LEARNABILITY OF XML DOCUMENT TRANSFORMATION RULES USING TYPED EFSS

Sugimoto, Noriko
Department of Informatics, Kyushu University

https://doi.org/10.5109/21044

出版情報：Bulletin of informatics and cybernetics. 41, pp.25-38, 2009-12. 統計科学研究会
バージョン：
権利関係：

LEARNABILITY OF XML DOCUMENT TRANSFORMATION RULES USING TYPED EFSS

by

Noriko Sugimoto

Reprinted from the Bulletin of Informatics and Cybernetics
Research Association of Statistical Sciences, Vol.41

FUKUOKA, JAPAN
2009

Bulletin of Informatics and Cybernetics, Vol. 41, 2009

LEARNABILITY OF XML DOCUMENT TRANSFORMATION RULES USING TYPED EFSS

By

Noriko Sugimoto*

Abstract

EFSs are logic programs expressing various formal languages. Typed EFSs are EFSs augmented by introducing types for variables. In this paper, we define a subclass of the typed EFSs, called DS-typed regular EFSs, to model XML documents. We show that the class of languages defined by DS-typed regular EFSs properly includes the class of languages defined by balanced grammars, one formal model of schemata of XML documents. We also define another subclass of the typed EFSs, called DS-typed regular TEFSs, which represent translation rules between languages defined by DS-typed regular EFSs. The DS-typed regular TEFSs can have local variables under some conditions. We prove that the class of translation rules defined by DS-typed regular TEFSs is learnable from positive examples under the restriction that the number of clauses in an EFS and the length of each clause are bounded by some constants. This restriction is essential since the class of translations is not learnable from positive examples without the restriction.

Key Words and Phrases: Typed EFSs, Elementary formal systems, XML document transform, Learnability.

1. Introduction

XML (eXtensible Markup Language) is a widely-known data format for structured documents. In this paper, we propose a unifying framework to discuss the learnability of transformations between XML documents from a viewpoint of the formal language theory.

The *elementary formal systems* (*EFSs*) were first introduced by Smullyan (1961), and can be regarded as logic programs over strings as shown by Arikawa et al. (1992) and Yamamoto (1992). The EFSs are flexible enough to define various classes of formal languages in the Chomsky hierarchy, and work as an adequate tool for learning not only formal languages (Arikawa et al. (1992) and Shinohara (1994)), but also translations between formal languages (Sugimoto et al. (1996) and Sugimoto (1998)). Kiwata & Arikawa (1996) introduced the typed EFSs, which enable us to easily describe some structures of data. The derivation procedure, defined as in the same way as in the ordinary EFSs, was shown to be sound and complete as an acceptor for the languages defined by the typed variable-bounded EFSs.

* Department of Informatics, Kyushu University, Moto-oka 744, Nishi-Ku Fukuoka 819-0395, Japan. tel +81–92–802–3786 noriko.horibe@i.kyushu-u.ac.jp

On the other hand, Berstel & Boasson (2002) discussed XML documents in the framework of extended context-free grammars. They modeled XML documents as Dyck strings and schemata as balanced grammars, where Dyck strings are well-formed sequences of brackets.

In this paper, we adopt the typed EFSs to represent transformation rules over XML documents. First, we introduce a subclass of the typed EFSs to model XML documents, named *DS-typed regular EFSs* (*DS-REFSs*), which consists of the regular EFSs such that all variables are typed with either the nonempty Dyck strings or the Dyck primes. The class of languages represented by DS-REFSs properly includes the class of languages defined by the balanced grammars. It follows from the results by Kiwata & Arikawa (1996) that the derivation procedure is sound and complete as an acceptor of languages defined by DS-REFSs.

Next, we define another subclass of the typed EFSs, called *DS-typed regular TEFSs* (*DS-RTEFSs*), which represent translations over languages defined by DS-REFSs. For a DS-RTEFS, each of the clauses in it may have local variables under some conditions, which will be useful to express translations.

We also discuss the learnability of translations from only positive examples. The learning in this setting was applied to various targets in the works by Gold (1967), Shinohara (1991), Shinohara (1994) and Sugimoto (1998). Arimura & Shinohara (1994) showed that the class of linearly covering programs, which is a useful subclass of logic programs with local variables, is learnable from positive examples. Rao (1996) extended this class by using the linear predicate inequalities to express standard Prolog programs as quick-sort or merge-sort. Sugimoto (1998) proposed the class of linearly-moded EFSs by introducing local variables and linear predicate inequalities based on mode information, which can express translations of context-sensitive languages. Each linearly-moded EFS represents a translation in which every output sentence is shorter than the corresponding input sentence. On the other hand, translations defined by DS-RTEFSs possibly contain some pairs of input-output sentences such that the output sentence is slightly longer than the input sentence.

In this paper, we show that the class of translations defined by DS-RTEFSs is learnable from positive examples under the restriction that the number of clauses in an EFS and the length of each clause are bounded by some constants. Furthermore, we show that this restriction is essential since the class of translations is not learnable from positive examples without the restriction.

2. Preliminaries

This section gives some basic definitions and notations according to the works of Arikawa et al. (1992), Sugimoto & Ishizaka (1999), and Yamamoto (1992).

2.1. Elementary formal systems

The set of finite strings over a set A is denoted by A^*. The empty string is denoted by ε. Let $A^+ = A^* - \{\varepsilon\}$.

Let Σ, X, and Π be a finite set of *constant symbols*, a countable set of *variables*, and a finite set of *predicate symbols*, respectively. We assume that Σ, X, and Π are mutually distinct. Each predicate symbol in Π is associated with a non-negative integer called its *arity*.

