Identification of Tree Translation Rules from Examples

Sakamoto, Hiroshi
Department of Informatics, Kyushu University

Arimura, Hiroki
Department of Informatics, Kyushu University

Arikawa, Setsuo
Department of Informatics, Kyushu University

出版情報：DOI Technical Report. 173, 2000-04-21. Department of Informatics, Kyushu University
バージョン：
権利関係：

DOI-TR-173

DOI Technical Report

Identification of Tree Translation Rules from Examples

by

H. Sakamoto, H. Arimura, and S. Arikawa

April 21, 2000

Department of Informatics
Kyushu University
Fukuoka 812-81, Japan

Email: hiroshi@i.kyushu-u.ac.jp Phone: +81-92-642-2693

Identification of Tree Translation Rules from Examples

Hiroshi Sakamoto, Hiroki Arimura, and Setsuo Arikawa
{hiroshi, arim, arikawa}@i.kyushu-u.ac.jp
Department of Informatics, Kyushu University
Hakozaki 6-10-1, Higashi-ku, Fukuoka-shi 812-8581, Japan
Phone: +81-92-642-2693, Fax: -2698

Abstract

Two models for simple translation between ordered trees are introduced. First is that output is obtained from input by renaming labels and deleting nodes. Several decision problems on the translation are proved to be tractable and intractable. Second is term rewriting system, called *k-variable linear translation*. The efficient learnability of this system using membership and equivalence queries is shown.

1 Introduction

By rapid progress of network environment, the use of a lot of text data in the form of HTML and XML is available. Such texts are called *markup texts* which are expressed by a rooted ordered tree. A root is the unique node of the meaning which points to the whole document, the other internal nodes represent many tags, and the leaves represent the contents of the document and attributes of tags. For example, `\chapter`{Introduction} denotes that the tag "chapter" has a child "Introduction" as its attribute.

We can define various type of tags in the *document type definition* of XML and the result is that a special meaning is added to a part of document. These tags are useful in order to take the part out from a large text. However, these tags are locally defined and other user may define these tags by other names. Then, compared with relational or object databases, the distributed documents among network are heterogeneous.

Therefore, it is necessary to translate the structures of documents (semi-)automatically in order to exchange data through network. The applications of the translation are widely found, e.g., Web browsing using XML-to-HTML translation like XSLT [4], query language such as XQL using XML database, and distributed database such as digital library using wrapper generator.

There are two types of data exchange models considered in this paper. One is *extraction* and another is *reconstruction*. The extraction is a very simple translation $T \to t$ such that a small tree t is obtained by only (1) renaming labels of T or (2) deleting nodes of T. This model is suitable for the situation that a user takes out specific entries from a very large table as a small table, or renaming a specific tag without changing the structure of the document.

complicated transformation of trees so that exchanging any two subtrees of an input tree and renaming labels depending on ancestors or descendants of the current node. For example, it is possible to change the order of title and author in digital books card. This translation can not be defined by the erasing homomorphism because the order of any two node must be preserved. However, an erasing homomorphism also can not be defined by the term rewriting, e.g., deleting any tag of subsection and making its children become the children of the parent of the tag. This operation is called *embedding* in graph theory. Thus, one model does not properly contain the other model.

This paper is organized as follows. In the next section, we give the formal definitions of the erasing homomorphism and term rewriting system. We also define the problem of identifying translation rules for given pairs of trees as examples with respect to the both models. In particular, single example is given in case of the erasing homomorphism, that is this problem is a decision problem. In case of the term rewriting system, we define the class of k *variable linear translation system* in which any rule contains at most k variables and no variable appears in a term twice. These restrictions guarantee the termination and confluence of the rewriting system.

In Section 3, the complexity of announced decision problem is considered. We first deal with a subclass of erasing homomorphism, called *erasing isomorphism*. This problem contains the tree inclusion problem [9] as a special case. The first result is that the hardness of this problem is not spoiled even if the given trees are in depth 1, that is strings. It is open whether this problem is in P, but we show that a nontrivial subproblem is in P. Moreover, we prove the NP-completeness of erasing homomorphism with respect to the restrictions either given trees are strings or output tree is labeled by a single alphabet.

In Section 4, the learning problem of linear translation system from membership and equivalence queries [1] are considered. The hypothesis space is the class of translation systems and the target class is the class of k-variable linear translation systems. A counterexample for a hypothesis is an ordered pair (t, t') of trees such that exactly one of the hypothesis and the target can translate t to t. We present a learning algorithm based on the theory of [2, 3] and we show that our algorithm identifies each target using at most $O(m)$ equivalence queries and at most $O(kn^{2k})$ membership queries, where m is the number of rules of the target and n is the number of nodes of counterexamples.

2 Tree Translation Models

2.1 Erasing homomorphisms

In this subsection, we introduce a very simple class of tree translations, called erasing homomorphisms, as a formal model of data extraction from semi-structured data. In Section 3, we will study the identification problem for erasing isomorphisms from given examples.

A tree is a connected, acyclic, directed graph. A *rooted tree* is a tree in which one of the vertices is distinguished from the others and is called the root. We refer to a vertex of a rooted tree as a *node* of the tree. An *ordered tree* is a rooted tree in which the children of each node are ordered. That is, if a node has k children, then we can designate them as the first child, the second child, and so on up to the k-th child.

The $node(T)$ denotes the set of nodes of tree T, $|T|$ denotes the number of nodes of T for each $i \in node(T)$, let $\ell(i)$ denote the label of i. An alphabet Σ is a set of symbols. In this paper, we use Σ as the set of labels for trees. A labeled tree T is considered as a pair of a graph Sk and a mapping h from $node(Sk)$ to Σ such that $h(i) = A$ iff $\ell(i) = A$ for each $i \in node(Sk) = node(T)$. We call the tree Sk a *skeleton* of T and the skeleton of a tree T is denoted by $Sk(T)$. Let T/n be a subtree of T whose root is $n \in node(T)$.

Suppose each label is a symbol in an alphabet Σ. Let λ denote the unique null symbol not in Σ. We define two operations on tree T. One is *renaming*, denoted by $\mathtt{a} \to \mathtt{b}$, to replace all labels $\mathtt{a}$ in T by $\mathtt{b}$. Another is *deleting*, denoted by $\mathtt{a} \to \lambda$, to remove any node n for $\ell(n) = \mathtt{a}$ in T and make the children of n become the children of the parent of n.

