数理解析研究所講究録 1325 巻 2003 年 69-74



# Polynomial Time Inductive Inference of Ordered Term Trees with Contractible Variables from Positive Data

鈴木祐介 (Yusuke Suzuki), 正代隆義 (Takayoshi Shoudai)
九州大学システム情報科学 { 府, 研究院 } 情報理学 { 専攻, 部門 }
Department of Informatics, Kyushu University
{y-suzuki,shoudai}@i.kyushu-u.ac.jp
松本哲志 (Satoshi Matsumoto)
東海大学 理学部 情報数理学科
Department of Mathematical Sciences, Tokai University
matumoto@ss.u-tokai.ac.jp
内田智之 (Tomoyuki Uchida), 宮原哲浩 (Tetsuhiro Miyahara)
広島市立大学 情報科学部
Faculty of Information Sciences, Hiroshima City University
{uchida@cs,miyahara@its}.hiroshima-cu.ac.jp

## 1 Introduction

Due to the rapid growth of Internet usage, tree structured data such as Web documents have been rapidly increasing. In order to analyze such tree structured data, efficient learning from tree structured data becomes more and more important. Tree structured data such as HTML/XML files are represented by rooted trees with ordered children and edge labels [1]. In order to represent a tree structured pattern common to such tree structured data, we proposed an *ordered term tree*, which is a rooted tree with ordered children and structured variables [7]. A variable can be substituted by an arbitrary tree. An ordered term tree $t$ is said to be *regular* if all variable labels in $t$ are mutually distinct. Many tree structured data such as HTML/XML files have no rigid structure and have essential information in subtrees containing leaves. In order to deal with such tree structured data, we introduce a new type of variable, called a *contractible variable*, which is regarded as an anonymous subtree in an ordered term tree and matches any subtree including a singleton vertex. A usual variable, called an *uncontractible variable*, in a term tree does not match a singleton vertex.

The *language* of a regular ordered term tree $t$ is the set of all ordered trees which are obtained from $t$ by substituting ordered trees for variables in $t$. The language of a regular ordered term tree $t$ shows the representing power of $t$. A *least generalized* regular ordered term tree $t$ explaining given tree structured data $S$ is a term tree $t$ whose language contains $S$ and is minimal. Consider the examples in Fig. 1. The term tree $t_2$ is a least generalized regular ordered term tree for $T_1$, $T_2$ and $T_3$. The ordered term tree $t_1$ also explains the three trees. But $t_1$ explains any tree with 2 or more vertices. So $t_1$ is overgeneralized and meaningless.

Let $\Lambda$ be a set of edge labels used in tree structured data. $\mathcal{OTT}^c_\Lambda$ denotes the set of all regular ordered term trees all of whose contractible variables are adjacent to leaves. First we give an efficient polynomial time algorithm for deciding whether or not a given regular ordered term tree in $\mathcal{OTT}^c_\Lambda$ matches a tree, where $|\Lambda| \geq 1$. Second when $|\Lambda| \geq 2$, we give a polynomial time algorithm for finding a least generalized regular ordered term tree in $\mathcal{OTT}^c_\Lambda$ which explains all given data. These results imply that the class $\mathcal{OTT}^c_\Lambda$ with $|\Lambda| \geq 2$ is polynomial time inductively inferable from positive data.

An ordered term tree is different from other representations of ordered tree structured patterns such as in [2, 4, 13] in that an ordered term tree has structured variables which can be substituted by arbitrary ordered trees. We showed that some classes of regular *unordered* term tree languages are polynomial time inductively inferable from positive data [5, 8, 9]. In [10, 12], we showed that some fundamental classes of regular ordered term tree languages without any contractible variable are polynomial time inductively inferable from positive data. In [6], we showed that some classes of regular ordered term tree languages without any contractible variable are exactly learnable in polynomial time using queries. In [7], we gave a data mining method from semistructured data using ordered term trees.