A *term* is an element of $(\Sigma \cup X)^+$. A term is said to be *regular* if every variable occurs at most once in the term. An *atomic formula* (*atom*) is of the form $p(\pi_1, \pi_2, \ldots, \pi_n)$, where p is a predicate symbol with arity n and each π_i is a term $(i = 1, 2, \ldots, n)$. A *definite clause* (*clause*) is of the form $A \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$, where $A, B_1, \ldots, B_n$ are atoms. The atom A and the sequence $B_1, \ldots, B_n$ are, respectively, called the *head* and the *body* of the clause. A *goal clause* (*goal*) is of the form $\leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ and the goal with $n = 0$ is called the *empty goal*. An *expression* is a term, an atom, a clause, or a goal. An expression E is said to be *ground* if E has no variable. For an expression E and a variable x, $var(E)$ and $oc(x, E)$ denote the set of all variables occurring in E and the number of occurrences of x in E, respectively. An *elementary formal system* (*EFS*) is a finite set of clauses.

A *substitution* θ is a (semi-group) homomorphism on $(\Sigma \cup X)^+$ satisfying the following conditions:

1. $a\theta = a$ for every $a \in \Sigma$, and
2. the set $D(\theta) = \{x \in X \mid x\theta \neq x\}$ is finite.

For a substitution θ, if $D(\theta) = \{x_1, x_2, \ldots, x_n\}$ and $x_i\theta = \pi_i$ for every i $(i = 1, 2, \ldots, n)$, then θ is denoted by the set $\{x_1/\pi_1, x_2/\pi_2, \ldots, x_n/\pi_n\}$. For an expression E and a substitution θ, $E\theta$ is defined to be the expression obtained by simultaneously replacing each occurrence of the variables x in E with $x\theta$.

Let (E_1, E_2) be a pair of expressions. A substitution θ is said to be a *unifier* of E_1 and E_2 if $E_1\theta = E_2\theta$. The set of all unifiers θ of E_1 and E_2 satisfying $D(\theta) \subseteq var(E_1) \cup var(E_2)$ is denoted by $U(E_1, E_2)$. We say that E_1 and E_2 are *unifiable* if the set $U(E_1, E_2)$ is not empty. An expression E_1 is a *variant* of E_2 if there exist two substitutions θ and δ such that $E_1\theta = E_2$ and $E_2\delta = E_1$.

2.2. Semantics of EFSs

We give two semantics of EFSs by using *provability relations* and *derivations*. First, we introduce the provability semantics. Let Γ and C be an EFS and a clause. Then, the provability relation $\Gamma \vdash C$ is defined inductively as follows:

1. If $C \in \Gamma$ then $\Gamma \vdash C$.
2. If $\Gamma \vdash C$ then $\Gamma \vdash C\theta$ for any substitution θ.
3. If $\Gamma \vdash A \leftarrow B_1, \ldots, B_m$ and $\Gamma \vdash B_m \leftarrow$ then $\Gamma \vdash A \leftarrow B_1, \ldots, B_{m-1}$.

A clause C is *provable from* Γ if $\Gamma \vdash C$ holds. The *provability semantics* of an EFS Γ, denoted by $PS(\Gamma)$, is defined as the set of all ground atoms A satisfying $\Gamma \vdash A \leftarrow$. For an EFS Γ and a unary predicate symbol p, the *language defined by* Γ *and* p is denoted by $L(\Gamma, p)$, and defined as the set of all strings $w \in \Sigma^+$ such that $p(w) \in PS(\Gamma)$.

The second semantics is based on a *derivation* for EFSs. We assume a *computation rule* R to select an atom from every goal. Let Γ be an EFS, G be a goal, and R be a computation rule. A *derivation from* G is a (possibly infinite) sequence of triplets (G_i, C_i, θ_i) $(i = 0, 1, \ldots)$ which satisfies the following conditions:

1. G_i is a goal, θ_i is a substitution, C_i is a variant of a clause in Γ, and $G_0 = G$.

2. $var(C_i) \cap var(C_j) = \emptyset$ for any i and j with $i \neq j$, and $var(C_i) \cap var(G_i) = \emptyset$ for any i.

3. If $G_i = \leftarrow A_1, \ldots, A_k$, and A_m is the atom selected by R, then C_i is of the form $A \leftarrow B_1, \ldots, B_n$ satisfying that A and A_m are unifiable, $\theta_i \in U(A, A_m)$, and G_{i+1} is of the following form:

$$(\leftarrow A_1, \ldots, A_{m-1}, B_1, \ldots, B_n, A_{m+1}, \ldots, A_k)\theta_i.$$

The atom A_m is called a *selected atom* of G_i, and G_{i+1} is called a *resolvent* of G_i and C_i by θ_i.

A *refutation* is a finite derivation ending with the empty goal. The *procedural semantics* of an EFS Γ, denoted by $RS(\Gamma)$, is defined as the set of all ground atoms A satisfying that there exists a refutation of Γ from the goal $\leftarrow A$.

Yamamoto (1992) showed that $PS(\Gamma) = RS(\Gamma)$ for every EFS Γ. This implies that a string $w \in \Sigma^+$ is in the language defined by an EFS Γ and a predicate symbol p if and only if there exists a refutation of Γ from $\leftarrow p(w)$. The derivation procedure can thus be regarded as an acceptor for the language.

3. Typed EFSs

This section augments EFSs by introducing types for variables according to the work of Kiwata & Arikawa (1996).

A *type* is a recursive subset of Σ^+. A *typed variable* is an expression $x : T$ consisting of a variable x and a type T, which implies that the type of x is T.