Let $S = \{\mathtt{a} \to \mathtt{b} \mid \mathtt{a} \in \Sigma, \mathtt{b} \in \Sigma \cup \{\lambda\}\}$ be a set of operations. Then, we write $T \to_S T'$ iff T' is obtained by applying all operations in S to T simultaneously.

Definition 2.1 Let (T, P) be a pair of trees over an alphabet Σ. Then, the problem of *erasing homomorphism* is to decode whether there exists a set S of operations such that $T \to_S P$. The input tree T is called *target* and P *pattern*. This problem is denoted by $EHP(T, P)$.

Definition 2.2 The problem of *erasing isomorphism*, denoted by $EIP(T, P)$ is to decide whether $T \to_S P$ such that if $\mathtt{a} \to \mathtt{b} \in S$, then $\mathtt{a} \to \mathtt{c} \notin S$ for all $\mathtt{c} \neq \mathtt{b}$.

There is other restriction for these problems. Let $EHP(T, P)_k$ and $EIP(T, P)_k$ denote the problems that the depth of the input tree T is bounded by k, where the depth of T is the length of the longest path of T. In particular, a string is a tree with depth 1.

When we consider the restriction that any two nodes of a pattern tree P are labeled by distinct symbols, this problem is the special case of $EIP(T, P)$. Moreover, this problem is equivalent to the tree inclusion problem [9] which is decidable in $O(|T| \cdot |P|)$ time.

2.2 Tree translation systems

In this subsection, we introduce a formal model of data reconstruction for semi-structured data, called tree translation systems, which is more expressive than the class of erasing isomorphism in the last subsection. In Section 4, we will consider the identification problem for tree translation systems in an interactive setting.

First, we introduce the class of *ranked trees* whose node label is ranked and the out-degree of a node is bounded by the rank of its node label, where we do not allow any

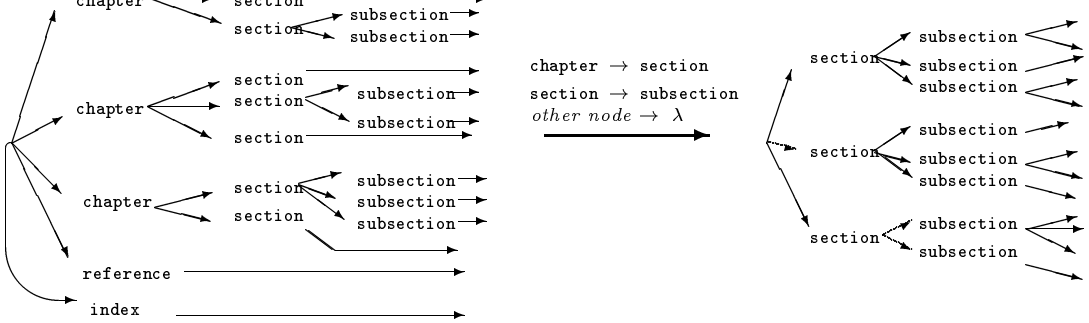

Figure 1: A translation by erasing isomorphism

operations such as deletion and insertion that may change the out-degree of a node. Let $\Sigma = \cup_{n \geq 0} \Sigma_n$ be a finite *ranked alphabet* of *function symbols*, where for each $f \in \Sigma$, a nonnegative integer $arity(f) \geq 0$, called *arity*, is associated. We assume that Σ contains at least one symbol of arity zero. Let X be a countable set of *variables* disjoint with Σ, where we assume that each $x \in X$ has arity zero.

Definition 2.3 We denote by $\mathcal{T}(\Sigma, X)$ the set of all labeled, rooted, ordered trees t such that

- Each node v of t is labeled with a symbol in $\Sigma \cup X$, denoted by $t(v)$.
- If $t(v)$ is a function symbol $f \in \Sigma$ of arity $k \geq 0$ then v has exactly k children.
- If $t(v)$ is a variable $x \in X$ then v is a leaf.

We call each element $t \in \mathcal{T}(\Sigma, X)$ a *pattern tree* (*pattern* for short).

A pattern tree is also called a first-order term in formal logic. We often write $\mathcal{T}$ by omitting Σ and X if they are clearly understood from context. For pattern t, we denote the set of variables appearing in t by $var(t) \subseteq X$ and define the number of the nodes of t by $size(t)$. A pattern t is said to be a *ground pattern* if it contains no variables.

A *position* in pattern t is simply a node of t. For pattern t, we denote by $occ(t)$ the set of all positions in t. If there is a downward path from a position α to another position w in t, we say that either α is *above* β or β is *below* α, and write $\alpha \geq \beta$. If $\alpha \geq \beta$ but $\beta \not\geq \alpha$ then we say that α is *strictly above* β, β is *strictly below* α and write $\alpha > \beta$. For pattern t and position $\alpha \in occ(t)$, the *subpattern* appearing at α, denoted by t/α, is the labeled subtree of t whose root is located at α.

A pattern t is called *linear* if any variable $x \in X$ appears in t at most once. A pattern t is *of k-variable* if $var(t) = \{x_1, \ldots, x_k\}$. For $k \geq 0$, we use the notation $t[x_1, \ldots, x_k]$ to indicate that pattern t is a k-variable linear pattern with mutually distinct variables $x_1, \ldots, x_k \in X$, where the order of variable in t is arbitrary. For k-variable linear pattern

Now, we introduce tree translation systems.

Definition 2.4 A *tree translation rule* (*rule* for short) is an ordered pair $(p, q) \in \mathcal{T} \times \mathcal{T}$ such that $var(p) \supseteq var(q)$. We also write $(p \to q)$ for rule (p, q). A *tree translation system* (TT) is a set H of translation rules.

Definition 2.5 A translation rule $C = (p, q)$ is *of k-variable* if $card(var(C)) \leq k$, and *linear* if both of p and q are linear.

For every $k \geq 0$, we denote by $LR(k)$ and $LTT(k)$ the classes of all k-variable linear translation rules, and all k-variable linear tree translation systems (LTT(k)), respectively. We also denote by $LTT = \cup_{k \geq 0} LTT(k)$ all linear tree translation systems.

