tokai University
†2 Hiroshima City University
†3 Kyushu University

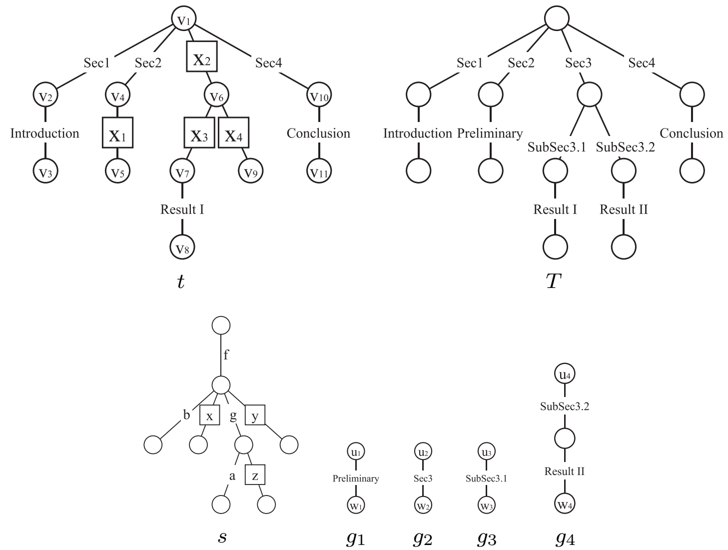


Fig. 1 A term tree t corresponding to a tree T. A term tree s represents the tree pattern $f(b, x, g(a, z), y)$. A variable is represented by a box with lines to its elements. The label inside the box is the variable's label.

first, we present a polynomial time algorithm that exactly learns any finite set in $\mathcal{OTF}_\Lambda$ having m_* term trees by using superset queries for a known number m_*. Second, we present a polynomial time algorithm for the same condition given above, except that the number of term trees in a set in $\mathcal{OTF}_\Lambda$ is unknown and restricted equivalence queries are used. Finally, we show that there exists no polynomial time learning algorithm for finite unions of term trees by using restricted equivalence, membership, and subset queries. This result indicates the hardness of learning finite unions of term trees in the exact learning model.

For the exact learning model, many researchers [1),2),5),6),10),11)] have shown the exact learnabilities of several kinds of tree-structured patterns: e.g., query learning for ordered forests under onto-matching semantics [5)], for unordered forests under into-matching semantics [1),2)], for ordered gapped forests [6)], and for linear term trees [10)]. A term tree t is said to be *linear* (or *repetition-free*) if all variable labels in t are mutually distinct. We showed the polynomial time exact

Table 1 Summary of our previous results and future works. We denote the class of single linear term trees by $\mu\mathcal{OTT}_\Lambda$, and the class of all finite unions of linear term trees by $\mu\mathcal{OTF}_\Lambda$.

	Exact learning		Inductive inference from positive data
$\mu\mathcal{OTT}_\Lambda$	*Yes* [10)] membership & a positive example ($\lvert\Lambda\rvert \geq 2$)		*Yes* [13)] polynomial time ($\lvert\Lambda\rvert \geq 1$)
$\mu\mathcal{OTF}_\Lambda$	*Yes* [11)] restricted subset & equivalence ($\lvert\Lambda\rvert$ is infinite)		*Open*
$\mathcal{OTF}_\Lambda$	sufficient [This work]	insufficient [This work]	*Open*
	superset & restricted equivalence ($\lvert\Lambda\rvert$ is infinite)	restricted equivalence membership subset ($\lvert\Lambda\rvert \geq 1$)	

learnability of finite unions of linear term trees by using restricted subset queries and equivalence queries [11)]. For string patterns, we showed that regular string patterns are exactly learnable by using membership queries and additional information [9)]. As for other learning models, we showed that the class of single linear term trees is polynomial time inductively inferable from positive data [13)]. Further, we gave a data mining method based on semi-structured data, which was based on a learning algorithm for linear term trees [12)]. **Table 1** summarizes our results.

This paper is organized as follows. In Section 2, we introduce some notations and basic definitions concerning term trees and term tree languages. In Section 3, we briefly explain the exact learning model using queries. In Section 4, we show that any finite union of languages defined by term trees is exactly identifiable in polynomial time by using superset queries and restricted equivalence queries. Finally, in Section 5, we show that finite sets of term trees are not learnable in polynomial time by using restricted equivalence, membership, and subset queries, before concluding the paper in Section 6.

2. Preliminaries

For a set S, the number of elements in S, called the *size* of S, is denoted by $|S|$. Let X be an infinite alphabet whose elements is called *variable labels*, and let Λ be an alphabet such that $\Lambda \cap X = \emptyset$. We call an element in Λ an *edge label*, and in this paper, we assume that $|\Lambda|$ is infinite.

Definition 1. Let $T = (V_T, E_T)$ be an edge-labeled rooted tree with a set V_T of vertices and a set E_T of edges labeled with elements in $\Lambda \cup X$. Let H_t be the set of all edges in E_T whose labels are in X. Let $V_t = V_T$ and $E_t = E_T - H_t$ (i.e., $E_t \cup H_t = E_T$ and $E_t \cap H_t = \emptyset$). A triplet $t = (V_t, E_t, H_t)$ is called a *term tree*, and an element in V_t, E_t, and H_t is called a *vertex*, an *edge*, and a *variable*, respectively.

For a term tree $t = (V_t, E_t, H_t)$ and its vertices v_1 and v_i, a *path* from v_1 to v_i is a sequence $v_1, v_2, \ldots, v_i$ of distinct vertices of t such that for any j with $1 \leq j < i$, there exists either an edge or a variable consisting of v_j and v_{j+1}. If there is an edge consisting of v and v' such that v lies on the path from the root to v', then v is said to be the *parent* of v', and v' is a *child* of v. We denote by (v, v') the edge in E_t. If there is a variable consisting of v and v' such that v lies on the path from the root to v', then v is said to be the *parent port* of v', and v' is a *child port* of v. We denote by $[v, v']$ the variable in H_t. A term tree t is called *ordered* if every internal vertex u in t has a total ordering on all children of u. We define the *size* of t as the number of vertices in t and denote it by $|t|$; that is, $|t| = |V_t|$.

For example, the ordered term tree $t = (V_t, E_t, H_t)$ in Fig. 1 is defined as follows: $V_t = \{v_1, \ldots, v_{11}\}$, $E_t = \{(v_1, v_2), (v_2, v_3), (v_1, v_4), (v_7, v_8), (v_1, v_{10}), (v_{10}, v_{11})\}$, with root v_1 and the sibling relation displayed in Fig. 1. $H_t = \{[v_4, v_5], [v_1, v_6], [v_6, v_7], [v_6, v_9]\}$.