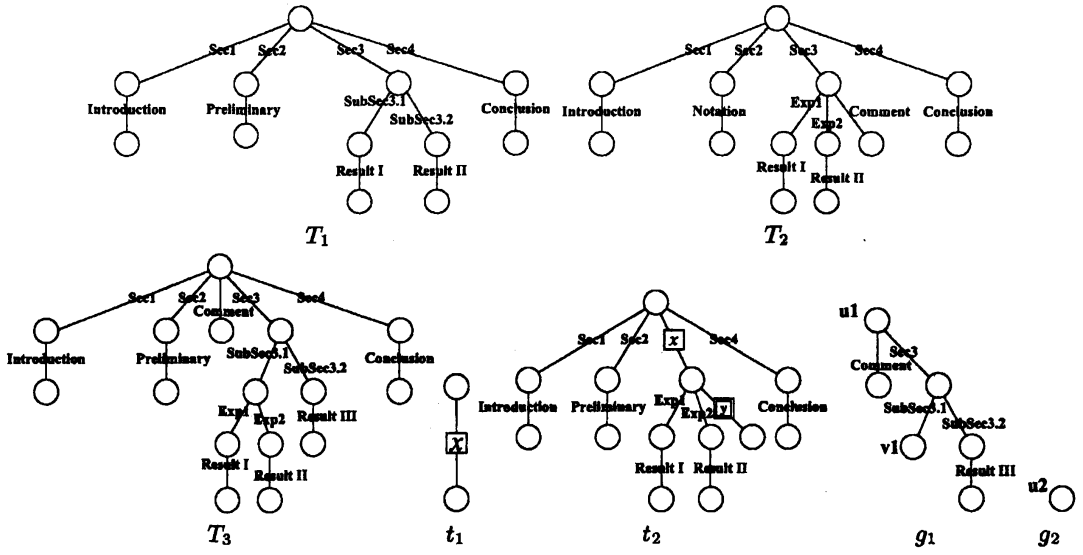



**Fig. 1.** Term trees $t_1$, $t_2$ and trees $T_1$, $T_2$ and $T_3$. An uncontractible (resp. contractible) variable is represented by a single (resp. double) lined box with lines to its elements. The label inside a box is the variable label of the variable.

## 2 Ordered Term Trees with Contractible Variables

Let $T = (V_T, E_T)$ be a rooted tree with ordered children (or simply a *tree*) which has a set $V_T$ of vertices and a set $E_T$ of edges. Let $E_g$ and $H_g$ be a partition of $E_T$, i.e., $E_g \cup H_g = E_T$ and $E_g \cap H_g = \emptyset$. And let $V_g = V_T$. A triplet $g = (V_g, E_g, H_g)$ is called a *term tree*, and elements in $V_g$, $E_g$ and $H_g$ are called a *vertex*, an *edge* and a *variable*, respectively. We assume that every edge and variable of a term tree is labeled with some words from specified languages. A label of a variable is called a *variable label*. $\Lambda$ and X denote a set of edge labels and a set of variable labels, respectively, where $\Lambda \cap X = \phi$. For a term tree $g$ 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 $g$ such that for any $j$ with $1 \leq j < i$, there exists an edge or a variable which consists of $v_j$ and $v_{j+1}$. If there is an edge or a variable which consists 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 use a notation $[v, v']$ to represent a variable $\{v, v'\} \in H_g$ such that $v$ is the parent of $v'$. Then we call $v$ the *parent port* of $[v, v']$ and $v'$ the *child port* of $[v, v']$.

**Definition 1.** Let $X^c$ be a distinguished subset of $X$. We call variable labels in $X^c$ ***contractible variable labels***. A contractible variable label can be attached to a variable whose child port is a leaf. We call a variable with a contractible variable label a **contractible variable**, which is allowed to substitute a tree with a singleton vertex, as stated later. We call a variable which is not a contractible variable an uncontractible variable. For a variable $[v, v']$, when we pay attention to the kind of the variable, we denote by $[v, v']^c$ and $[v, v']^u$ a contractible variable and an uncontractible variable, respectively.

A term tree $g$ is called *ordered* if every internal vertex $u$ in $g$ has a total ordering on all children of $u$. The ordering on the children of $u$ is denoted by $<_u^g$. An ordered term tree $g$ is called *regular* if all variables in $H_g$ have mutually distinct variable labels in $X$. For a set $S$, the number of elements in $S$ is denoted by $|S|$.