Definition 2.6 Let $H \in LLT$ be a linear translation system. The *translation relation defined by* H with the set $M(H) \subseteq \mathcal{T} \times \mathcal{T}$ is defined recursively as follows.

- *Identity*: For every pattern $p \in \mathcal{T}$, $(p, p) \in M(H)$.
- *Congruence*: If $f \in \Sigma$ is a function symbol of arity $k \geq 0$ and $(p_i, q_i) \in M(H)$ for every i then $(f(p_1, \ldots, p_k), f(q_1, \ldots, q_k)) \in M(H)$.
- *Application*: If $(l[x_1, \ldots, x_l], r[x_1, \ldots, x_l]) \in H$ is a k-variable linear rule in H, where $0 \leq l \leq k$, and $(p_i, q_i) \in M(H)$ for every i then $(l[p_1, \ldots, p_k], r[q_1, \ldots, q_k]) \in M(H)$.

If $C \in M(H)$ then we say that rule C is *derived by* H. The definition of the meaning $M(H)$ above corresponds to the computation of top-down tree transducer [6] or the a special case of term rewriting relation [5] where only top-down rewriting are allowed.

Lemma 2.1 *Given pair $C \in \mathcal{T} \times \mathcal{T}$ and $LTT(k)$ H, the problem of deciding the membership $C \in M(H)$ can be computed in $O(mn^5)$ time, where $m = card(H)$ and $n = size(C)$.*

Proof: By using dynamic programming, we can compute all position pairs $\pi \in occ(C)$ such that $C/\pi \in M(H)$ with the time complexity stated above. □

In what follows, we will normally denote patterns by letters p, q, s and t, translation rules by capital letters C, D, and translation systems by capital letter H, possibly subscripted. Now, we extend the notions of sizes, the set of variables, positions, and subpatterns for rules as follows. For rule $C = (p, q)$, we define $size(C) = size(p) + size(q)$, $var(C) = var(p) \cup var(q)$.

A *position pair* in rule $C = (p_1, p_2)$ is any pair (α_1, α_2) such that $\alpha_i \in occ(p_i)$ for every $i = 1, 2$. We denote the set of position pairs in C by $occ(C) = occ(p) \times occ(q)$. For position pairs $\pi = (\alpha_1, \alpha_2)$ and $\tau = (\beta_1, \beta_2)$, we extend $\geq$ by $\pi \geq \tau$ iff $\alpha_i \geq \beta_i$ for every $i = 1, 2$.

a subrule D of C at π is said to be *smaller than* subrule E of C at τ if $\pi \leq \tau$ holds.

Definition 2.7 Let $C = (l[x_1, \ldots, x_l], r[x_1, \ldots, x_l])$ be a k-variable linear rule and $D \in \mathcal{T} \times \mathcal{T}$ be a rule. If there exist some $(p_i, q_i) \in M(H_*)$ for every i such that $D = (l[p_1, \ldots, p_k], r[q_1, \ldots, q_k])$ then we say that C *covers* D relative to H_* and write $C \succeq_{H_*} D$.

3 Deciding Translation Rules by Single Examples

In this section, we consider the erasing homomorphism and the erasing isomorphism problem. First, the $EIP(T, P)$ is considered. This problem is a sub-problem of the $EHP(T, P)$. Since all these problems are clearly in NP, our interest is in the point that which problem is in P. The following result tells us that the restriction for trees to be strings does not make $EIP(T, P)$ easy.

Theorem 3.1 $EIP(T, P)$ is polynomial time reducible to $EIP(T, P)_1$.

Proof: The reduction is constructed as follows. Let $node(T) = \{0, 1, \ldots, n\}$, where 0 is the root. For each $0 \leq i \leq n$, compute n strings such that $\ell(i) = \mathtt{A}$ iff $\mathtt{a_i A a_i A a_i} = (\mathtt{a_i A})^2 \mathtt{a_i}$, where let $\{\ell(0), \ldots, \ell(n)\} \cap \{a_0, \ldots, a_n\} = \emptyset$.

Next, construct the string $t(0)$ from T recursively as follows.

1. For each leaf node i of T, let $t(i) = (\mathtt{a_i}\ell(\mathtt{i}))^2 \mathtt{a_i} \equiv \mathtt{a_i}\ell(\mathtt{i})\mathtt{a_i}\ell(\mathtt{i})\mathtt{a_i}$.
2. For each internal node i, let $t(i) = (\mathtt{a_i}\ell(\mathtt{i}))^2 \cdot t(j_i) \cdots t(j_k) \cdot \mathtt{a_i}$, where $j_i, \ldots, j_k$ are the children of i.

Similarly, compute the string $p(0)$ from the pattern P. The $(t(0), p(0))$ is log-space reducible.

Let $\mathcal{T}_\Sigma$ be the set of trees over an alphabet Σ, and $t(\mathcal{T}_\Sigma)$ be the set of all strings obtained by the above construction for $T \in \mathcal{T}_\Sigma$. We first show that there exists a one-to-one mapping h between $\mathcal{T}_\Sigma$ and $t(\mathcal{T}_\Sigma)$. Since the string $t(0)$ is uniquely obtained from T, it is sufficient to show that $T \neq T'$ implies $t(0) \neq t'(0)$.

In case of $|T| \neq |T'|$, clearly $|t(0)| \neq |t'(0)|$ because $|T(0) = 5n$ for $n = |T|$. In case of $Sk(T) \neq Sk(T')$, there exist $i, j \in node(T)$ such that j is a child of i in T but not in T'. Then, $t(0)$ contains a string of the form $(\mathtt{a_i}\ell(\mathtt{i}))^2 \cdots t(j) \cdots \mathtt{a_i}$. However, $t'(0)$ contains $(\mathtt{a_i}\ell(\mathtt{i}))^2 \alpha \mathtt{a_i}$ and α does not contain $t'(j)$. Thus, $t(0) \neq t'(0)$.

In case of $Sk(T) = Sk(T')$ and $T = T'$, there exists $i \in node(T)$ such that $\ell(i)$ of T is not equal to that of T'. Then, $t(i) \neq t'(i)$ and the occurrence of $t(i)$ in $t(0)$ is equal to that of $t'(i)$ in $t'(0)$. Since $|t(0)| = |t'(0)|$, it holds $t(0) \neq t'(0)$. Thus, there exists such a one-to-one mapping h.