We call an ordered term tree simply a *term tree*. In particular, a term tree $t = (V_t, E_t, H_t)$ is *linear* if all variables in H_t have mutually distinct variable labels in X. We denote by $\mathcal{OTT}_\Lambda$ the set of all term trees with Λ as the set of edge labels, and by $\mathcal{OTF}_\Lambda$, the set of all finite sets of term trees with Λ as the set of edge labels; that is, $\mathcal{OTF}_\Lambda = \{S \subset \mathcal{OTT}_\Lambda \mid |S| \text{ is finite}\}$. Similarly, we denote by $\mu\mathcal{OTT}_\Lambda$ the set of all linear term trees with Λ as the set of edge labels, and by $\mu\mathcal{OTF}_\Lambda$, the set of all finite sets of linear term trees with Λ as the set of edge labels; that is, $\mu\mathcal{OTF}_\Lambda = \{S \subset \mu\mathcal{OTT}_\Lambda \mid |S| \text{ is finite}\}$. A term tree with no variable is called a *ground term tree* and considered a tree with ordered children. $\mathcal{OT}_\Lambda$ denotes the set of all ground term trees with Λ as the set of edge labels.

For any term tree t, a vertex u of t, and two children u' and u'' of u, we write $u' <_u^t u''$ if u' is of lower order than u'' among the children of u. Let $f = (V_f, E_f, H_f)$ and $g = (V_g, E_g, H_g)$ be term trees. We say that f and g are *isomorphic*, denoted by $f \equiv g$, if there is a bijection φ from V_f to V_g such that (i) the root of f is mapped to the root of g by φ, (ii) $(u, u') \in E_f$ if and only if $(\varphi(u), \varphi(u')) \in E_g$ and the two edges have the same edge label, (iii) $[u, u'] \in H_f$ if and only if $[\varphi(u), \varphi(u')] \in H_g$, (iv) for any two variables $[u, u']$ and $[v, v']$ in H_f, the variable label of $[u, u']$ is equal to that of $[v, v']$ if and only if the variable label of $[\varphi(u), \varphi(u')]$ is equal to that of $[\varphi(v), \varphi(v')]$, and (v) for any vertex u in f having more than one child, and for any two children u' and u'' of u, $u' <_u^f u''$ if and only if $\varphi(u') <_{\varphi(u)}^g \varphi(u'')$. Two isomorphic term trees are considered identical.

Let f and g be term trees with at least two vertices. Let $h = [v, v']$ be a variable in f with the variable label x, and let $\sigma = [u, u']$ be a list of two distinct vertices in g, where u is the root of g and u' is a leaf of g. The form $x := [g, \sigma]$ is called a *binding* for x. A new term tree $f' = f\{x := [g, \sigma]\}$ is obtained by applying the binding $x := [g, \sigma]$ to f in the following way. Let $e_1 = [v_1, v_1'], \ldots, e_m = [v_m, v_m']$ be the variables in f with the variable label x. Let $g_1, \ldots, g_m$ be m copies of g, and let u_i, u_i' be the vertices of g_i corresponding to u, u' of g, respectively. For each variable $e_i = [v_i, v_i']$, we attach g_i to f by removing the variable e_i from H_f and identifying the vertices v_i, v_i' with the vertices u_i, u_i' of g_i.

A *substitution* θ is a finite collection of bindings $\{x_1 := [g_1, \sigma_1], \cdots, x_n := [g_n, \sigma_n]\}$, where the $x_1, \ldots, x_n$ are mutually distinct variable labels in X. The term tree $f\theta$, called the *instance* of f by θ, is obtained by applying all the bindings $x_i := [g_i, \sigma_i]$ on f simultaneously. We define a new total ordering $<_v^{f\theta}$ on every vertex v in $f\theta$ in the following natural way. Suppose that v has more than one child, and let v' and v'' be two children of v in $f\theta$. There are five cases in which the ordering between v' and v'' must be newly defined. (1) $v \in V_{f\theta} - V_f$: In this case, there is a term tree $g \in \{g_1, \cdots, g_n\}$ such that all of v, v', v'' are in V_g. Then

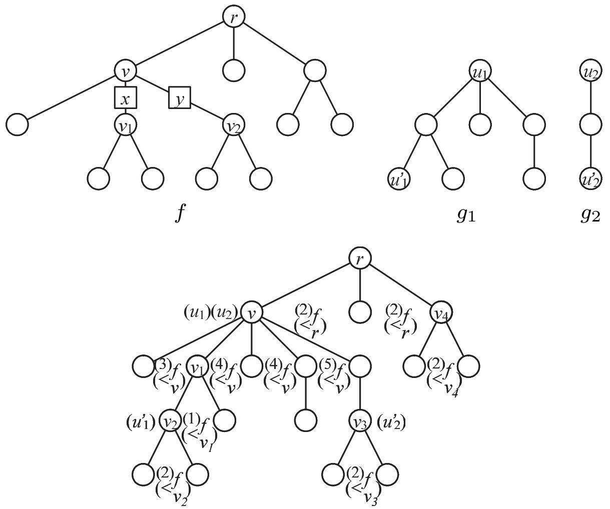


Fig. 2 The new ordering of vertices in the linear term tree $f' = f\{x := [g_1, [u_1, u_1']], y := [g_2, [u_2, u_2']]\}$.

$v' <_v^{f\theta} v''$ is defined if and only if $v' <_v^{g} v''$. On the other hand, if $v \in V_f$, we have the following four subcases. (2) $v' \in V_f$ and $v'' \in V_f$: $v' <_v^{f\theta} v''$ is defined if and only if $v' <_v^{f} v''$. (3) $v' \in V_f$ and there is a term tree $g \in \{g_1, \cdots, g_n\}$ such that $v'' \in V_g$: Let w be the child port of the variable for which g is substituted. Note that v is the parent port of the variable. Then $v' <_v^{f\theta} v''$ (resp. $v'' <_v^{f\theta} v'$) is defined if and only if $v' <_v^{f} w$ (resp. $w <_v^{f} v'$). (4) There is a term tree $g \in \{g_1, \cdots, g_n\}$ such that both v' and v'' are in V_g: Since v is identified with the root of g (say u), $v' <_v^{f\theta} v''$ is defined if and only if $v' <_u^{g} v''$. (5) There are two distinct term trees $g, g' \in \{g_1, \cdots, g_n\}$ such that $v' \in V_g$ and $v'' \in V_{g'}$: Let w and w' be the child ports of the variables for which g and g' are substituted, respectively. Then $v' <_v^{f\theta} v''$ is defined if and only if $w <_v^{f} w'$. In **Fig. 2**, we give an example of the new ordering of vertices in a term tree.

We define the root of the resulting term tree $f\theta$ as the root of f. Consider the examples shown in Fig. 1. An example of a term tree t is given. Let $\theta = \{x_1 := [g_1, [u_1, w_1]], x_2 := [g_2, [u_2, w_2]], x_3 := [g_3, [u_3, w_3]], x_4 := [g_4, [u_4, w_4]]\}$ be a substitution, where g_1, g_2, g_3, and g_4 are the ground term trees in Fig. 1. Then the instance $t\theta$ of the term tree t by θ is isomorphic to the tree T in Fig. 1. Let t and t' be term trees. We write $t \preceq t'$ if there exists a substitution θ such that $t \equiv t'\theta$. If $t \preceq t'$ and $t \not\equiv t'$, then we write $t \prec t'$. The *term tree language* $L_\Lambda(t)$ of a term tree $t \in \mathcal{OTT}_\Lambda$ is $\{s \in \mathcal{OT}_\Lambda \mid s \preceq t\}$. For a set H of term trees, we define $L_\Lambda(H) = \bigcup_{t \in H} L_\Lambda(t)$, and $L_\Lambda(H)$ is called the *term tree language defined by* H. In particular, we define $L_\Lambda(\emptyset) = \emptyset$.