A *context* is a finite set of typed variables. For a context $CN = \{x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n\}$, the set of variables $x_1, x_2, \ldots, x_n$ is denoted by $var(CN)$. A context CN is *consistent* if, for any typed variables $x_i : T_i$ and $x_j : T_j$ in CN, $x_i = x_j$ implies $T_i = T_j$. Let E be an expression and CN be a context such that $var(E) = var(CN)$. Then a couple (E, CN) is said to be a *typed expression*.

EXAMPLE 3.1. Let $E = axaya$, $CN_1 = \{x : \{bb\}, y : \{bbb\}\}$, $CN_2 = \{x : \{(bb)^n \mid n \geq 1\}, y : \{(bbb)^n \mid n \geq 1\}\}$, and $CN_3 = \{x : \{(bb)^n \mid n \geq 1\}\}$. Then, both (E, CN_1) and (E, CN_2) are typed expressions. On the other hand, (E, CN_3) is not a typed expression because $y \in var(E)$ and $y \notin var(CN_3)$.

Let $\theta = \{x_1/\tau_1, \ldots, x_n/\tau_n\}$ be a substitution and CN be a context such that $var(CN) = var(\tau_1) \cup var(\tau_2) \cup \cdots \cup var(\tau_n)$. Then, a couple (θ, CN) is said to be a *typed substitution*.

EXAMPLE 3.2. Let $\theta = \{x'/ax, y/aaya\}$, $CN_1 = \{x : \{bb\}, y : \{bbb\}\}$, $CN_2 = \{x : \{(bb)^n \mid n \geq 1\}, y : \{(bbb)^n \mid n \geq 1\}\}$, and $CN_3 = \{x : \{(bb)^n \mid n \geq 1\}\}$. Then, (θ, CN_1) and (θ, CN_2) is typed substitution. However, (θ, CN_3) is not a typed substitution because $y \in var(ax) \cup var(aaya)$ and $y \notin var(CN_3)$.

Let E be an expression and $CN = \{x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n\}$ be a context. Then, the *interpretation of E on CN*, denoted by $I(E, CN)$, is defined as the set of all expressions E' such that there exists a substitution θ satisfying the following conditions:

1. $E' = E\theta$,

2. $D(\theta) = var(CN)$, and
3. for each $x_i \in var(CN)$, $x_i\theta \in T_i$.

Let (E, CN) be a typed expression. Then, the *interpretation* of (E, CN), denoted by $I(E, CN)$, is defined as the interpretation of E on CN. It is clear that all elements in $I(E, CN)$ are ground.

Example 3.3. Let E, CN_1, CN_2, and CN_3 be an expression and contexts used in Example 3.1. Then, $I(E, CN_1) = \{abbabbba\}$ and $I(E, CN_2) = \{a(bb)^m a(bbb)^n a \mid m, n \geq 1\}$, and $I(E, CN_3) = \{a(bb)^m aya \mid m \geq 1\}$ hold.

Let (E, CN) be a typed expression with $CN = \{x_1 : T_1, x_2 : T_2, \ldots, x_m : T_m\}$ and (θ, D) be a typed substitution, where $\theta = \{y_1/\tau_1, y_2/\tau_2, \ldots, y_n/\tau_n\}$. Then, (θ, D) *is applicable to* (E, CN) if and only if it satisfies the following conditions:

1. For any $i \in \{1, 2, \ldots, m\}$, if there exists $j \in \{1, 2, \ldots, n\}$ such that $y_j = x_i$ then the interpretation of τ_j on D is a subset of T_i.
2. Context $C((E, CN) \cdot (\theta, D)) = \{x_i : T_i \mid x_i \notin \{y_1, y_2, \ldots, y_n\}\} \cup \{z_j : \mathcal{S}_j \in D \mid z_j \in var(E \cdot \theta)\}$ is consistent.

Then the typed expression $(E, CN) \cdot (\theta, D)$ is defined to be $(E \cdot \theta, C((E, CN) \cdot (\theta, D)))$.

In general, it is difficult to determine whether (θ, D) is applicable to (E, CN) or not, since we have to consider the inclusion relation on sets of strings in the first condition of the above statements. However, we can avoid the difficulty by restricting the class of types as in the following sections

Example 3.4. Suppose that E, θ, CN_1, CN_2, and CN_3 are the same as in Example 3.1 and Example 3.2. Let T_1, T_2, T_3, and T_4 be the types $\{bb\}$, $\{bbb\}$, $\{(bb)^n \mid n \geq 1\}$, and $\{(bbb)^n \mid n \geq 1\}$, respectively. Then, $(\theta, CN_2) = (\{x'/ax, y/aaya\}, \{x : T_3, y : T_4\})$ is not applicable to $(E, CN_1) = (axaya, \{x : T_1, y : T_2\})$, since $y \in var(CN_1) \cap D(\theta)$ but the interpretation of $aaya$ on CN_2 is $\{aa(bbb)^n a \mid n \geq 1\}$ and it is not a subset of T_2.

On the other hand, $(\theta, CN_1) = (\{x'/ax, y/aaya\}, \{x : T_1, y : T_2\})$ is applicable to $(E, CN_2) = (axaya, \{x : T_3, y : T_4\})$, and

$$\begin{aligned}(E, CN_2) \cdot (\theta, CN_1) &= (axaya, \{x : T_3, y : T_4\}) \cdot (\{x'/ax, y/aaya\}, \{x : T_1, y : T_2\}) \\ &= (E \cdot \theta, C((E, CN_2) \cdot (\theta, CN_1))) \\ &= (axaaayaa, \{x : T_3, y : T_2\})\end{aligned}$$

Definition 3.5. A typed EFS is a finite set of typed definite clauses.

We can also define the provability, the derivations, the refutations, and the definable languages for the typed EFSs in a similar way to the ordinary EFSs.