**Definition 2.** An ordered term tree with no variable is called a **ground ordered term tree**, which is a standard ordered tree. $\mathcal{OT}_\Lambda$ denotes the set of all ground ordered term trees whose edge labels are in $\Lambda$. $\mathcal{OTT}_\Lambda^c$ denotes the set of all ordered term trees with contractible or uncontractible variables whose edge labels are in $\Lambda$. In this paper, we treat only regular ordered term trees with contractible variables. Therefore we call it a **term tree** simply.

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 $\varphi$ from $V_f$ to $V_g$ such that (i) the root of $f$ is mapped to the root of $g$ by $\varphi$, (ii) $\{u, v\} \in E_f$ if and only if $\{\varphi(u), \varphi(v)\} \in E_g$ and the two edges have the same edge label, (iii) $[u, v] \in H_f$ if and only if $[\varphi(u), \varphi(v)] \in H_g$, in particular, $[u, v]^c \in H_f$ if and only if $[\varphi(u), \varphi(v)]^c \in H_g$, and (iv) for any internal vertex $u$ in $f$ which has more than one child, and for any two children $u'$ and $u''$ of $u$, $u' <_u^f u''$ if and only if $\varphi(u') <_{\varphi(u)}^g \varphi(u'')$.

Let $\sigma = [u, u']$ be a list of two 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 $x$. If $x$ is a contractible variable label in $X^c$, $g$ may be a tree with a singleton vertex $u$ and thus $\sigma = [u, u]$. It is the only case that a tree with a singleton vertex is allowed for a binding. A new term tree $f\{x := [g, \sigma]\}$ is obtained by applying the binding $x := [g, \sigma]$ to $f$ in the following way. Let $e = [v, v']$ be a variable in $f$ with the variable label $x$. Let $g'$ be one copy of $g$ and $w, w'$ the vertices of $g'$ corresponding to $u, u'$ of $g$, respectively. For the variable $e = [v, v']$, we attach $g'$ to $f$ by removing the variable $e$ from $H_f$ and by identifying the vertices $v, v'$ with the vertices $w, w'$ of $g'$, respectively. If $g$ is a tree with a singleton vertex, i.e., $u = u'$, then $v$ becomes identical to $v'$ after the binding. A *substitution* $\theta$ is a finite collection of bindings $\{x_1 := [g_1, \sigma_1], \cdots, x_n := [g_n, \sigma_n]\}$, where $x_i$'s are mutually distinct variable labels in $X$. The term tree $f\theta$, called the *instance* of $f$ by $\theta$, is obtained by applying the all bindings $x_i := [g_i, \sigma_i]$ on $f$ simultaneously. Further we define a new total ordering $<_v^{f\theta}$ on every vertex $v$ of $f\theta$ in a natural way. In this paper, we omit the exact definition of the ordering after applying a binding to a term tree. The readers can refer it to [10] or [12].

For example, let $t_2$ be a term tree described in Fig. 1 and $\theta = \{x := [g_1, [u_1, v_1]], y := [g_2, [u_2, u_2]]\}$ be a substitution, where $g_1$ and $g_2$ are trees in Fig. 1. Then the instance $t_2\theta$ of the term tree $t_2$ by $\theta$ is the tree $T_3$ in Fig. 1.

## 3 An Efficient Matching Algorithm for Term Trees

A matching algorithm for term trees is an algorithm which decides whether or not a term tree $t$ matches a tree $T$. We gave matching algorithms for term trees with no contractible variable in [10] and for term trees with variables having more than two child ports in [11]. In this section, we give a matching algorithm for $\mathcal{OTT}_A^c$ by extending the matching algorithm in [10].

Let $t = (V_t, E_t, H_t)$ and $T = (V_T, E_T)$ be a term tree in $\mathcal{OTT}_A^c$ and a tree in $\mathcal{OT}_A$, respectively. We assume that all vertices of a term tree $t$ are associated with mutually distinct numbers, called *vertex identifiers*. We denote by $I(u')$ the vertex identifier of $u' \in V_t$. A *correspondence-set* (C-set for short) is a set of vertex identifiers, which are with or without parentheses, of vertices of $t$. A vertex identifier with parentheses shows that the vertex is a child port of a variable.