3. Learning Model

In this paper, let R_* be a set of term trees in $\mathcal{OTF}_\Lambda$ to be identified, which we refer to as a *target*. Without loss of generality, we assume that $L_\Lambda(R_*) \neq L_\Lambda(R_* - \{r\})$ for any $r \in R_*$.

We introduce the exact learning model via queries by Angluin [4]. In this model, learning algorithms can access *oracles* that answer specific kinds of queries about the unknown term tree language $L_\Lambda(R_*)$. We consider the following queries.

(1) *Membership query*: The input is a ground term tree T in $\mathcal{OT}_\Lambda$. The output is *"yes"* if $T \in L_\Lambda(R_*)$, and *"no"* otherwise. The oracle that answers a *membership* query is called a *membership oracle*.

(2) *Subset query*: The input is a set R in $\mathcal{OTF}_\Lambda$. The output is *"yes"* if $L_\Lambda(R) \subseteq L_\Lambda(R_*)$; otherwise, the output is a ground term tree, called a *counterexample*, in $L_\Lambda(R) - L_\Lambda(R_*)$. The oracle that answers a *subset* query is called a *subset oracle*.

(3) *Superset query*: The input is a set R in $\mathcal{OTF}_\Lambda$. The output is *"yes"* if $L_\Lambda(R_*) \subseteq L_\Lambda(R)$; otherwise, the output is a ground term tree, called a *counterexample*, in $L_\Lambda(R_*) - L_\Lambda(R)$. The oracle that answers a *superset* query is called a *superset oracle*.

(4) *Restricted equivalence query*: The input is a set R in $\mathcal{OTF}_\Lambda$. The output is *"yes"* if $L_\Lambda(R) = L_\Lambda(R_*)$, and *"no"* otherwise. The oracle that answers a *restricted equivalence* query is called a *restricted equivalence oracle*.

A learning algorithm $\mathcal{A}$ collects information about $L_\Lambda(R_*)$ by using queries and outputs a set R in $\mathcal{OTF}_\Lambda$. We say that a learning algorithm $\mathcal{A}$ *exactly identifies* a target R_* in polynomial time by using certain kinds of queries if $\mathcal{A}$ halts in polynomial time and outputs a set $R \in \mathcal{OTF}_\Lambda$, such that $L_\Lambda(R) = L_\Lambda(R_*)$, by using the certain kinds of queries.

4. Learning Finite Unions of Term Tree Languages

In this section, we show the learnability of finite unions of term tree languages in the framework of the exact learning model.

4.1 An Overview of Our Learning Algorithms

The property shown by the following lemma, called *compactness*, plays an important role in the learning of unions of languages [6),7)]. We remark that $|\Lambda|$ is infinite, again.

Lemma 1. Let r be a term tree in $\mathcal{OTT}_\Lambda$ and R a set in $\mathcal{OTF}_\Lambda$. Then, $r \preceq r'$ for some $r' \in R$ if and only if $L_\Lambda(r) \subseteq L_\Lambda(R)$.

Proof. From the definition of the binary relation $\preceq$, only if part is clear. Let w_r be a ground term tree obtained from r by substituting edges having mutually distinct labels not appearing in R. If $L_\Lambda(r) \subseteq L_\Lambda(R)$ then w_r is in $L_\Lambda(R)$. Therefore, there exists a term tree r' in R such that w_r is in $L_\Lambda(r')$. Since any edge of w_r whose label doesn't appear in R, we have $r \preceq r'$ by inverting the substitution. □

If $|\Lambda|$ is finite, Lemma 1 does not hold because of an example violating the property of compactness as follows: For $\Lambda = \{a_1, \ldots, a_k\}$ $(k \geq 1)$ and linear term trees f, $g_1, \ldots, g_{k+2}$ given in **Fig. 3**, the equation $L_\Lambda(f) = L_\Lambda(g_1) \cup \cdots \cup L_\Lambda(g_{k+2})$ holds, but $L_\Lambda(f) \not\subseteq L_\Lambda(g_i)$ for all i $(1 \leq i \leq k+2)$.

We introduce some notations. For a term tree r in $\mathcal{OTT}_\Lambda$, we define $R_*(r) = \{r_* \in R_* \mid |r| = |r_*| \text{ and } r_* \prec r\}$. For linear term trees r, r', we write $r \vdash r'$ if r' is obtained from r by replacing one of the variables in r with one of the three linear term trees g_1, g_2, g_3 given in **Fig. 4**. For a linear term tree r in $\mu\mathcal{OTT}_\Lambda$, let $\mathcal{ES}(r) = \{r' \in \mu\mathcal{OTT}_\Lambda \mid r \vdash r'\}$. Note that $|r'| > |r|$ and $r' \prec r$ for any $r' \in \mathcal{ES}(r)$, and $|\mathcal{ES}(r)| \leq 3(|r|-1)$. If r has variables, then $L_\Lambda(\mathcal{ES}(r))$ includes all ground term trees t such that $t \preceq r$ and $|t| > |r|$ hold.

Let r be a term tree in $\mathcal{OTT}_\Lambda$, α an edge label, and x, y variable labels appearing in r. We denote by X_r the set of all variable labels appearing in r. $\rho_e(r, x, \alpha)$ denotes the term tree obtained from r by replacing variables having the variable label x with edges having the edge label α. $\rho_v(r, x, y)$ denotes the term tree obtained from r by replacing variables having the variable label x with variables having the variable label y. For a finite subset Δ of Λ, we define the set $\mathcal{RS}_\Delta(r)$ as follows:

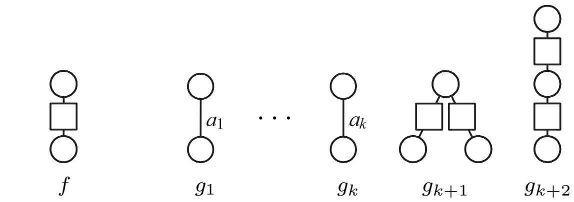


Fig. 3 An example violating the property *compactness*.

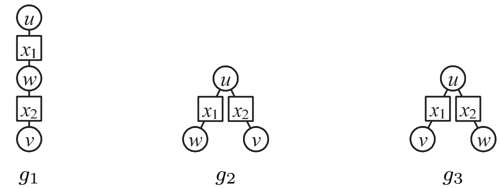


Fig. 4 Linear term trees $g_1 = (\{u, v, w\}, \emptyset, \{[u, w], [w, v]\})$, $g_2 = (\{u, v, w\}, \emptyset, \{[u, w], [u, v]\})$ where $w <_u^{g_2} v$ and $g_3 = (\{u, v, w\}, \emptyset, \{[u, v], [u, w]\})$ where $v <_u^{g_3} w$.

$$\mathcal{RS}_\Delta(r) = \{\rho_e(r, x, \alpha) \in \mathcal{OTT}_\Lambda \mid x \in X_r \text{ and } \alpha \text{ is an edge label in } \Delta\}$$
$$\cup\ \{\rho_v(r, x, y) \in \mathcal{OTT}_\Lambda \mid x, y \in X_r,\ x \text{ and } y \text{ are different}\}$$

Since $r' \prec r$ and $|r'| = |r|$ for any $r' \in \mathcal{RS}_\Delta(r)$, we have $r \notin \mathcal{RS}_\Delta(r)$. The number of non-isomorphic term trees in $\mathcal{RS}_\Delta(r)$ is at most $|r| \cdot |\Delta| + |r|^2$. If $t \in \mathcal{OT}_\Lambda$, then we define $\mathcal{RS}_\Delta(t) = \emptyset$.