Lemma 3.6 (Kiwata & Arikawa (1996)). *Let E_1 and E_2 be a pair of typed expressions. If one of them is ground, then every unifier of E_1 and E_2 is ground and the set of all unifiers $U(E_1, E_2)$ is finite and computable.*

A typed EFS Γ is said to be *variable-bounded*, if each clause $A \leftarrow B_1, B_2, \ldots, B_m$ in Γ satisfies $var(B_i) \subseteq var(A)$ for each i $(i = 1, 2, \ldots, m)$.

Lemma 3.7 (Kiwata & Arikawa (1996)). *The derivation procedure is complete and sound as an acceptor for languages defined by the typed variable-bounded EFSs.*

4. Relationships between typed EFSs and ordinary EFSs

For a typed EFS Γ, let $\mathcal{ORG}(\Gamma)$ denote the set of ordinary EFSs that are equivalent to Γ, and let $\mathcal{T}(\Gamma)$ denote the set of types occurring in Γ.

Lemma 4.1. *Let Γ be a typed variable-bounded EFS, then there exists at least one variable-bounded EFS in $\mathcal{ORG}(\Gamma)$.*

Proof. As shown by Arikawa et al. (1992), the class of languages defined by the variable-bounded EFSs subsumes the class of recursive languages. Let $T_1, T_2, \ldots, T_n$ be the types in $\mathcal{T}(\Gamma)$. We can build variable-bounded EFSs $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ such that, for any i and j in $\{1, 2, \ldots, n\}$, $L(\Gamma_i) = T_i$ and $i \neq j$ implies $\Pi(\Gamma_i) \cap \Pi(\Gamma_j) = \emptyset$. For each clause $(A \leftarrow B_1, \ldots, B_m, CN) \in \Gamma$ and each $x_j : T_{i_j} \in CN$ $(j = 1, \ldots, k)$, a clause $A \leftarrow B_1, B_2, \ldots, B_m, q_0^{i_0}(x_0), q_0^{i_1}(x_1), \ldots, q_0^{i_k}(x_k)$ be in Γ', where $q_0^{i_j}$ is a start predicate symbol of Γ_j. It is clear that $\Gamma' \cup \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_n$ is variable-bounded in $\mathcal{ORG}(\Gamma)$.

A typed EFS Γ is said to be a *typed length-bounded EFS*, if each clause $A \leftarrow B_1, B_2, \ldots, B_m$ in Γ satisfies $|A\theta| \geq |B_1\theta| + |B_2\theta| + \cdots + |B_m\theta|$ for any substitution θ.

Lemma 4.2. *Let Γ be a typed length-bounded EFS satisfying that the types in $\mathcal{T}(\Gamma)$ are definable by length-bounded EFSs. Then there exists at least one length-bounded EFS in $\mathcal{ORG}(\Gamma)$.*

Proof. Let $T_1, T_2, \ldots, T_n$ be the types in $\mathcal{T}(\Gamma)$. From the assumption of this lemma, we can build length-bounded EFSs $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ satisfying that, for any i and j, $L(\Gamma_i, q_0^i) = T_i$ and $i \neq j$ implies $\Pi(\Gamma_i) \cap \Pi(\Gamma_j) = \emptyset$. Let Γ' be the set of clauses C' constructed from Γ by the following procedure: for each clause C in Γ and for each variable x occurring in the head of C, if x does not occur in the body, then C' is constructed by adding the atom $q_0^i(x)$ to the body of C, where the type of x is T_i.

It is clear that $\Gamma' \cup \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_n$ is length-bounded and in $\mathcal{ORG}(\Gamma)$.

A typed EFS is said to be a *typed regular EFS*, if each clause $A \leftarrow B_1, B_2, \ldots, B_m$ in Γ satisfies the following conditions:

1. The predicate symbols are unary.
2. Every variable x occurs at most once in A.
3. For any i $(i = 1, 2, \ldots, m)$, B_i is of the form $p(x_i)$ and $x_i \in var(A)$.
4. For any i and j $(i, j = 1, 2, \ldots, m)$, $i \neq j$ implies $x_i \neq x_j$.
5. The types in $\mathcal{T}(\Gamma)$ are definable by regular EFSs.

Lemma 4.3. *There exists a typed regular EFS Γ such that $\mathcal{ORG}(\Gamma)$ has no regular EFSs.*

Proof. Let

$$\Gamma = \left\{ \begin{array}{l} (q_0(x) \leftarrow q_1(x), \{x : \{a^m b^m c^n \mid n \geq 1\}\}), \\ (q_1(xy) \leftarrow, \{x : \{a^m \mid m \geq 1\}, y : \{b^n c^n \mid n \geq 1\}\}) \end{array} \right\}.$$

Then, types $\{a^m b^m c^n \mid n \geq 1\}$, $\{a^m \mid m \geq 1\}$, and $\{b^n c^n \mid n \geq 1\}$ are defined by the following regular EFSs:

$$\Gamma_1 = \left\{ \begin{array}{l} q_0(xy) \leftarrow q_1(x), q_2(y); \\ q_1(axb) \leftarrow q_1(x); \\ q_1(ab) \leftarrow; \\ q_2(cx) \leftarrow q_2(x); \\ q_2(c) \leftarrow; \end{array} \right\},$$

$$\Gamma_2 = \left\{ \begin{array}{l} q_0(ax) \leftarrow q_0(x); \\ q_0(a) \leftarrow; \end{array} \right\}, \text{ and } \Gamma_3 = \left\{ \begin{array}{l} q_0(bxc) \leftarrow q_0(x); \\ q_0(bc) \leftarrow; \end{array} \right\}.$$

Since Γ_1, Γ_2, and Γ_3 are regular EFSs, Γ is a typed regular EFS. On the other hand, we can prove that $L(\Gamma, q_0) = \{a^n b^n c^n \mid n \geq 1\}$ by induction on the length of strings. The class of languages defined by regular EFSs is equivalent to the context-free languages, as proved by Arikawa et al. (1992). Thus, $\mathcal{ORG}(\Gamma)$ contains no regular EFSs.