Our matching algorithm proceeds by constructing C-sets for each vertex of a given tree $T$ in the bottom-up manner, that is, from the leaves to the root of $T$. At first, we construct the *C-set-attaching rule* of a vertex $u'$ of $t$ as follows. Let $c'_1, \cdots, c'_{m'}$ be all ordered children of $u'$. The C-set-attaching rule of $u'$ is of the form $I(u') \Leftarrow J(c'_1), \ldots, J(c'_{m'})$, where $J(c'_i) = I(c'_i)$ if $\{u', c'_i\}$ is an edge, $J(c'_i) = I(\emptyset)$ if $[u', c'_i]$ is a contractible variable, $J(c'_i) = (I(c'_i))$ otherwise. $I(\emptyset)$ is a special symbol which shows $c'_i$ is the child port of a contractible variable. The C-set-attaching rule of $t$, denoted by $Rule(t)$, is defined as follows.

$$\begin{aligned} Rule(t) = \ & \{I(u') \Leftarrow J(c'_1), \ldots, J(c'_{m'}) \mid \text{the C-set-attaching rule of all inner vertices}\} \\ & \cup \{(I(u')) \Leftarrow (I(u')) \mid u' \text{ is the child port of an uncontractible variable}\} \\ & \cup \{I(u') \Leftarrow I(u') \mid u' \text{ has just one child and connects to} \\ & \qquad\qquad \text{the child with a contractible variable}\}. \end{aligned}$$

The algorithm (Fig. 2) runs for $|\Lambda| = 1$. A matching algorithm for $|\Lambda| \geq 2$ can be easily constructed from this algorithm. The only work we have to do is to check whether or not edge labels of a tree is the same as corresponding edge labels of a term tree at the first foreach-loop of C-SET-ATTACHING. Then we can prove this theorem in a similar way to the proof of the correctness of the matching algorithm for a term tree [10].

**Theorem 1.** *Let $t$ be a term tree with $n$ vertices in $\mathcal{OTT}_A^c$ and $T$ a tree with $N$ vertices in $\mathcal{OT}_A$. The problem for deciding whether or not $t$ matches $T$ is solvable in $O(nN)$ time.*

**Procedure** MATCHING($t, T$);
**input** $t$: a term tree in $\mathcal{OTT}^c_\Lambda$ with root $r$, $T$: a tree in $\mathcal{OT}_\Lambda$ with root $R$;
**begin**
Construct $Rule(t)$;
**foreach** leaf $\ell$ of $T$ **do**
$CS(\ell) := \{I(\ell') \mid \ell'$ is a leaf of $t$ that is not a child port of a contractible variable,
or $\ell'$ has just one child and connects to it with a contractible variable$\}$;
**while** there is a vertex $v$ of $T$ s.t. $v$ has no C-set and all children of $v$ have C-sets
**do** C-Set-Attaching($v, Rule(t)$);
**if** $I(r) \in CS(R)$ **then** $t$ matches $T$ **else** $t$ does not match $T$
**end.**

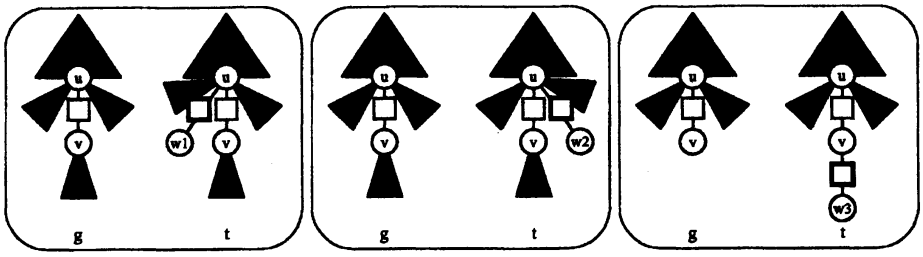
**Procedure** C-SET-ATTACHING($v, Rule(t)$);
**input** $v$: a vertex of $T$, $Rule(t)$: the C-set-attaching rule of $t$;
**begin**
$CS(v) := \emptyset$; Let $c_1, \cdots, c_m$ be all ordered children of $v$ in $T$;
**foreach** $I(u') \Leftarrow J(c'_1), \cdots, J(c'_{m'})$ in $Rule(t)$ **do**
**if** there is a sequence $0 = j_0 \leq j_1 \leq \cdots \leq j_i \leq \cdots \leq j_{m'-1} \leq j_{m'} = m$ s.t.
*1.* if $J(c'_i) = I(c'_i)$ then $j_i - j_{i-1} = 1$ and $I(c'_i) \in CS(c_{j_i})$,
*2.* if $J(c'_i) = (I(c'_i))$ then $CS(c_{k_i})$ has $I(c'_i)$ or $(I(c'_i))$ for some $k_i$ $(j_{i-1} < k_i \leq j_i)$
for all $i = 1, ..., m'$ // we have no condition on $j_i$ when $J(c'_i) = I(\emptyset)$.
**then** $CS(v) := CS(v) \cup \{(I(u')\}$;
**foreach** $(I(u')) \Leftarrow (I(u'))$ in $Rule(t)$ **do**
**if** there is a set in $CS(c_1), \cdots, CS(c_m)$ which has $I(u')$ or $(I(u'))$ **then** $CS(v) := CS(v) \cup \{(I(u')\}$;
**foreach** $I(u') \Leftarrow I(u')$ in $Rule(t)$ **do** $CS(v) := CS(v) \cup \{I(u')\}$
**end.**