In this paper, we consider two cases: the size of R_* is either known or unknown in advance. Let $|R_*| = m_*$. In case that the size of R_* is known in advance, we present Algorithm *LEARN_KNOWN* given in **Fig. 5** which exactly identifies any set $R_* \in \mathcal{OTF}_\Lambda$ in polynomial time by using superset queries when the size m_* of R_* is given as an input. Algorithm *LEARN_KNOWN* starts with a term tree consisting of only one variable. By recursively replacing a variable with a term tree consisting of two variables, Algorithm *LEARN_KNOWN* generates a set of linear term trees $r = (V_r, E_r, H_r)$ such that $E_r = \emptyset$, $|r| = |r_*|$, and $r_* \preceq r$ for some $r_* \in R_*$. Next, in Algorithm *LEARN_KNOWN*, for each linear term tree r in the resultant set, Algorithm *LEARN_OTT* given in **Fig. 6** changes variable labels with other variable labels, or replaces variables with edges. Finally,

Algorithm *LEARN_KNOWN*
Input: A positive integer m_*;
Output: A set $R_{hypo} \in \mathcal{OTF}_\Lambda$ with $L_\Lambda(R_{hypo}) = L_\Lambda(R_*)$;
begin
Let $R_{hypo} := \emptyset$;
if $Sup_{R_*}(R_{hypo})$ = "*yes*" **then output** R_{hypo};
else begin
  Let $r = (\{u,v\}, \emptyset, \{[u,v]\}) \in \mu\mathcal{OTT}_\Lambda$; $R_{hypo} := R_{nocheck} := \{r\}$;
  while $R_{nocheck} \neq \emptyset$ **do begin**
    $ES_{total} := \emptyset$;
    foreach $r \in R_{nocheck}$ **do begin**
      if $Sup_{R_*}((R_{hypo} - \{r\}) \cup \mathcal{ES}(r))$ = "*yes*" **then begin**
        $(R_{hypo}, ES_{tmp}) := REMOVE(R_{hypo}, r)$;
      end
      else begin
        $R' := LEARN\_OTT(m, (R_{hypo} - \{r\}) \cup \mathcal{ES}(r), r)$;
        $(R_{hypo}, ES_{tmp}) := REMOVE(R_{hypo} \cup R', r)$;
      end;
      $ES_{total} := ES_{total} \cup ES_{tmp}$;
    end;
    $R_{nocheck} := ES_{total}$;
  end;
end;
output R_{hypo};
end.

Fig. 5 Algorithm *LEARN_KNOWN*. We denote a superset query by Sup_{R_*}.

Algorithm *LEARN_KNOWN* finds a set R_{hypo} of term trees with $L_\Lambda(R_{hypo}) = L_\Lambda(R_*)$.

In case that the size of R_* is unknown in advance, we present Algorithm *LEARN_OTF* in Fig. 8 which outputs a set $R \in \mathcal{OTF}_\Lambda$ with $L_\Lambda(R) = L_\Lambda(R_*)$ by using Algorithm *LEARN_KNOWN* and restricted equivalence queries.

4.2 The Correctness of Algorithm *LEARN_OTT*

At first, in case that the size of R_* is known in advance, we show that Algorithm *LEARN_KNOWN* exactly identifies any set $R_* \in \mathcal{OTF}_\Lambda$ by using superset queries. In Algorithm *LEARN_KNOWN*, we use Algorithm *LEARN_OTT* and Algorithm *REMOVE* in **Fig. 7**. Lemma 2 ensures that Algorithm *LEARN_OTT* takes as input a term tree r such that $r_* \preceq r$ and $|r_*| = |r|$ for some $r_* \in R_*$. Lemmas 3 and 4 ensures that Algorithm *LEARN_OTT* outputs the set $R_*(r)$ finally.

Algorithm *LEARN_OTT*
Given: a positive integer m, a set R_{in} in $\mathcal{OTF}_\Lambda$ and a term tree r_{in} in $\mathcal{OTT}_\Lambda$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(R_{in} \cup \{r_{in}\})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in})$;
Output: A set S in $\mathcal{OTF}_\Lambda$;
begin
$S := \emptyset$;
if $r_{in} \in \mathcal{OT}_\Lambda$ **then** $S := \{r_{in}\}$
else begin
  Let $n := 0$;
  Let t be a counterexample given by $Sup_{R_*}(R_{in})$;
  Let $X_{r_{in}}$ be the set of all variable labels in r_{in}
  Let Δ be the set of all edge labels in t;
  while $Sup_{R_*}(R_{in} \cup \mathcal{RS}_\Delta(r_{in})) \neq$ "*yes*" and $n \leq m$ **do begin**
    Let t' be a counterexample and Δ' the set of all edge labels in t';
    $\Delta := \Delta \cup \Delta'$; $n := n + 1$;
  end;
  if $n > m$ **then** $S := \{r_{in}\}$
  else begin
    foreach $r \in \mathcal{RS}_\Delta(r_{in})$ **do** /* Remove redundant term trees in $\mathcal{RS}_\Delta(r_{in})$. */
      if $Sup_{R_*}(R_{in} \cup (\mathcal{RS}_\Delta(r_{in}) - \{r\}))$ = "*yes*" **then**
        $\mathcal{RS}_\Delta(r_{in}) := \mathcal{RS}_\Delta(r_{in}) - \{r\}$;
    $RS_{tmp} := \mathcal{RS}_\Delta(r_{in})$;
    foreach $r \in \mathcal{RS}_\Delta(r_{in})$ **do begin**
      $RS_{tmp} := RS_{tmp} - \{r\}$;
      $S' := LEARN\_OTT(m, R_{in} \cup RS_{tmp} \cup S, r)$; $S := S \cup S'$;
    end;
  end;
end;
output S;
end.

Fig. 6 Algorithm *LEARN_OTT*. We denote by Sup_{R_*} *Superset* query.

Lemma 2. Let R be a set in $\mu\mathcal{OTF}_\Lambda$, r a term tree in R, and R' a set in $\mathcal{OTF}_\Lambda$. If $L_\Lambda(R') \subseteq L_\Lambda(R)$ and $L_\Lambda(R') \not\subseteq L_\Lambda(R - \{r\}) \cup L_\Lambda(\mathcal{ES}(r))$, then there exists a term tree $r' \in R'$ such that $r' \preceq r$ and $|r'| = |r|$.


```
Algorithm REMOVE
Input:  A set R ∈ OTF_Λ and a term tree r ∈ R;
Output: a pair (H, S) of sets in OTF_Λ;
begin
  H := R − {r} ∪ ES(r); S := ES(r);
  foreach r′ ∈ ES(r) do begin
    if Sup_{R_*}(H − {r′}) = "yes" then begin
      H := H − {r′}; S := S − {r′}
    end;
  end;
  output (H, S);
end.
```

Fig. 7 Algorithm *REMOVE*. We denote by Sup_{R_*} *Superset* query.