$Sk(T') = Sk(P)$, $T \to_{S_\lambda} T'$, and $T' \to_{S_\Sigma} P$.

On the other hand, $t(0) \to_{S'} t'$, where $S' = S_\lambda \cup \{\mathtt{a} \to \lambda \mid \mathtt{A} \to \lambda \in S_\lambda, t(0) \text{ has } \mathtt{aAaA}\}$. Since $t(0)$ contains $\mathtt{a_i A a_i} \cdots t(k) \cdot \mathtt{a_i}$ iff the node i of T labeled by $\mathtt{A}$, $t(0) \to_{S'} t'$ corresponds to that $T \to_{S_\lambda} T'$, that is $t' = t'(0)$. Moreover, $t'(0) \to_{S_\Sigma} p(0)$. Thus, $T \to_S P$ implies $t(0) \to_{S_h} p(0)$ for $S_h = S' \cup S_\lambda$.

Suppose the contrary that $t(0) \to p(0)$. Let $t(0) \to_{S_\lambda} t'(0)$, where $S_\lambda = S_1 \cup S_2$, $S_1 = \{\mathtt{A} \to \lambda \mid \mathtt{A} \in \Sigma\}$ and $S_2 = \{\mathtt{a} \to \lambda \mid \mathtt{a} \notin \Sigma\}$. And let $t'(0) \to_{S'} p(0)$, where $S' = \{\mathtt{a} \to \mathtt{b} \mid \mathtt{b} \neq \lambda\}$.

Assume that $t(0)$ contains a sting of the form $\mathtt{aAaA}$ and exactly one of $\mathtt{A} \to \lambda \in S_1$ and $\mathtt{a} \to \lambda \in S_2$ is satisfied. Then, $t'(0)$ contains $\mathtt{aa}$ or $\mathtt{AA}$. However, by the definition of the string $p(0)$, it contains no square of a symbol. It is a contradiction for $t'(0) \to p(0)$ via S' because S' contains no deleting operation. Thus, we can suppose that for any $\mathtt{aAaA}$ contained in $t(0)$, $\mathtt{A} \to \lambda, \mathtt{a} \to \lambda \in S_\lambda$. Hence, $\mathtt{A} \to \lambda \in S_1$, $\mathtt{a_i} \to \lambda \in S_2$, and $t(0)$ contains $(\mathtt{a_i A})^2$ iff $i \in node(T)$ and $\ell(i) = A$. It follows that $T \to_{S_1} T'$. Since S' maps any symbol not in Σ to itself, $t'(0) \to_{S'} p(0)$ implies $T' \to_{S'} P$. Then, $T \to_{S_\lambda \cup S'} P$.

Therefore, $EIP(T, P)$ is reducible to $EIP(t(0), p(0))$, that is $EIP(T, P)$ is polynomial time reducible to $EIP(T, P)_1$. □

Let $w = \mathtt{a_1 a_2} \cdots \mathtt{a_n} \in \Sigma^n$. The i-th symbol of w is denoted by $w[i]$. The substring $\mathtt{a_i a_{i+1}} \cdots \mathtt{a_j}$ of w is denoted by $w[i,\ j]$. An *occurrence* of a string α on w is a number i such that $w[i,\ i + |\alpha| - 1] = \alpha$. The number of occurrences of α in w is denoted by $\sharp(\alpha, w)$. The set of all occurrences of α in w is denoted by $occ(\alpha, w)$.

Let w, α, β be strings. There exists an *overlap* of α and β on w if there exist occurrences i and j of α and β on w such that $i < j < |\alpha| + i - 1$ or $j < i < |\beta| + j - 1$.

If a string is of the form $\mathtt{A}\alpha\mathtt{A}$ for some $\mathtt{A} \in \Sigma$ and $\sharp(\mathtt{A}, \alpha) = 0$, then we call the string an *interval* of $\mathtt{A}$. A string $w \in \Sigma^*$ is called k-interval free if w contains an overlap of at most $(k-1)$ intervals.

A string $w \in \Sigma^*$ is said to have a *split* if there exists an $1 \leq i \leq |w| - 1$ such that $[pre(i)_w] \cap [suf(i)_w] = \emptyset$, where $pre(i)_w$ ($suf(i)_w$) is the prefix (suffix) of w in length i and $[w]$ is the set of symbols in w.

Definition 3.1 The problem $EIP(T, P)_1$ is denoted $EIP(T, P)_1^k$ if if T and P are both k-interval free.

Lemma 3.1 *If T and P are 3-interval free and have no split, then $EIP(T, P)_1^3$ is decidable in polynomial time in $|T|$ and $|P|$.*

Proof: For given T, define $\langle T \rangle_i$ recursively as follows.

1. Let $\langle T \rangle_1 = occ(T[1], T)$.

The sequence $\langle T\rangle_1, \ldots, \langle T\rangle_i$ is uniquely decided because if both $occ(\mathtt{a}, T)$ and $occ(\mathtt{a}', T)$ satisfy the condition for $\langle T\rangle_{i+1}$, then either an $\mathtt{a}$ contained in intervals of $\mathtt{a_i}$ and $\mathtt{a}'$ or an $\mathtt{a}'$ contained in intervals of $\mathtt{a_i}$ and $\mathtt{a}$. This is a contradiction for that T is 3-interval tree. Moreover, since T has no split, if $suf(1)_T = \mathtt{a}$, then the end of the sequence must be $\langle T\rangle_i = occ(\mathtt{a}, T)$.

All symbols appearing in T at least two times are classified into three categories. First is the above $\langle T\rangle_i$. We can compute the sets of all $\langle T\rangle_i$ and $\langle P\rangle_{i'}$, denoted by $\langle T\rangle$ and $\langle P\rangle$ in $O(|T|^2 + |P|^2)$ time.

Second is $[\langle T\rangle_i]$ which is a set of $occ(\mathtt{a}, T)$ such that at least two intervals of $\mathtt{a_i}$ for $occ(\mathtt{a_i}, T) \in \langle T\rangle$ contains $\mathtt{a}$ and $occ(\mathtt{a}, T) \notin \langle T\rangle$. We can compute all $[\langle T\rangle_i]$ and $[\langle P\rangle_{i'}]$ in $O(|T| + |P|)$ time.