The above lemmata show the difference between the expressive powers of the typed EFSs and the ordinary EFSs.

5. DS-typed regular EFSs

This section gives a class of typed EFSs to represent languages consisting of well-formed strings of brackets.

Let I be a finite set of indices, and define B_L and B_R by:

$$B_L = \{[_i \mid i \in I\} \qquad B_R = \{]_i \mid i \in I\}.$$

In what follows, we assume that $\Sigma = B_L \cup B_R$.

Definition 5.1 (Bruggemann-Klein & Wood (2004)). A *Dyck string* is defined inductively as follows.

1. The empty string ϵ is a Dick string.
2. If u is a Dyck string and $a \in I$, then $[_a u]_a$ is a Dyck string.
3. If u and v are Dick strings, then uv is also a Dyck string.

The set of non-empty Dyck strings is denoted by DS^+. A Dyck string is said to be a *Dyck prime* if it is of the form $[_a u]_a$ where u is a Dick string. The set of Dyck primes is denoted by DS^1.

Definition 5.2. A typed regular EFS Γ is said to be a *DS-typed regular EFS* (*DS-REFS*), if the term in the head of each clause is of the form $[_a x_1 x_2 \cdots x_k]_a$ or $x_1 x_2 \cdots x_k$, and all types in $\mathcal{T}(\Gamma)$ are DS^1 or DS^+.

From now on, variables x typed with DS^+ are written as x^+ for the sake of simplicity. Thus, the clause

$$(p(xy) \rightarrow q_1(x), q_2(y), \{x : DS^1, y : DS^+\})$$

can be written as

$$p(xy^+) \rightarrow q_1(x), q_2(y^+).$$

Proposition 5.3. *The set of languages defined by DS-REFSs properly includes the set of languages defined by balanced grammars.*

Proof. Let G be a balanced grammar (Σ, N, P, A_0), where N is a finite set of non-terminals, P is a finite set of productions, and A_0 is the start symbol in N. From the definition of balanced grammars, each element of P is of the form $A \rightarrow [_a\tau]_a$, where τ is a regular expression on N. If τ includes A^* for some $A \in N$, it is replaced by $\epsilon + A^+$. We can construct equivalent clauses as follows:

1. If $\tau = E_1E_2 \cdots E_k$ where $E_i \in \{A_i, A_i^+\}$ for each $i = 1, 2, \ldots k$, then a clause
$$q_A([_aD_1D_2 \cdots D_k]_a) \leftarrow q_1(D_1), q_2(D_2) \cdots q_k(D_k),$$
is constructed, where if $E_i = A_i$ then $D_i = x_i$ and $q_i = q_{A_i}$, else $D_i = x_i^+$ and q_i is defined as follows:
$$q_i(xy^+) \leftarrow q_{A_i}(x), q_i(y^+), \text{ and}$$
$$q_i(x) \leftarrow q_{A_i}(x).$$
2. If $\tau = \pi_1 + \pi_2 + \cdots + \pi_m$ where each π_i has the above form, then we can construct equivalent clauses by the above method for rules $A \rightarrow \pi_i$ for each $i = 1, 2, \ldots, m$. Let Γ be a DS-REFS constructed by the above method. We can prove that, for any $w \in \Sigma^*$ and any $A \in N$, if w is derived from A on G then $w \in L(\Gamma, q_A)$ holds, by induction on the length of w.

Example 5.4. A balanced grammar $(\{[_0,]_0, [_1,]_1\}, \{A_0, A_1\}, \{A_0 \rightarrow [_0A_1^*]_0, A_1 \rightarrow [_1]_1\}, A_0)$ is is represented by the following DS-REFS:

$$\Gamma = \left\{ \begin{array}{l} p_{A_0}([_0xy^+]_0) \leftarrow p_{A_1}(x), p_1(y^+); \\ p_{A_0}([_0x]_0) \leftarrow p_{A_1}(x); \\ p_1(xy^+) \leftarrow p_{A_1}(x), p_1(y^+); \\ p_1(x) \leftarrow p_{A_1}(x); \\ p_{A_1}([_1]_1) \leftarrow \end{array} \right\}.$$

From the result of Kiwata & Arikawa (1996), we can prove that, for any DS-REFS Γ and any $w \in \Sigma^+$, $w \in RS(\Gamma)$ if and only if $w \in L(\Gamma)$.

6. Translations by typed EFSs

This section gives a class of typed EFSs to represent translation rules between Dyck strings.

An EFS is said to be a *translation* EFS (*TEFS*) if all the predicate symbols in it are binary. Let Γ be a TEFS and q_0 be a start predicate symbol in Π_Γ. Then, the *translation defined by* Γ, denoted by *Trans*(Γ), is defined by

$$\mathit{Trans}(\Gamma) = \{(s, t) \in \Sigma^+ \times \Sigma^+ \mid q_0(s, t) \in PS(\Gamma)\}.$$

We define a class of TEFSs in order to represent translations between Dyck strings.