Proof. Let r_c be a ground term tree in $L_\Lambda(R') - (L_\Lambda(R - \{r\}) \cup L_\Lambda(\mathcal{ES}(r)))$. Since $r_c \in L_\Lambda(R')$, there exists a term tree r' in R' such that $r_c \in L_\Lambda(r')$. We assume $r' \not\preceq r$. By Lemma 1 and $L_\Lambda(r') \subseteq L_\Lambda(R') \subseteq L_\Lambda(R)$, there exists a term tree r'' in R such that $r' \preceq r''$. Then, $r_c \in L_\Lambda(r') \subseteq L_\Lambda(r'') \subseteq L_\Lambda(R - \{r\})$. This is a contradiction, so we must have $r' \preceq r$. Since $r' \preceq r$, it is clear that $|r'| \geq |r|$. Next, we assume $|r'| > |r|$. Then, $r_c \in L_\Lambda(r') \subseteq L_\Lambda(\mathcal{ES}(r))$. This is a contradiction, and thus, we must have $|r'| = |r|$. □

By Lemma 2, if the answer for $Sup_{R_*}((R_{hypo} - \{r\}) \cup \mathcal{ES}(r))$ in line 8 of Algorithm *LEARN_KNOWN* is "*no*", then r satisfies $r_* \preceq r$ and $|r_*| = |r|$ for some $r_* \in R_*$.

We denote by r_{in} and R_{in} a term tree and a set of term trees, respectively, which are inputs of Algorithm *LEARN_OTT*. By Lemma 2, Algorithm *LEARN_OTT* always takes as input a term tree r_{in} such that $r_* \preceq r_{in}$ and $|r_*| = |r_{in}|$ for some $r_* \in R_*$. We have two cases for r_{in}: (1) There exists a term tree $r_* \in R_*$ with $r_{in} \equiv r_*$. (2) There exist term trees $r_* \in R_*$ with $r_* \prec r_{in}$ and $|r_*| = |r_{in}|$. For case (1), Lemma 3 ensures that Algorithm *LEARN_OTT* repeats the while-loop in lines 8-11 more than m times. Algorithm *LEARN_OTT* outputs the term tree r_{in}. For case (2), Lemma 4 ensures that Algorithm *LEARN_OTT* repeats the while-loop in lines 8-11 less than $m+1$ times. Algorithm *LEARN_OTT* calls itself recursively and gives a term tree r with $|r| = |r_{in}|$ and $r \prec r_{in}$. Note that $L_\Lambda(R_*) \subseteq L_\Lambda(R_{in} \cup \{r_{in}\})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in})$. Thus, r_{in} is not included in R_{in}.

In Algorithm *LEARN_OTT*, let $t_1', t_2', \ldots, t_n', \ldots$ and $\Delta_1, \Delta_2, \ldots, \Delta_n, \ldots$ $(n \geq 1)$ be the sequence of counterexamples returned by the superset queries in line 8 and the sequence of finite subsets of Λ obtained in line 10, respectively. Let Δ_0 be the finite subset of Δ obtained in line 7. We suppose that at each stage $n \geq 0$, Algorithm *LEARN_OTT* makes a superset query $Sup_{R_*}(R_{in} \cup \mathcal{RS}_{\Delta_n}(r_{in}))$ and receives a counterexample t_{n+1} to the query.

First, we consider the case (1), that is, there exists a term tree $r_{in} \equiv r_*$ for some $r_* \in R_*$.

Lemma 3. If $r_{in} \equiv r_*$ for some $r_* \in R_*$, then $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in} \cup \mathcal{RS}_{\Delta_n}(r_{in}))$ for any $n \geq 0$.

Proof. If r_{in} has no variable, then $r_{in} \in \mathcal{OT}_\Lambda$. Thus $\mathcal{RS}_\Delta(r_{in}) = \emptyset$. Moreover, we have $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in})$. Then we have $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in} \cup \mathcal{RS}_{\Delta_n}(r_{in}))$.

We thus assume that r_{in} has variables. Let D be a finite set of Λ. We assume $L_\Lambda(R_*) \subseteq L_\Lambda(R_{in} \cup \mathcal{RS}_D(r_{in}))$. Since $r_{in} \equiv r_*$ for some $r_* \in R_*$, we have $L_\Lambda(r_{in}) \subseteq L_\Lambda(R_*) \subseteq L_\Lambda(R_{in} \cup \mathcal{RS}_D(r_{in}))$. By Lemma 1, we have two cases: (i) There exists a term tree $r \in R_{in}$ with $r_{in} \preceq r$. (ii) There exists a term tree $r \in \mathcal{RS}_D(r_{in})$ with $r_{in} \preceq r$. For case (i), this contradicts with $L_\Lambda(R_*) \subseteq L_\Lambda(R_{in} \cup \{r_{in}\})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in})$. For case (ii), by the definition of $\mathcal{RS}_D(r_{in})$ and $r \in \mathcal{RS}_D(r_{in})$, we have $r \prec r_{in}$. This contradicts with $r_{in} \preceq r$. Therefore, we have $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in} \cup \mathcal{RS}_{\Delta_n}(r_{in}))$ for any $n \geq 0$. □

By the above lemma, if $r_{in} \equiv r_*$ for some $r_* \in R_*$, then the while-loop in lines 8-11 of Algorithm *LEARN_OTT* is repeated more than m times. Thus, Algorithm *LEARN_OTT* outputs a term tree in R_*.

Next, we consider the case (2), that is, there exist term trees $r_* \in R_*$ with $r_* \prec r_{in}$ and $|r_*| = |r_{in}|$.

Lemma 4. If there exist term trees $r_* \in R_*$ with $r_* \prec r_{in}$ and $|r_*| = |r_{in}|$, then there exists a subset S of $R_*(r_{in})$ such that $|S| \geq n+1$ and $L_\Lambda(S) \subseteq L_\Lambda(\mathcal{RS}_{\Delta_n}(r_{in}))$ for any $n \in \{1, \ldots, \ell - 1\}$, where $\ell = |R_*(r_{in})|$.