Third is $[\langle T\rangle_i]_j$ which is the sequence of $occ(\mathtt{a}, T)$ such that $\sharp(\mathtt{a}, T) \geq 2$ and exactly one interval of $\mathtt{a_i}$ for $occ(\mathtt{a_i}, T) \in \langle T\rangle$ contains $\mathtt{a}$. We can compute $[\langle T\rangle_i]_j$ and $[\langle P\rangle_{i'}]_{j'}$ in $O(|T| + |P|)$ time.

Let $T \rightarrow_S P$ and $\mathtt{a} \rightarrow \mathtt{b} \in S$. Then, since T and P are 3-interval free, $occ(\mathtt{a}, T) \in \langle T\rangle$ iff $occ(\mathtt{b}, P) \in \langle P\rangle$, $occ(\mathtt{a}, T) \in [\langle T\rangle_i]$ iff $occ(\mathtt{b}, P) \in [\langle P\rangle_{i'}]$, and $occ(\mathtt{a}, T) \in [\langle T\rangle_i]_j$ iff $occ(\mathtt{b}, P) \in [\langle P\rangle_{i'}]_{j'}$. A correct matching for $occ(\mathtt{a}, T)$ and $occ(\mathtt{b}, P)$ is computed as follows. Let $n = |\langle P\rangle|$.

(1) For $\langle T\rangle$, $\langle P\rangle$, and $i = 1, \ldots, n$, check whether there exists a k such that;

(1-a) The length of $\langle T\rangle_{k+i}$ is equal to that of $\langle P\rangle_i$.

(1-b) The number of $\mathtt{a_{k+i+1}}$ in j-th interval of $\mathtt{a_{k+i}}$ is equal to that of $\mathtt{b_{i+1}}$ in j-th interval of $\mathtt{b_i}$, where $occ(\mathtt{a_{k+i+1}}, T), occ(\mathtt{a_{k+i}}, T) \in \langle T\rangle$, $occ(\mathtt{b_{i+1}}, P), occ(\mathtt{a_i}, P) \in \langle P\rangle$.

For all $i = 1, \ldots, n$, $occ(\mathtt{a_{k+i}}, T) \in \langle T\rangle$, and $occ(\mathtt{b_i}, P) \in \langle P\rangle$, the number of $\mathtt{a_{k+i+1}}$ in j-th interval of $\mathtt{a_i}$ is equal to the number of $\mathtt{b_{i+1}}$ in j-th interval of $\mathtt{b_i}$, where $1 \leq j \leq n-1$. This check is done in $O(n(|T| + |P|)) = O(|T|^2)$.

If there is no such a k, then answer "no" and terminate. If a k is found, then make the matching $S = \{\mathtt{a_{k+i}} \rightarrow \mathtt{b_i} \mid i = 1, \ldots, n\}$ and go to the next stage.

(3) If $\mathtt{a_{k+i}} \rightarrow \mathtt{b_i} \in S$, then for each $occ(\mathtt{b}, P) \in [\langle P\rangle_i]$, find an $occ(\mathtt{a}, T) \in [\langle T\rangle_{k+i}]$ such that the number of $\mathtt{b}$ in the j-th interval of $\mathtt{b_i}$ is equal to the number of $\mathtt{a}$ in the j-th interval of $\mathtt{a_{k+i}}$. This check is done in $(|T| + |P|)$. If there is no $occ(\mathtt{a}, T)$ for an $occ(\mathtt{b}, P)$, then let $S = \emptyset$ and go to the condition (1) to find the next k, and add all the $\mathtt{a} \rightarrow \mathtt{b}$ to S otherwise.

(4) Let $[\langle P\rangle_i]_j = (occ(\mathtt{b_1}, P), \ldots, occ(\mathtt{b_m}, P))$ and $(occ(\mathtt{a_1}, T), \ldots, occ(\mathtt{a_m}, T))$ be a subsequence of $[\langle T\rangle_{k+i}]_j$. If $|occ(\mathtt{a_1}, T)| = |occ(\mathtt{b_1}, P)|, \ldots, |occ(\mathtt{a_m}, T)| = |occ(\mathtt{b_m}, P)|$, then add all the $\mathtt{a_1} \rightarrow \mathtt{b_1}, \ldots, \mathtt{a_m} \rightarrow \mathtt{b_m}$ to S. This check is done in $O(|T| + |P|)$.

check whether $T \to P$ is $O(|T|^2 + |P|^2)$. □

Theorem 3.2 $EIP(T,P)^3_1 \in \mathrm{P}$.

Proof: Each string w is represented by a concatenation of $w_1, \ldots, w_n$ such that $w = \alpha w_1 \beta \cdots w_n \gamma$, where all w_i have no split, and for every symbol $\mathtt{a}$ in $\alpha, \beta, \ldots, \gamma$, $\sharp(\mathtt{a}, w) = 1$.

Let $T = \alpha t_1 \beta t_2 \cdots t_n \gamma$ and $P = \alpha' p_1 \beta' p_2 \cdots p_m \gamma'$. All symbols in $\alpha\beta\cdots\gamma$ appear in T exactly once and all symbols in $\alpha'\beta'\cdots\gamma'$ appear in P exactly once. Thus, we have that $T \to_S P$ iff $t_{j_i} \to_{S_i} p_i$, where $j_1, \ldots, j_m$ is a subsequence of $1, \ldots, n$, and $|\alpha| \geq |\alpha'|, \ldots, |\gamma| \geq |\gamma'|$. □

Theorem 3.3 The $EHP(T,P)$ is NP-complete even if P is labeled by a single alphabet.

Proof: The general problem is clearly in NP, then we prove only the hardness, that is 3-SAT is log-space reducible to this problem. 3-SAT is the problem to decide whether there exists a truth assignment for a given 3-CNF over the set $X = \{x_1, \ldots, x_n\}$ of variables. A 3-CNF is a Boolean formula of the form $C = \bigwedge_{i=1}^{m} C_i$ and $C_i = (\tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3})$, where $\tilde{x}$ is a literal of a variable x. The reduction is as follows.

In this proof we use a special notation $t(t_1, \ldots, t_k)$ for a graph which denotes that the edges $(t,t_1)\ldots,(t,t_k)$ are defined and the order $t_1 < \cdots < t_k$ is fixed.