Definition 6.1. A TEFS Γ is said to be a *DS-typed regular TEFS* (*DS-RTEFS*) if it satisfies the following conditions:

1. The types in $\mathcal{T}(\Gamma)$ are either DS^1 or DS^+.
2. For each clause in Γ, the term of the head has one of the forms $[_a x_1 x_2 \cdots x_k]_a$ or $x_1 x_2 \cdots x_k$ such that $i \neq j$ implies $x_i \neq x_j$.
3. The terms in the body of each clause in Γ are variables.
4. Each clause
$$p_0(s_0, t_0) \leftarrow p_1(s_1, t_1), \ldots, p_m(s_m, t_m)$$
in Γ satisfies the following conditions:
   (a) $var(t_0) \subseteq var(s_0) \cup var(t_1) \cup \cdots \cup var(t_m)$,
   (b) $var(s_i) \subseteq var(s_0) \cup var(t_1) \cup \cdots \cup var(t_{i-1})$ for any i $(1 \leq i \leq m)$, and
   (c) $var(t_i) \subseteq var(t_0) \cup var(s_{i+1}) \cup \cdots \cup var(s_m)$ for any i $(1 \leq i \leq m)$.

Since we can determine whether the above condition holds or not for any typed EFSs, the problem of deciding whether Γ is a DS-RTEFS or not is solvable. For a DS-RTEFS, each clause in the EFS may have local variables under some conditions, which is a useful to express translation rules.

EXAMPLE 6.2. Let Γ be a typed EFS defined as follows:

$$\Gamma = \left\{ \begin{array}{l} q_0([_a x_1 x_2^+]_a, [_a y_1 y_2^+]_a) \leftarrow q_1(x_1, y_1), q_2(x_2^+, y_2^+); \\ q_1([_b x_1 x_2 x_3]_b, x_2) \leftarrow; \\ q_2(x_1 x_2^+, y_1 y_2^+) \leftarrow q_1(x_1, y_1), q_2(x_2^+, y_2^+); \\ q_2(x, y) \leftarrow q_1(x, y) \end{array} \right\}.$$

It is clear that each clause in Γ satisfies the statements in Definition 6.1. The DS-RTEFS Γ represents the translation from $[_a[_b u_1 v_1 w_1]_b[_b u_2 v_2 w_2]_b \cdots [_b u_n v_n w_n]_b]_a$ to $[_a v_1 v_2 \cdots v_n]_a$, where $u_i, v_i, w_i \in DS^1$ for each $i = 1, 2, \ldots, n$.

In Sugimoto & Ishizaka (1999), a *restricted derivation* is proposed in order to generate languages by using *maximally general unifiers* instead of unifiers in the ordinary derivation. In this section, we apply the restricted derivation to DS-RTEFSs in order to compute output sentences from a given input sentence on translations defined by DS-RTEFSs.

In the following discussion, we consider only the term π which satisfies that all variables in $var(\pi)$ are typed by DS^1 or DS^+. A typed term (π, CN) is represented by the expression π' obtained by replacing each variable x occurs in π such that $x : DS^+ \in CN$ into x^+. The interpretation of (π, CN) is denoted by $I(\pi')$. For any typed term π, we can prove the following statements:

1. whether $I(\pi) \subseteq DS^1$ or not is computable, and
2. whether $I(\pi) \subseteq DS^1$ or not is computable,

by the induction on the length of w. This implies the following lemma.

LEMMA 6.3. *For any typed term τ and typed substitution θ such that all types used in τ and θ are DS^1 or DS^+, whether θ is applicable to τ or not is computable.*

Proof. Let θ be a typed substitution $\{x_1/\pi_1, x_2/\pi_2, \ldots, x_m/\pi_m\}$. We can prove that θ is applicable to τ if and only if, for each i $(i = 1, 2, \ldots, m)$, if $\tau = ux_iv$ then $I(\pi_i) \subseteq DS^1$ holds, by the induction on the length of π. This result proves this lemma.

The *equivalence* and the *composition* of two typed substitutions are defined same way as those of ordinary substitutions (Sugimoto & Ishizaka (1999)).

Let E_1 and E_2 be a pair of typed expressions. A *maximally general unifier* (*mxgu*, for short) of E_1 and E_2 is a unifier $\theta \in U(E_1, E_2)$ satisfying that, for any $\delta \in U(E_1, E_2)$ such that θ and δ are equivalent on $var(E_1) \cup var(E_2)$, there is no substitution γ such that $\theta = \delta \cdot \gamma$. The set of all mxgu's of E_1 and E_2 is denoted by $MXGU(E_1, E_2)$.

We define the number of mxgu's of two typed terms π and τ as the cardinality of equivalence classes of substitutions on $var(\pi) \cup var(\tau)$. Thus, we say that $MXGU(\pi, \tau)$ *is finite*, if the number of mxgu's is finite without equivalent substitutions on $var(\pi) \cup var(\tau)$. From the definition of maximally general unifiers, the following lemmata hold (Sugimoto & Ishizaka (1999)).

Lemma 6.4. *Let π and τ be terms. If π is ground, then the set $MXGU(\pi, \tau)$ is finite and computable, and $MXGU(\pi, \tau) = U(\pi, \tau)$ holds.*

Lemma 6.5. *Let π be x or x^+ and τ be a typed term such that $x \notin var(\tau)$. Then, $MXGU(\pi, \tau)$ is a singleton set $\{x/\tau\}$ or $\{\pi/\tau\}$, if π and τ are unifiable.*

Note that, if $\pi = x$ and $\tau = [_a y_1]_a [_a y_2]_a$ then π and τ are not unifiable, because $\pi\theta$ is of the form $[_a u]_a$ for any typed substitution θ which is applicable to π.

A *restricted derivation* is defined by replacing unifiers with mxgu's in the definition of the ordinary derivation. A restricted derivation ending with the empty goal is called a *restricted refutation.* We assume a *computation rule R* to select the left-most atom from every goal.