Proof. The proof is by induction on the number of iterations $n \geq 0$ of the while-loop in lines 8-11 of Algorithm *LEARN_OTT*. In the case of $n = 0$, let t be a ground term tree given by $Sup_{R_*}(R_{in})$ as a counterexample in line 5. Then, $t \in L_\Lambda(r_*')$ for some $r_*' \in R_*(r_{in})$. Since Δ_0 is the set of edge labels appearing in t, $r'_* \preceq r$ for some $r \in \mathcal{RS}_{\Delta_0}(r_{in})$. Thus, we have $L_\Lambda(\{r'_*\}) \subseteq L_\Lambda(\mathcal{RS}_{\Delta_0}(r_{in}))$. Next, we assume inductively that the result holds for any number of iterations of the while-loop less than n. By the inductive hypothesis, there exists a subset S of $R_*(r_{in})$ such that $|S| \geq n$ and $L_\Lambda(S) \subseteq L_\Lambda(\mathcal{RS}_{\Delta_{n-1}}(r_{in}))$. If $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in} \cup \mathcal{RS}_{\Delta_{n-1}}(r_{in}))$, we obtain t'_n. Since $L_\Lambda(S) \subseteq L_\Lambda(\mathcal{RS}_{\Delta_{n-1}}(r_{in}))$, there exists a term tree $r'_* \in R_*(r_{in}) - S$ such that $t'_i \in L_\Lambda(r'_*)$. We have $r \in \mathcal{RS}_{\Delta_n}(r_{in})$ with $r'_* \preceq r$. Thus, there exists a subset S' of $R_*(r_{in})$ such that $|S'| \geq n+1$ and $L_\Lambda(S') \subseteq L_\Lambda(\mathcal{RS}_{\Delta_n}(r_{in}))$, where $S \cup \{r'_*\} \subseteq S'$. □

By the above lemma, if there exists a term tree $r_* \in R_*$ such that $r_* \prec r_{in}$ and $|r_*| = |r_{in}|$, then the while-loop in lines 8-11 of Algorithm *LEARN_OTT* is repeated less than $m+1$ times. Thus, Algorithm *LEARN_OTT* calls itself recursively. By Lemmas 3 and 4, we have the following theorem.

Theorem 5. Algorithm *LEARN_OTT* correctly outputs the set $R_*(r)$ for an input term tree r such that $r_* \preceq r$ and $|r_*| = |r|$ for some $r_* \in R_*$.

4.3 An Algorithm for Reducing a Set of Term Trees

Algorithm *REMOVE* removes an unnecessary term tree from R_{hypo}, which is a term tree r' such that $L_\Lambda(R_{hypo} - \{r'\}) = L_\Lambda(R_*)$ holds, under several conditions for its input. We give two lemmas for showing that Algorithm *REMOVE* correctly works in Algorithm *LEARN_KNOWN*. Lemmas 6 and 7 describe the conditions which correspond to lines 9 and 13 of Algorithm *LEARN_KNOWN*, respectively.

Let H_i $(i \geq 1)$ be a set of term trees immediately after the i-th execution of Algorithm *REMOVE*.

Lemma 6. Let R be a set in $\mathcal{OTF}_\Lambda$ and a term tree $r \in R$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(R - \{r\} \cup \mathcal{ES}(r))$, $L_\Lambda(R_*) \subseteq L_\Lambda(R)$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R - \{r'\})$ for any $r' \in R$. Let (H, S) be a pair of sets in $\mathcal{OTF}_\Lambda$ output by Algorithm *REMOVE* given R and r. Then $|H| \leq |R_*|$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(H - \{r'\})$ for any $r' \in H$.

Proof. By the algorithm, we have $L_\Lambda(R_*) \subseteq L_\Lambda(H)$. We assume that there exists a term tree $s \in H$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\})$. We have two cases: (i) $s \in \mathcal{ES}(r) \cap H$. (ii) $s \in (R - \{r\}) \cap H$. At first, we show the case (i). We assume that there exists a term tree $s \in \mathcal{ES}(r) \cap H$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\})$. Let $r'_1, r'_2, \ldots, r'_i, \ldots$ be the sequence of term trees in $\mathcal{ES}(r)$ used in line 2. Let i_0 be the minimum integer which satisfies $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{r'_{i_0}\})$. Since $H \subseteq H_i$ for any i, $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{r'_{i_0}\}) \subseteq L_\Lambda(H_{i_0} - \{r'_{i_0}\})$. By the algorithm, since $r'_{i_0} \in H$, we have $L_\Lambda(R_*) \not\subseteq L_\Lambda(H_{i_0} - \{r'_{i_0}\})$. This is a contradiction.

Next, we show the case (ii). We have $H_1 = R - \{r\} \cup \mathcal{ES}(r)$. We assume that there exists a term tree $s \in (R - \{r\}) \cap H$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\})$. Since $H \subseteq H_1$ and $L_\Lambda(H_1) = L(R - \{r\} \cup \mathcal{ES}(r)) \subseteq L_\Lambda(R)$, we have $L_\Lambda(H) \subseteq L_\Lambda(R)$. Thus, $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\}) \subseteq L_\Lambda(R - \{s\})$. This contradicts with $L_\Lambda(R_*) \not\subseteq L_\Lambda(R - \{r'\})$ for any $r' \in R$. Therefore, we have $L_\Lambda(R_*) \not\subseteq L_\Lambda(H - \{r'\})$ for any $r' \in H$. Moreover, by $L_\Lambda(R_*) \subseteq L_\Lambda(H)$ and Lemma 1, we have $|H| \leq |R_*|$. □

Lemma 7. Let R be a set in $\mathcal{OTF}_\Lambda$ and a term tree $r \in R$ such that $L_\Lambda(R_*) \not\subseteq L_\Lambda(R - \{r\} \cup \mathcal{ES}(r))$, $L_\Lambda(R_*) \subseteq L_\Lambda(R \cup R_*(r) - \{r\} \cup \mathcal{ES}(r))$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R \cup R_*(r) - \{r\} \cup \mathcal{ES}(r) - \{r'\})$ for any $r' \in R \cup R_*(r) - \{r\}$. Let (H, S) be a pair of sets in $\mathcal{OTF}_\Lambda$ output by Algorithm *REMOVE* given $R \cup R_*(r)$ and r. Then $|H| \leq |R_*|$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(H - \{r'\})$ for any $r' \in H$.

Proof. By the algorithm, we have $L_\Lambda(R_*) \subseteq L_\Lambda(H)$. We assume that there exists a term tree $s \in H$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\})$. We have two cases: (i) $s \in \mathcal{ES}(r) \cap H$. (ii) $s \in (R \cup R_*(r) - \{r\}) \cap H$. At first, we show the case (i). We assume that there exists a term tree $s \in \mathcal{ES}(r) \cap H$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\})$. Let $r'_1, r'_2, \ldots, r'_i, \ldots$ be the sequence of term trees in $\mathcal{ES}(r)$ used in line 2. Let i_0 be the minimum integer which satisfies $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{r'_{i_0}\})$. By the algorithm, since $r'_{i_0} \in H$, we have $L_\Lambda(R_*) \not\subseteq L_\Lambda(H_{i_0} - \{r'_{i_0}\})$. Since $H \subseteq H_i$ for any i, $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{r'_{i_0}\}) \subseteq L_\Lambda(H_{i_0} - \{r'_{i_0}\})$. This is a contradiction.