(1) P is the tree defined by the graph (V_p, E_p) such that $E_p = E_{p_0} \cup E_{p_1} \cup E_{p_2}$ for

$$E_{p_0} = \{r(t_0, \ldots, t_m)\}, \quad E_{p_1} = \{t_0(s_1),\ s_1(s_2),\ s_2(a_1, \ldots, a_n)\}, \text{ and}$$
$$E_{p_2} = \{t_i(p_{(i,1)}, \ldots, p_{(i,5)}) \mid i = 1, \ldots, m\}, \text{ where}$$

V is the set of all v and v' for $(v,v') \in E_p$ and the root is r. For any node $v \in V_P$, let $\ell(v) = \mathtt{c}$.

(2) T is the tree defined by the graph (V_t, E_t) such that $E_t = E_{t_0} \cup E_{t_1} \cup E_{t_2}$ for

$$E_{t_0} = \{r(t_0, \ldots, t_m)\}, \quad E_{t_1} = \{t_0(s_1),\ s_1(s_2),\ s_2(a_1(b_1), \ldots, a_n(b_n))\}, \text{ and}$$
$$E_{t_2} = \{t_i(t_{(i,0)}, \ldots, t_{(i,4)}),\ t_{(i,j)}(t_{(i,j,1)}, t_{(i,j,2)}) \mid i = 1, \ldots, m, j = 1,2,3\}, \text{ where}$$

V_t is the set of all v and v' for $(v,v') \in E_t$ and the root is r. For each $i = 1, \ldots, n$, let $\ell(a_i) = \mathtt{A_i}$ and $\ell(b_i) = \mathtt{B_i}$. For each $i = 1, \ldots, n$, and $j = 0, 4$, let $\ell(t_{(i,j)}) = \mathtt{c_{(i,j)}}$.

For each $i = 1, \ldots, n$ and $j = 1,2,3$, let $\ell(t_{(i,j)}) = \mathtt{B_k}$, $\ell(t_{(i,j,1)}) = \ell(t_{(i,j,2)}) = \mathtt{A_k}$ if the j-th literal of C_i is positive, and let $\ell(t_{(i,j)}) = \mathtt{A_k}$, $\ell(t_{(i,j,1)}) = \ell(t_{(i,j,2)}) = \mathtt{B_k}$ if the j-th literal of C_i is negative. Any other node is labeled by $\mathtt{c}$.

node is 2. Thus, if $T \to P$, then $E_{t_1} \to E_{p_1}$ and $E_{t_2} \to E_{p_2}$.

Moreover, if $E_{t_1} \to_S E_{p_1}$, then either $\mathtt{A_i} \to \mathtt{c}, \mathtt{B_i} \to \lambda \in S$ or $\mathtt{B_i} \to \mathtt{c}, \mathtt{A_i} \to \lambda \in S$ for all $i = 1, \ldots, n$. We can consider the corresponding $(\mathtt{A_i} \to \mathtt{c}, \mathtt{B_i} \to \lambda \in S) \Leftrightarrow x_i = 1$ and $(\mathtt{B_i} \to \mathtt{c}, \mathtt{A_i} \to \lambda \in S) \Leftrightarrow \neg x_i = 1$. We denote the operation corresponding to a truth assignment f by S_f. Note that only the matching S_f decide whether $E_{t_2} \to E_{p_2}$ because all labels in E_{t_2} also appear in E_{t_1}.

Suppose that the CNF $C = \bigwedge_{i=1}^{m} C_i$ is satisfiable by f. Then, for each clause C_i of C, at least one literal $\tilde{x_{i_j}}$ in C_i satisfies that $\tilde{x_{i_j}}$ is positive and assigned 1, or $\tilde{x_{i_j}}$ is negative and assigned 0. These are corresponding $(\mathtt{A_i} \to \mathtt{c}, \mathtt{B_i} \to \lambda)$, or $(\mathtt{B_i} \to \mathtt{c}, \mathtt{A_i} \to \lambda)$, respectively. This operation deletes the children of $t_(i, j)$ for at least one of $j = 1, 2, 3$. It follows that T/t_i has 5, 6 or 7 children. In these cases, $T/t_i \to P/t_i$ by one of $(\mathtt{c_{(i,0)}} \to \mathtt{c}, \mathtt{c_{(i,4)}} \to \mathtt{c})$, $(\mathtt{c_{(i,0)}} \to \mathtt{c}, \mathtt{c_{(i,4)}} \to \lambda)$, and $(\mathtt{c_{(i,0)}} \to \lambda, \mathtt{c_{(i,4)}} \to \lambda)$. Thus, we have $T \to_{S_f} P$.

Conversely, if the CNF C is unsatisfiable, then for any assignment f, at least one C_i is not satisfiable, that is $\tilde{x_{i_j}}$ is positive and assigned 0, or $\tilde{x_{i_j}}$ is negative and assigned 1. It follows that all leaves of T/t_i are not deleted by S_f. This implies that T/t_i has 8 children. Thus, $T/t_i \not\to P/t_i$. Hence C is satisfiable iff $T \to P$. The proof is completed. □

Theorem 3.4 The $EHP(T, P)$ is NP-complete even if T is a string.

Proof: This problem is also reducible from 3-SAT defined in the above. A 3-CNF is of the form $C = \bigwedge_{i=1}^{m} C_i$ and $C_i = (\tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3})$. The reduction is as follows.

(1) $T = T_1 \cdot T_2$, $T_1 = \mathtt{A^2 x_1 \neg x_1 \cdots A^2 x_n \neg x_n A^2}$, $T_2 = \alpha_1 \cdots \alpha_m$,

(2) $P = P_1 \cdot P_2$, $P_1 = \mathtt{A^2 x_1 \cdots A^2 x_n A^2}$, $P_2 = \beta_1 \cdots \beta_m$ such that

$\alpha_i = \mathtt{\tilde{x}_{i_1} u_i \tilde{x}_{i_2} v_i \tilde{x}_{i_3} A^2}$ and $\beta_i = \mathtt{x_{i_1} x_{i_2} x_{i_3} A^2}$ iff $C_i = (\tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3})$, where $\sharp(\mathtt{u_i}, T) = \sharp(\mathtt{v_i}, T) = 1$ for all $i = 1, \ldots, m$.