Let Γ be a DS-RTEFS and $G = \leftarrow q(\pi_1, \pi_2)$ be a typed goal. If π_1 is ground and π_2 is y or y^+, then all resolvents form G is of the form

$$\leftarrow q_1(w, \tau_{(1,2)}), q_2(\tau_{(2,1)}, \tau_{(2,2)}), \ldots, q_m(\tau_{(m,1)}, \tau_{(m,2)}),$$

where $w \in DS^+$ and $\tau_{(i,j)} \in DS^+ \cup \{x, x^+ \mid x \text{ is a variable}\}$. Thus, for each step of the restricted derivation, the number of mxgu's is finite and computable. This result implies the following theorems.

Theorem 6.6. *Let $\leftarrow p(w, y^+)$ be a goal and Γ be a DS-RTEFS. Then, the set of all strings $w' \in DS^+$ such that $\Gamma \vdash p(w, w') \leftarrow$ is computable.*

Theorem 6.7. *Let A be a ground atom and Γ be a DS-RTEFS. Then, the problem of deciding whether $\Gamma \vdash A \leftarrow$ or not is solvable.*

Note that, DS-RTEFSs can represent translations including $(s, t) \in \Sigma^+ \times \Sigma^+$ such that $|s| \leq |t|$. Thus, DS-RTEFSs can represent translations which can not be represented by any linearly-moded EFSs (Sugimoto (1998)).

Example 6.8. Let Γ be a DS-RTEFS given in Example 6.2. We describe a refutation from a goal clause $\leftarrow q_0([_a[_b[_1]_1[_2]_2[_3]_3]_b[_b[_4]_4[_5]_5[_6]_6]_b]_a, y)$ in Figure 1.

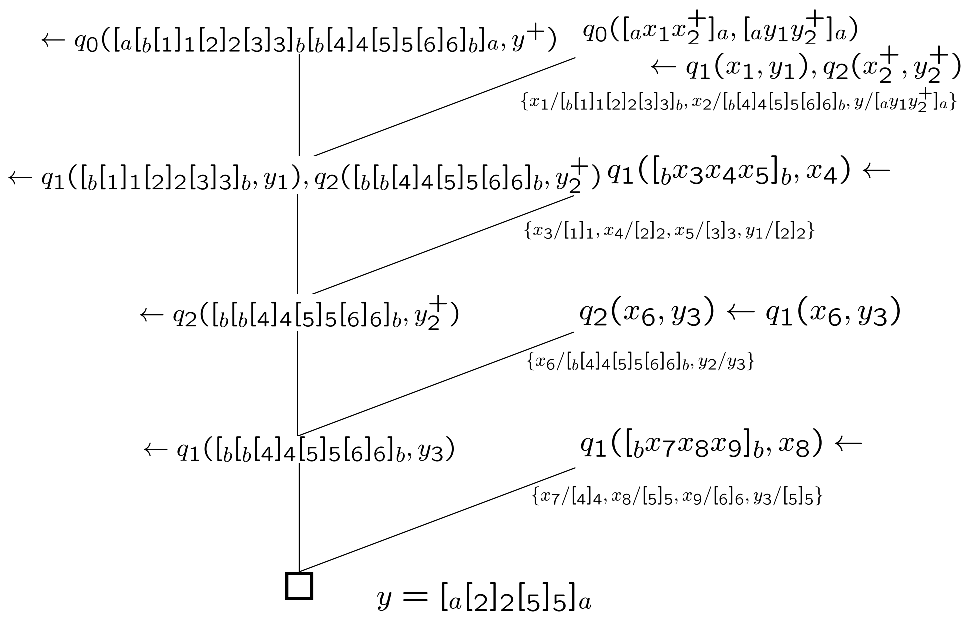
$\leftarrow q_0([_a[_b[_1]_1[_2]_2[_3]_3]_b[_b[_4]_4[_5]_5[_6]_6]_b]_a, y^+)$
$q_0([_a x_1 x_2^+]_a, [_a y_1 y_2^+]_a) \leftarrow q_1(x_1, y_1), q_2(x_2^+, y_2^+)$
$\{x_1/[_b[_1]_1[_2]_2[_3]_3]_b, x_2/[_b[_4]_4[_5]_5[_6]_6]_b, y/[_a y_1 y_2^+]_a\}$
$\leftarrow q_1([_b[_1]_1[_2]_2[_3]_3]_b, y_1), q_2([_b[_b[_4]_4[_5]_5[_6]_6]_b, y_2^+)$
$q_1([_b x_3 x_4 x_5]_b, x_4) \leftarrow$
$\{x_3/[_1]_1, x_4/[_2]_2, x_5/[_3]_3, y_1/[_2]_2\}$
$\leftarrow q_2([_b[_b[_4]_4[_5]_5[_6]_6]_b, y_2^+)$
$q_2(x_6, y_3) \leftarrow q_1(x_6, y_3)$
$\{x_6/[_b[_4]_4[_5]_5[_6]_6]_b, y_2/y_3\}$
$\leftarrow q_1([_b[_b[_4]_4[_5]_5[_6]_6]_b, y_3)$
$q_1([_b x_7 x_8 x_9]_b, x_8) \leftarrow$
$\{x_7/[_4]_4, x_8/[_5]_5, x_9/[_6]_6, y_3/[_5]_5\}$
$y = [_a[_2]_2[_5]_5]_a$

Figure 1: Restricted refutation for the DS-RTEFS

7. Learnability of DS-RTEFSs from positive examples

This section discusses the learnability of subclasses of DS-RTEFSs from positive examples.