Next, we show the case (ii). We have $H_1 = R \cup R_*(r) - \{r\} \cup \mathcal{ES}(r)$. We assume that there exists a term tree $s \in (R \cup R_*(r) - \{r\}) \cap H$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\})$. Since $H \subseteq H_1$ and $L_\Lambda(H_1) = L(R \cup R_*(r) - \{r\} \cup \mathcal{ES}(r))$, we have $L_\Lambda(H) \subseteq L_\Lambda(R \cup R_*(r) - \{r\} \cup \mathcal{ES}(r))$. Thus, $L_\Lambda(R_*) \subseteq L_\Lambda(H - \{s\}) \subseteq L_\Lambda(R \cup R_*(r) - \{r\} \cup \mathcal{ES}(r) - \{s\})$. This contradicts with $L_\Lambda(R_*) \not\subseteq L_\Lambda(R \cup R_*(r) - \{r\} \cup \mathcal{ES}(r) - \{r'\})$ for any $r' \in R \cup R_*(r) - \{r\}$. Therefore, we have $L_\Lambda(R_*) \not\subseteq L_\Lambda(H - \{r'\})$ for any $r' \in H$. Moreover, by $L_\Lambda(R_*) \subseteq L_\Lambda(H)$ and Lemma 1, we have $|H| \leq |R_*|$. □

4.4 Main Theorem

We show that Algorithm *LEARN_KNOWN* correctly outputs a minimal set of term trees that is equal to $L_\Lambda(R_*)$ in polynomial time.

Theorem 8. If Algorithm *LEARN_KNOWN* takes an integer m with $m \geq |R_*|$ as input, then it exactly identifies a set $R_* \in \mathcal{OTF}_\Lambda$ in polynomial time using at most $O(m^2n^3+1)$ superset queries, where n is the maximum size of term trees in R_*.

Proof. By Theorem 5 and the process of Algorithm *LEARN_KNOWN*, we easily see that the algorithm outputs a set of term trees that is equal to $L_\Lambda(R_*)$. First we show that the output set of Algorithm *LEARN_KNOWN* is a minimal set of term trees that is equal to $L_\Lambda(R_*)$.

Let R^i_{hypo} $(i \geq 1)$ be a hypothesis set in $\mathcal{OTF}_\Lambda$ immediately after the i-th execution of line 7 of Algorithm *LEARN_KNOWN*. We show that $L_\Lambda(R_*) \subseteq L_\Lambda(R^i_{hypo})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R^i_{hypo} - \{r\})$ for any $r \in R^i_{hypo}$. The proof is by induction on the number of iterations $i \geq 1$. In the case of $i=1$, it is clear. We assume inductively that the result holds for any number less than i. By the inductive hypothesis, $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R^{i-1}_{hypo} - \{r\})$ for any $r \in R^{i-1}_{hypo}$. Then we have two cases: (i) $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo} - \{r\} \cup \mathcal{ES}(r))$. (ii) $L_\Lambda(R_*) \not\subseteq L_\Lambda(R^{i-1}_{hypo} - \{r\} \cup \mathcal{ES}(r))$. For case (i), by Lemma 6, it is clear that $L_\Lambda(R_*) \subseteq L_\Lambda(R^i_{hypo})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R^i_{hypo} - \{r\})$ for any $r \in R^i_{hypo}$. For case (ii), we show that $L_\Lambda(R_*) \not\subseteq L_\Lambda(R^{i-1}_{hypo} \cup R' - \{r\} \cup \mathcal{ES}(r) - \{r'\})$ for any $r' \in R^{i-1}_{hypo} \cup R' - \{r\}$. We assume that there exists a term tree $s \in R^{i-1}_{hypo} \cup R' - \{r\}$ such that $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo} \cup R' - \{r\} \cup \mathcal{ES}(r) - \{s\})$. Moreover, we consider two cases: (ii-1) $s \in R^{i-1}_{hypo} - \{r\}$. (ii-2) $s \in R' - \{r\}$. In the case (ii-1), since $r \in R^{i-1}_{hypo}$ and $t \preceq r$ for any $t \in R' \cup \mathcal{ES}(r)$, we have $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo} - \{s\})$. This is contradiction. In the case (ii-2), since $s \in R' - \{r\}$ and $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo} \cup R' - \{r\} \cup \mathcal{ES}(r) - \{s\})$, there exists a term tree $t \in R^{i-1}_{hypo}$ with $s \preceq t$. Then, $s \equiv t \preceq r$, $s \prec t \preceq r$, or $s \prec r \preceq t$. These follow $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo} - \{s\})$, $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo} - \{t\})$, or $L_\Lambda(R_*) \subseteq L_\Lambda(R^{i-1}_{hypo} - \{r\})$. These are contradictions. By Lemma 7, we have $L_\Lambda(R_*) \subseteq L_\Lambda(R^i_{hypo})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R^i_{hypo} - \{r\})$ for any $r \in R^i_{hypo}$. Therefore, since Lemma 1, $L_\Lambda(R_*) \subseteq L_\Lambda(R^i_{hypo})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R^i_{hypo} - \{r\})$ for any $i \geq 1$ and any $r \in R^i_{hypo}$, we have $|R^i_{hypo}| \leq |R_*|$. From the above, redundant term trees are not included in R_{hypo}.

Let ES^i_{total} be a hypothesis set in $\mathcal{OTF}_\Lambda$ immediately after the i-th execution of line 7 of Algorithm *LEARN_KNOWN*. By the algorithm, we have $ES^i_{total} \subseteq R^i_{hypo}$ for any i. Thus $|ES^i_{total}| \leq |R^i_{hypo}| \leq |R_*|$ for any i.

In a similar proof to Lemmas 6 and 7, for RS_{tmp} in line 17 in Algorithm *LEARN_OTT*, we can show that $|RS_{tmp}| \leq |R_*(r_{in})|$ and $L_\Lambda(R_*) \subseteq L_\Lambda(R_{in} \cup RS_{tmp})$ and $L_\Lambda(R_*) \not\subseteq L_\Lambda(R_{in} \cup RS_{tmp} - \{r'\})$ for any $r' \in RS_{tmp}$. Thus, in the foreach-loop in lines 18-21, Algorithm *LEARN_OTT* avoids redundant recursive calls.

By Lemma 4, the while-loop in lines 8–11 of Algorithm *LEARN_OTT* is repeated no more than m times. After removing redundant term trees in $\mathcal{RS}_\Delta(r_{in})$, Algorithm *LEARN_OTT* is called recursively. The algorithm is called recursively at most $O(\ell|r_{in}|)$ times in all. The while-loop in lines 8–11 is repeated at most $O(m)$ times. Note that $|t_i| = |r_{in}|$ for any i. Thus, in the foreach-loop in lines 14–16, $|\Delta| \leq |t_1|+\ldots+|t_m| = m|r_{in}|$. The loop uses at most $O(m|r_{in}|^2)$ superset queries. The number of superset queries needed to identify the set $\{r^1_*, \ldots, r^\ell_*\}$ is at most $O(\ell m|r_{in}|^3)$. Algorithm *LEARN_KNOWN* uses at most $O(|r_{in}|^2)$ superset queries to obtain a term tree r_{in}. Thus, the number of superset queries the algorithm needs to identify a target R_* is at most $O(m^2n^3)$, where n is the maximum size of term trees in R_*. □

Algorithm *LEARN_OTF*
Output: A set $R \in \mathcal{OTF}_\Lambda$ with $L_\Lambda(R) = L_\Lambda(R_*)$.
begin
  $m := 0$; $R := \phi$;
  repeat
    $m := m+1$;
    $R :=$ *LEARN_KNOWN*(m);
  until $rEquiv_{R_*}(R) =$ "yes";
  output R;
end.

Fig. 8 Algorithm *LEARN_OTF*. We denote a restricted equivalence query by $rEquiv_{R_*}$.