Let $T \to P$. Then $\mathtt{A} \to \mathtt{A}$ or $\mathtt{A} \to \lambda$. Assume the latter. Then, $|T'| = 2n + 5m$ and $|P| = 3n + 5m + 2$, where $T \to_s T'$ for $s = \{\mathtt{A} \to \lambda\}$. Thus, $T \not\to P$ in this case. Since $\mathtt{A} \to \mathtt{A}$ is fixed, if $T \to P$, then at least $T_1 \to P_1$ and $T_2 \to P_2$. Moreover, for each $i = 1, \ldots, n$, exactly one of $\mathtt{x_i}$ and $\mathtt{\neg x_i}$ of T_1 must be mapped to $\mathtt{x_i}$ of P_1.

There exists a one-to-one mapping: $f \to S_f$ such that the variable x_i is assigned 1 by f if $\mathtt{x_i} \to \mathtt{x_i}, \mathtt{\neg x_i} \to \lambda \in S_f$, or x_i is assigned 0 by f if $\mathtt{x_i} \to \lambda, \mathtt{\neg x_i} \to \mathtt{x_i} \in S_f$.

Suppose that the CNF C is satisfiable for an assignment f. Then for each clause C_i, at least one of $\tilde{x}_{i_1}$, $\tilde{x}_{i_2}$ and $\tilde{x}_{i_3}$ is assigned 1. It follows that at least one of $\mathtt{\tilde{x}_{i_1}}$, $\mathtt{\tilde{x}_{i_2}}$ and $\mathtt{\tilde{x}_{i_3}}$ of α_i is not deleted by S_f. Since $\mathtt{u_i} \to \mathtt{a}$ and $\mathtt{v_i} \to \mathtt{b}$ can be defined for any symbols $\mathtt{a}$ and $\mathtt{b}$, it holds that $\alpha_i \to_{S_f \cup \{\mathtt{u_i} \to \mathtt{a}, \mathtt{v_i} \to \mathtt{b}\}} \beta_i$. Thus, $T \rightsquigarrow P$. Conversely, suppose that C is unsatisfiable by any f. Then for at least one α_i, $|\alpha_i'| = 2$ for $\alpha_i \to_{S_f} \alpha_i'$. If follows that $T \not\to P$. Hence, it is proved that C is satisfiable iff $T \to P$. □

In this section, we show that there exists a polynomial time algorithm that exactly identifies any translation system in $LTT(k)$ using equivalence and membership queries.

4.1 The learning problem

Our problem is identifying an unknown tree translation system H_* from examples of ordered pairs $E \in M(H_*)$ that are either derived or not derived by H_*. As a formal model, we employ a variant of exact learning model by Angluin [1] called learning from entailment[2, 3, 7, 8], which is tailored for translation systems.

Let $\mathcal{H}$ be a class of translation systems to be learned, called *hypothesis space*, and LR be the set of all ordered pairs, called the *domain of learning*. In our learning framework, the *meaning* or *the concept* represented by $H \in \mathcal{H}$ is the set $M(H_*)$. If $M(P) = M(Q)$ then we define $P \equiv Q$ and say that P and Q are *equivalent*.

A *learning algorithm* $\mathcal{A}$ is an algorithm that can collect the information about H_* using the following type of queries. In this paper, we assume that the alphabet Σ is given to $\mathcal{A}$ in advance and the maximum arity of symbols in Σ is constant.

Definition 4.1 An *equivalence query* (EQ) is to propose any translation system $H \in \mathcal{H}$. If $H \equiv H_*$ then the answer to the query is "yes". Otherwise the answer is "no", and $\mathcal{A}$ receives any translation $C \in LR$ as a *counterexample* such that either $C \in M(H_*) \backslash M(H)$, or $C \in M(H) \backslash M(H_*)$. A counterexample is *positive* if $C \in M(H_*)$ and *negative* if $C \notin M(H_*)$. A *membership query* (MQ) is to propose any translation $C \in LR$. The answer to the membership query is "yes" if $C \in M(H_*)$, and "no" otherwise.

Definition 4.2 The goal of $\mathcal{A}$ is *exact identification* in polynomial time. $\mathcal{A}$ must halt and output a rewriting system $H \in \mathcal{H}$ such that $H_* \equiv H$, where at any stage in learning, the running time and thus the number of queries must be bounded by a polynomial $poly(m, n)$ in the size m of H_* and the size n of the longest counterexample returned by equivalence queries so far.

Although this setting first seems to be unnatural, it is known that any exact learnability with equivalence queries implies *polynomial time PAC-learnability* [10] and *polynomial time online learnability* [1] under a mild condition on the class of target hypothesis whether additional membership queries are allowed or not [1].

4.2 The learning algorithm

Figure 2 gives our learning algorithm $Learn\_LTT(k)$ that uses equivalence and membership queries to identify a k-variable linear tree translation system H that is equivalent to the target H_*, but may be polynomially larger than H_*.

Input: positive integer k;
Given: the equivalence and the membership queries for the target set $H_* \in LTT(k)$;
Output: a set H of linear tree translations equivalent to H_*;
begin
  $H := \emptyset$;
  until $EQ(H)$ returns "yes" **do**
  begin
    Let E be a counterexample returned by the equivalence query;
    $D := Shrink(E, H)$; /* (See Definition 4.3) */
    $H := H \cup Expand(D, k)$; /* (See Definition 4.4) */
  end /* main loop */
  return H;

Figure 2: A learning algorithm for k-variable linear tree translations using equivalence and membership queries

In the algorithm, we denote by $Shrink(E, H)$ and $Expand(D, k)$ the procedures to return any smallest positive sub-counter examples of rule E and to return the set of k-variable linear rules, called $LTT(k)$-expansion of D.

Definition 4.3 A rule D is called a *smallest positive sub-counterexample* if D is in $M(H_*)\backslash M(H)$ and and there exists no subrule D' of D strictly smaller than D such that $D' \in M(H_*)\backslash M(H)$. If D is a subrule of some rule E then we call D a smallest positive sub-counterexample of E.

Definition 4.4 For a positive counterexample D of H_* w.r.t H and $k \geq 0$, we define the $LTT(k)$-expansion of D by $Expand(D, k) = \{\, C \in LR(k) \mid C \succeq_{H_*} D,\ C \in M(H_*) \,\}$.

The following lemmas state that these procedures *Shrink* and *Expand* work in polynomial time.