**Fig. 2.** An algorithm for deciding whether or not a term tree $t \in \mathcal{OTT}^c_\Lambda$ matches a tree $T \in \mathcal{OT}_\Lambda$, where $|\Lambda| = 1$.

## 4 Algorithms for Finding a Least Generalized Term Tree

In this section, we present polynomial time algorithm for finding a least generalized term tree where $|\Lambda| \geq 2$, explaining given semistructured data. We can consider the language $L_\Lambda(t)$ to be the descriptive power of a term tree $t$. A *least generalized term tree* explaining a given set of trees $S \subseteq \mathcal{OT}_\Lambda$ is a term tree $t$ such that $S \subseteq L_\Lambda(t)$ and there is no term tree $t'$ satisfying that $S \subseteq L_\Lambda(t') \subsetneq L_\Lambda(t)$. The problem for finding a least generalized term tree for a given set of trees is discussed as the minimal language problem (MINL for short) in computational learning theory. The main algorithm is described in Fig. 4.

**Lemma 1.** *Let $g$ and $t$ be term trees in $\mathcal{OTT}^c_\Lambda$ for any $\Lambda$ which are described in one of Cases 1-3 of Fig. 3. Then $L_\Lambda(g) = L_\Lambda(t)$.*




*Case 1.* *Case 2.* *Case 3.*

**Fig. 3.** *Cases 1-3*: $g \not\equiv t$ and $L_\Lambda(g) = L_\Lambda(t)$ for $|\Lambda| \geq 2$. The parts $A, B, C, D$ of $g$ are the same as the corresponding parts $A, B, C, D$ of $t$.

**Algorithm** MINL($S$);
**input** $S = \{T_1, \ldots, T_m\} \subseteq \mathcal{OT}_\Lambda$: a set of trees;
**output** $t$: a least generalized term tree for $S$;
**begin**
  Let $\Lambda_S$ be the set of all edge labels which appear in $S$;
  $t := (\{u, v\}, \emptyset, \{[u, v]\})$; Let $q$ be a list initialized to be $[[u, v]]$;
  VARIABLE-EXTENSION($t, S, q$);
  Let $r_t$ be the root of $t$;
  EDGE-REPLACING($t, S, r_t$);
  **output** $t$
**end.**