In case that the size of R_* is unknown in advance, we present Algorithm *LEARN_OTF* in **Fig. 8** which outputs a set $R \in \mathcal{OTF}_\Lambda$ with $L_\Lambda(R) = L_\Lambda(R_*)$ using superset queries and restricted equivalence queries.

Theorem 9. Algorithm *LEARN_OTF* of Fig. 8 exactly identifies any set $R_* \in$

$\mathcal{OTF}_\Lambda$ in polynomial time using at most $O(m_*^3 n^3 + 1)$ superset queries and at most $O(m_* + 1)$ restricted equivalence queries, where n is the maximum size of term trees in R_*.

5. Hardness Result on Learnability

In this section, we show the insufficiency of learning of $\mathcal{OTF}_\Lambda$ in the exact learning model, by using the following lemma. $\mathcal{OTF}_\Lambda$.

Lemma 10. (László Lovász [8]) Let UT_n be the number of all rooted unordered trees with no edge labels, where the size is n. Then, $2^n < UT_n < 4^n$, where $n \geq 6$.

We denote by OT_n the set of all rooted ordered trees with no edge labels and the size n. From the above lemma, we have $OT_n \geq 2^n$, where $n \geq 6$. By Lemma 7 and Lemma 1 in another paper [4], we have the following Theorem 8.

Theorem 11. Any learning algorithm that exactly identifies all finite sets of term trees of size n by using restricted equivalence, membership, and subset queries must make $\Omega(2^n)$ queries in the worst case, where $n \geq 6$ and $|\Lambda| \geq 1$.

Proof. We denote by $\mathcal{S}_n$ the class of singleton sets of ground term trees of size n. The class $\mathcal{S}_n$ is a subclass of $\mathcal{OTF}_\Lambda$. For any L and L' in $\mathcal{S}_n$, $L \cap L' = \phi$. The empty set, however, is included in $\mathcal{OTF}_\Lambda$. Thus, by Lemma 7 and Lemma 1 in another paper [4], any learning algorithm that exactly identifies all finite sets of term trees of size n by using restricted equivalence, membership, and subset queries must make $\Omega(2^n)$ queries in the worst case, even when $|\Lambda| = 1$. □

6. Conclusions

We have studied the learnability of $\mathcal{OTF}_\Lambda$ in the exact learning model. In Section 4, we showed that any finite set $R_* \in \mathcal{OTF}_\Lambda$ is exactly identifiable by using at most $O(m_*^3 n^3)$ superset queries and at most $O(m_*)$ restricted equivalence queries, where $m_* = |R_*|$, n is the the maximum size of term trees in R_* and $|\Lambda|$ is infinite. In Section 5, we showed that it is hard to exactly identify any set in $\mathcal{OTF}_\Lambda$ efficiently by using restricted equivalence, membership, and subset queries.

We previously showed the learnabilities of $\mu\mathcal{OTT}_\Lambda$ and $\mu\mathcal{OTF}_\Lambda$ in the exact learning model [10),11)]. Suzuki, et al. [13] showed the learnability of $\mu\mathcal{OTT}_\Lambda$ in the framework of polynomial time inductive inference from positive data [3]. Thus, we will study the learnabilities of $\mu\mathcal{OTF}_\Lambda$ and $\mathcal{OTF}_\Lambda$ in the same framework. Table 1 summarizes our results and future works.

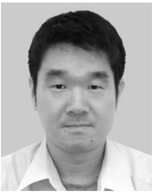
References

1) Amoth, T.R., Cull, P. and Tadepalli, P.: Exact learning of tree patterns from queries and counterexamples, *Proc. 11th Annual Workshop on Computational Learning Theory*, pp.175–186 (1998).
2) Amoth, T.R., Cull, P. and Tadepalli, P.: Exact learning of unordered tree patterns from queries, *Proc. 12th Annual Workshop on Computational Learning Theory*, pp.323–332 (1999).
3) Angluin, D.: Finding pattern common to a set of strings, *J. Comput. Syst. Sci.*, Vol.21, pp.46–62 (1980).
4) Angluin, D.: Queries and concept learning, *Machine Learning*, Vol.2, pp.319–342 (1988).
5) Arimura, H., Ishizaka, H. and Shinohara, T.: Learning unions of tree patterns using queries, *Theoretical Computer Science*, Vol.185, No.1, pp.47–62 (1997).
6) Arimura, H., Sakamoto, H. and Arikawa, S.: Efficient learning of semi-structured data from queries, *Proc. 12th Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence*, Vol.2225, pp.315–331 (2001).
7) Arimura, H., Shinohara, T. and Otsuki, S.: Polynomial time algorithm for finding finite unions of tree pattern languages, *Proc. 2nd International Workshop on Nonmonotonic and Inductive Inference, Lecture Notes in Artificial Intelligence*, Vol.659, pp.118–131 (1993).
8) László Lovász: *Combinatorial Problems and Exercises*, chapter Two classical enumeration problems in graph theory, North-Holland Publishing Company (1979).
9) Matsumoto, S. and Shinohara, A.: Learning pattern languages using queries, *Proc. 3rd European Conference on Computational Learning Theory*, pp.185–197 (1997).
10) Matsumoto, S., Shoudai, T., Miyahara, T. and Uchida, T.: Learning unions of term tree languages using queries, *Proc. LA Summer Symposium, July 2002*, pp.21–1–21–10 (2002).
11) Matsumoto, S., Shoudai, T., Uchida, T., Miyahara, T. and Suzuki, Y.: Learning of finite unions of tree patterns with internal structured variables from queries, *IEICE Trans*, Vol.E91-D, No.2, pp.222–230 (2008).
12) Miyahara, T., Suzuki, Y., Shoudai, T., Uchida, T., Takahashi, K. and Ueda, H.: Discovery of frequent tag tree patterns in semistructured web documents, *Proc. 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD-2002*)*, LNAI 2336*, pp.341–355, Springer-Verlag (2002).
13) Suzuki, Y., Akanuma, R., Shoudai, T., Miyahara, T. and Uchida, T.: Polynomial time inductive inference of ordered tree patterns with internal structured variables

from positive data, *Proc. 15th Annual Conference on Computational Learning Theory* (*COLT-2002*), *LNAI 2375*, pp.169–184, Springer-Verlag (2002).

(Received February 5, 2009)
(Revised March 26, 2009)
(Revised(2) June 11, 2009)
(Accepted July 21, 2009)

Satoshi Matsumoto is an associate professor of Department of Mathematical Sciences, Tokai University, Kanagawa, Japan. He received B.S. degree in Mathematics, 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.



Yusuke Suzuki received B.S. degree in Physics, 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.

Takayoshi Shoudai received B.S. in 1986, M.S. degrees in 1988 in Mathematics and Dr. Sci. in 1993 in Information Science all from Kyushu University. Currently, he is an associate professor of Department of Informatics, Kyushu University. His research interests include algorithmic graph theory, algorithmic learning theory, and data mining from graph-structured data. He is a member of IEICE and ACM.

Tetsuhiro Miyahara is an associate professor of Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. He received B.S. degree in Mathematics, 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.

Tomoyuki Uchida received B.S. degree in Mathematics, 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. He is a member of IEICE and ACM.