Lemma 4.1 *Given a positive counterexample E of H_* wrt. H, a smallest positive sub-counterexample of E can be computed in polynomial time in m and n using $O(n^2)$ membership queries, where $m = card(H)$ and $n = size(E)$.*

Proof: For all pairs $\pi \in occ(E)$, we can check if the subrule $D = E/\pi$ satisfies $D \in M(H_*)$ using a membership query and if $D \in M(H)$ in $O(mn^5)$ time by Lemma 2.1. □

Lemma 4.2 *Let D be a positive counterexample of H_* wrt. H. Then, (i) the cardinality of the set $Expand(D, k)$ is bounded by $O(n^{2k})$. Furthermore, (ii) $Expand(D, k)$ can be computed in polynomial time using $O(kn^{2k})$ membership queries.*

$H_0, H_1, \ldots, H_n, \ldots$ and $E_0, E_1, \ldots, E_n, \ldots$ $(n \geq 0)$, respectively, be the sequence of hypotheses asked in the equivalence queries by the algorithm and the sequence of counterexamples returned by the queries. H_0 is the initial hypothesis $\emptyset$, and at stage $n \geq 1$, $LEARN\_LTT$ makes an equivalence query $EQ(H_{n-1})$, receives a counterexample E_n, and produce the next hypothesis H_n from E_n and H_{n-1}. For unknown rule $C \in H_*$, if $C \in M(H)$ then we say that C is *missing wrt.* H, and otherwise we say that C is *covered by* H.

Lemma 4.3 *Let D be any smallest positive counterexample of H_* wrt. H. Then, there exists some missing rule $C \in H_*$ wrt. H such that $C \succeq_{H_*} D$.*

Proof: Since $D \in M(H_*)$, we know that there exist some rule $C = (l, r) \in H_*$ and some $(p_i, q_i) \in M(H_*)$ for every i such that $D = (l[p_1, \ldots, p_k], r[q_1, \ldots, q_k])$ and thus $C \succeq_{H_*} D$. Note that each $(p_i, q_i) \in M(H_*)$ is a subrule strictly smaller than D. Since D is smallest positive sub-counterexample, we can see that $(p_i, q_i) \in M(H)$ for all i. On the other hand, If $C \in M(H)$ then the contradiction $D \in M(H)$ is derived by the definition of the set $M(\cdot)$. Hence the result follows. □

Lemma 4.4 *If $C \succeq_{H_*} D$ and $C \in M(H_*)$ for some $C \in LR(k)$ then $C \in Expand(D, k)$.*

Lemma 4.5 *For every $n \geq 0$, $M(H_n) \subseteq M(H_*)$, and furthermore, the counterexample E_n is positive.*

Proof: By construction, it is easy to see that $Expand(D, k) \subseteq M(H_*)$ and thus every rule added to H is a member of $M(H*)$. By induction on the construction of $M(H)$, we can show that $H \subseteq M(H_*)$ implies $M(H) \subseteq M(H_*)$. This proves the lemma. □

Lemma 4.6 *For every $n \geq 0$, let $c_n \geq 0$ be the number of missing rules in H_* wrt. H. Then, $c_0 = m > c_1 > \cdots > c_n > \cdots$, where $m = card(H_*)$.*

Proof: First suppose that $c_n = 0$. Then, for every $n \geq 0$, H_n covers all rules in H_*, and thus $M(H_n) \supseteq M(H_*)$. Since $M(H_n) \subseteq M(H_*)$ from Lemma 4.5, this implies that $H_n \equiv H_*$ and the algorithm terminates. On the other hand, suppose that $c_n > 0$. Since there is some missing clause C in H_*, we know $H_n \not\equiv H_*$ and thus a positive counterexample E is given to the algorithm. Then by Lemma 4.1 a smallest positive counterexample D is obtained. By Lemma 4.3 and Lemma 4.4, $Expand(D, k)$ contains some missing rule $C \in H_*$. Thus, $c_{n-1} > c_n$ holds. □

Theorem 4.1 *The algorithm $LEARN\_LTT$ of Figure 2 exactly identifies any translation system H_* in $LTT(k)$ using $O(m)$ equivalence queries and $O(kn^{2k})$ membership queries.*

Proof: By construction of the algorithm, if it terminates then $H = H_n \equiv H_*$. Hence, the correctness of the algorithm immediately follows from Lemma 4.6. The time complexity and the query complexity follows from Lemma 4.1 and Lemma 4.2 (ii). Hence the theorem is proved. □

In this paper, we consider the extraction and reconstruction problems in semi-structured data. We first show that nontrivial subproblem of the erasing isomorphism is in P and the NP-completeness of the erasing homomorphism problem. It is an open question whether there is a gap between the isomorphism and homomorphism problem. Next, we show that if we allow a learner additional information obtained by active queries then the class of k-variable linear translation systems is polynomial time identifiable using equivalence and membership queries wrt. the translation relation.

References

[1] D. Angluin, "Queries and concept learning," Machine Learning, vol.2, pp.319–342, 1988.

[2] H. Arimura, "Learning Acyclic First-order Horn Sentences From Entailment," Proc. 7th Int. Workshop on Algorithmic Learning Theory (LNAI 1316), pp.432–445, 1997.

[3] H. Arimura, H. Ishizaka, and T. Shinohara, "Learning unions of tree patterns using queries," Theoretical Computer Science, vol.185, pp.47–62, 1997.

[4] J. Clark (ed.), "XSL Transformations (XSLT) Version 1.0," W3C Recommendation 19 November 1999, `http://www.w3.org/TR/xslt`.

[5] N. Dershowitz, J.-P. Jouannaud, Rewrite Systems, Chapter 6, Formal Models and Semantics, Handbook of Theoretical Computer Science, Vol. B, Elseveir, 1990.

[6] F. Drewes, Computation by Tree Transductions, Ph D. Thesis, University of Bremen, Department of Mathematics and Informatics, February 1996.

[7] M. Frazier and L. Pitt, Learning from entailment: an application to propositional Horn sentences, Proc. 10th Int. Conf. Machine Learning, pp.120–127, 1993.

[8] R. Khardon, "Learning function-free Horn expressions," Proc. COLT'98, pp.154–165, 1998.

[9] P. Kilpelainen and H. Mannila, "Ordered and unordered tree inclusion," SIAM J. Comput., pp.340–356, 1995.

[10] L. G. Valiant, "A theory of learnable," Commun. ACM, vol.27, pp.1134–1142, 1984.