**Procedure** VARIABLE-EXTENSION($t_{input}, S, q$);
**input** $t_{input}$: a term tree, $S$: a set of trees, $q$: a queue of variables;
**begin**
  $t := t_{input}$;
  **while** $q$ is not empty **do begin**
    $[u, v] := q[1]$; Let $w_1, w_2$, and $w_3$ be new vertices;
    // $w_1$ becomes a vertex between $u$ and $v$.
    $t' := (V_t \cup \{w_1\}, E_t, H_t \cup \{[u, w_1], [w_1, v]\} - \{[u, v]\})$;
    **if** $S \subseteq L_\Lambda(t')$ **then begin** $t := t'$; $q := q\&[[w_1, v]]$; **continue end else** $q := q[2..]$;
    // $w_2$ and $w_3$ become the previous and next siblings of $v$, respectively.
    $t' := (V_t \cup \{w_2\}, E_t, H_t \cup \{[u, w_2]\})$;
    **if** $S \subseteq L_\Lambda(t')$ **then begin** $t := t'$; $q := q\&[[u, w_2]]$ **end**;
    $t' := (V_t \cup \{w_3\}, E_t, H_t \cup \{[u, w_3]\})$;
    **if** $S \subseteq L_\Lambda(t')$ **then begin** $t := t'$; $q := q\&[[u, w_3]]$ **end**;
  **end**;
  **return** $t$
**end**;

**Procedure** EDGE-REPLACING($t_{input}, S, u$);
**input** $t_{input}$: a term tree, $S$: a set of trees, $u$: a vertex;
**begin**
  **if** $u$ is a leaf **then return**;
  $t := t_{input}$; Let $v_1, \ldots, v_k$ be the children of $u$;
  **for** $i := 1$ **to** $k$ **do** LABELED-EDGE-REPLACING($t, S, v_i$);
  **for** $i := 1$ **to** $k$ **do**
    **foreach** edge label $\lambda \in \Lambda_S$ **do**
      // Let $\{u, v_i\}$ be an edge with label $\lambda$ and $w_1$, $w_2$, and $w_3$ new vertices. $w_1$ and $w_2$ become
      // the previous and next siblings of $v_i$, respectively, and if $v_i$ is a leaf, $w_3$ becomes a child of $v_i$
      **if** $v_i$ is a leaf **then begin**
        $t' := (V_t \cup \{w_1, w_2, w_3\}, E_t \cup \{\{u, v_i\}\}, H_t \cup \{[u, w_1]^c, [u, w_2]^c, [v_i, w_3]^c\} - \{[u, v_i]\})$;
        **if** $S \subseteq L_\Lambda(t')$ **then begin**
          $t_1 := (V_{t'} - \{w_1\}, E_{t'}, H_{t'} - \{[u, w_1]^c\})$; **if** $S \subseteq L_\Lambda(t_1)$ **then** $t' := t_1$;
          $t_2 := (V_{t'} - \{w_2\}, E_{t'}, H_{t'} - \{[u, w_2]^c\})$; **if** $S \subseteq L_\Lambda(t_2)$ **then** $t' := t_2$;
          $t_3 := (V_{t'} - \{w_3\}, E_{t'}, H_{t'} - \{[v_i, w_3]^c\})$; **if** $S \subseteq L_\Lambda(t_3)$ **then** $t' := t_3$;
          $t := t'$; **continue**
        **end**
      **end else begin**
        $t' = (V_t \cup \{w_1, w_2\}, E_t \cup \{\{u, v_i\}\}, H_t \cup \{[u, w_1]^c, [u, w_2]^c\} - \{[u, v_i]\})$;
        **if** $S \subseteq L_\Lambda(t')$ **then begin**
          $t_1 := (V_{t'} - \{w_1\}, E_{t'}, H_{t'} - \{[u, w_1]^c\})$; **if** $S \subseteq L_\Lambda(t_1)$ **then** $t' := t_1$;
          $t_2 := (V_{t'} - \{w_2\}, E_{t'}, H_{t'} - \{[u, w_2]^c\})$; **if** $S \subseteq L_\Lambda(t_2)$ **then** $t' := t_2$;
          $t := t'$; **continue**
        **end**
      **end**;
  **return** $t$
**end**;

**Fig. 4.** Algorithm MINL

**Definition 3.** Let $t$ be a term tree in $\mathcal{OTT}^c_\Lambda$ for $|\Lambda| > 2$. The term tree $t$ is said to be a *canonical term tree* if $t$ has no combination of variables and edges of the right term trees $t$ described in *Cases 1-3*. For an arbitrary term tree $t$, we can transform $t$ into the canonical term tree $g$ such that $L_\Lambda(g) = L_\Lambda(t)$ by transforming the right trees in *Cases 1-3* into the left trees.

**Lemma 2.** *Let $g$ and $t$ be canonical term trees in $\mathcal{OTT}^c_\Lambda$ for any $\Lambda$. If $L_\Lambda(g) \subseteq L_\Lambda(t)$ then there exists a substitution $\theta$ such that $g \equiv t\theta$.*

The procedure VARIABLE-EXTENSION extends a term tree by adding uncontractible variables as much as possible. A term tree outputted by VARIABLE-EXTENSION is a least generalized term tree for $S$ consisting of only uncontractible variables. Then the procedure EDGE-REPLACING tries to replace variables with labeled edges from leaves to the root. Since the output term tree is not larger than the largest tree in $S$, we have the following lemma.

**Lemma 3.** *Let $t$ be the output of the algorithm* MINL *for an input $S$. Let $t'$ be a term tree satisfying that $S \subseteq L_\Lambda(t') \subseteq L_\Lambda(t)$. Let $g$ and $g'$ be the canonical term trees such that $L_\Lambda(g) = L_\Lambda(t)$ and $L_\Lambda(g') = L_\Lambda(t')$, respectively. Then $g \equiv g'$.*

**Theorem 2.** *Let $\Lambda$ be a set of edge labels where $|\Lambda| \geq 2$. The algorithm* MINL *finds a least generalized term tree in $\mathcal{OTT}^c_\Lambda$ for a given set of trees in $\mathcal{OT}_\Lambda$ in polynomial time.*

In this paper, we have presented polynomial time algorithms for solving the membership and MINL problems for the class of labeled term trees. From these algorithms and Angluin's theorem [3], we can show that the class is polynomial time inductively inferable from positive data.

## References


1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, 2000.
2. T. R. Amoth, P. Cull, and P. Tadepalli. Exact learning of unordered tree patterns from queries. *Proc. COLT-99, ACM Press*, pages 323–332, 1999.
3. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Science*, 21:46–62, 1980.
4. H. Arimura, H. Sakamoto, and S. Arikawa. Efficient learning of semi-structured data from queries. *Proc. ALT-2001, Springer-Verlag, LNAI 2225*, pages 315–331, 2001.
5. S. Matsumoto, Y. Hayashi, and T. Shoudai. Polynomial time inductive inference of regular term tree languages from positive data. *Proc. ALT-97, Springer-Verlag, LNAI 1316*, pages 212–227, 1997.
6. S. Matsumoto, T. Shoudai, T. Miyahara, and T. Uchida. Learning of finite unions of tree patterns with internal structured variables from queries. *Proc. AI-2002, Springer-Verlag, LNAI 2557*, pages 523–534, 2002.
7. T. Miyahara, Y. Suzuki, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Discovery of frequent tag tree patterns in semistructured web documents. *Proc. PAKDD-2002, Springer-Verlag, LNAI 2336*, pages 341–355, 2002.
8. T. Shoudai, T. Miyahara, T. Uchida, and S. Matsumoto. Inductive inference of regular term tree languages and its application to knowledge discovery. *Information Modelling and Knowledge Bases XI, IOS Press*, pages 85–102, 2000.
9. T. Shoudai, T. Uchida, and T. Miyahara. Polynomial time algorithms for finding unordered tree patterns with internal variables. *Proc. FCT-2001, Springer-Verlag, LNCS 2138*, pages 335–346, 2001.
10. Y. Suzuki, R. Akanuma, T. Shoudai, T. Miyahara, and T. Uchida. Polynomial time inductive inference of ordered tree patterns with internal structured variables from positive data. *Proc. COLT-2002, Springer-Verlag, LNAI 2375*, pages 169–184, 2002.
11. Y. Suzuki, T. Shoudai, T. Miyahara, and T. Uchida. A polynomial time matching algorithm of structured ordered tree patterns for data mining from semistructured data. *Proc. ILP-2002, Springer-Verlag, LNAI 2583*, pages 270–284, 2003.
12. Y. Suzuki, T. Shoudai, T. Uchida, and T. Miyahara. Ordered term tree languages which are polynomial time inductively inferable from positive data. *Proc. ALT-2002, Springer-Verlag, LNAI 2533*, pages 188–202, 2002.
13. K. Wang and H. Liu. Discovering structural association of semistructured data. *IEEE Trans. Knowledge and Data Engineering*, 12:353–371, 2000.