Let $\Gamma_1, \Gamma_2, \ldots$ be any recursive enumeration of DS-RTEFSs. Then, the class $C = \mathit{Trans}(\Gamma_1), \mathit{Trans}(\Gamma_2), \ldots$ is an indexed family of recursive sets. A translation is a subset of $\Sigma^+ \times \Sigma^+$. A *semantic mapping* is a mapping from EFSs to translations. A semantic mapping M is *monotonic* if $\Gamma' \subseteq \Gamma$ implies $M(\Gamma') \subseteq M(\Gamma)$. An EFS Γ is *reduced w.r.t.* a set S of atoms if for any $\Gamma' \subset \Gamma$, $S \subseteq M(\Gamma)$ but $S \not\subseteq M(\Gamma')$. A *concept defining framework* is a triple (U, E, M) of a universe U of objects, a universe E of expressions, and a semantic mapping M.

Definition 7.1. A concept defining framework (U, E, M) has *bounded finite thickness* if M is monotonic, and for any finite set $S \subseteq U$ and any n $(n \geq 0)$, the set

$$\{M(\Gamma) \mid \Gamma \text{ is reduced w.r.t. } S \text{ and } |\Gamma| \leq n\}$$

is finite.

Shinohara (1991) showed that if a concept defining framework $C = (U, E, M)$ has bounded finite thickness, then the class

$$C_k = \{M(\Gamma) \mid \Gamma \subseteq E \text{ and } |\Gamma| \leq k\}$$

is learnable from positive examples.

Let E^m denote the set of DS-RTEFSs in which each clause has at most m atoms in its body. Consider the concept defining framework $(\Sigma^+ \times \Sigma^+, E^m, \mathit{Trans})$. Then the next theorem follows.

THEOREM 7.2. *For any $k \geq 0$, the class*

$$TE_k^m = \{ \mathit{Trans}(\Gamma) \mid \Gamma \subseteq E^m \text{ and } |\Gamma| \leq k \}$$

is learnable from positive examples.

PROOF. We show that the concept defining framework $(\Sigma^+ \times \Sigma^+, E^m, \mathit{Trans})$ has bounded finite thickness for any $m \geq 1$.

Since the function PS is monotonic, so is the function *Trans*. Let n be a positive integer, S be a finite subset of $\Sigma^+ \times \Sigma^+$, and l be the maximum length of the amount of s and t such that $(s,t) \in S$. If an DS-RTEFS Γ is reduced w.r.t. S and $|\Gamma| \leq n$, then each clause $p_0(s_0,t_0) \leftarrow p_1(s_1,t_1), \ldots, p_i(s_i,t_i) \in \Gamma$ satisfies the conditions $|s_0| \leq l$, $|t_0| \leq l$, $|s_j| = |t_j| = 1$ for any j $(1 \leq j \leq i)$. Since $i \leq m$ and the number of all predicate symbols in Γ is at most n, the set

$$\{ \mathit{Trans}(\Gamma) \mid \Gamma \text{ is reduced w.r.t. } S \text{ and } |\Gamma| \leq n \}$$

is finite. By Shinohara's theorem (Shinohara (1991)), the class E_k^m is learnable from positive examples.

The following two theorems show that the restriction for the number of clauses and atoms in the body of each clause are essential for the learnability.

THEOREM 7.3. *The class*

$$TE^m = \{ \mathit{Trans}(\Gamma) \mid \Gamma \subseteq E^m \}$$

is not learnable from positive examples for any $m \geq 1$.

PROOF. The class E^1 contains the following EFSs Γ_n $(n \geq 1)$ and Γ_∞:

$$\Gamma_n = \left\{ \begin{array}{l} q_0(x,[y]) \leftarrow q_1(x,y); \\ q_1(x,[y]) \leftarrow q_2(x,y); \\ \cdots \\ q_{n-1}(x,[y]) \leftarrow q_n(x,y); \\ q_0(x,x) \leftarrow; \\ q_1(x,x) \leftarrow; \\ \cdots \\ q_n(x,x) \leftarrow; \end{array} \right\},$$

$$\Gamma_\infty = \left\{ \begin{array}{l} q_0(x,x) \leftarrow; \\ q_0(x,[y]) \leftarrow q_0(x,y) \end{array} \right\}.$$

Then, $\mathit{Trans}(\Gamma_n) = \{([^i]^i,[^j]^j) \mid i \leq j \leq i+n\}$ and $\mathit{Trans}(\Gamma_\infty) = \{([^i]^i,[^j]^j) \mid i \leq j\}$. Since $\mathit{Trans}(\Gamma_i) \subseteq \mathit{Trans}(\Gamma_{i+1})$ and $\mathit{Trans}(\Gamma_i) \subseteq \mathit{Trans}(\Gamma_\infty)$ for any $i \geq 1$, the class E^1 is superfinite. Hence, it is not learnable from positive examples (Gold (1967)).

We denote the set of all DS-RTEFSs which have at most k clauses by E_k.

THEOREM 7.4. *The class*

$$TE_k = \{ \mathit{Trans}(\Gamma) \mid \Gamma \subseteq E_k \}$$

is not learnable from positive examples for any $k \geq 3$.

PROOF. The class E_3 contains the following EFSs Γ_n $(n \geq 1)$ and Γ_∞:

$$\Gamma_n = \left\{ \begin{array}{l} q_0(x,y) \leftarrow q_1(x,y_1), q_1(y_1,y_2), \ldots, q_1(y_n,y); \\ q_1(x,x) \leftarrow; \\ q_1(x,[x]) \leftarrow \end{array} \right\},$$

$$\Gamma_\infty = \left\{ \begin{array}{l} q_0(x,x) \leftarrow; \\ q_0(x,[y]) \leftarrow q_0(x,y) \end{array} \right\}.$$

Then, $\mathit{Trans}(\Gamma_n) = \{([^i]^i, [^j]^j) \mid i \leq j \leq i+n\}$ and $\mathit{Trans}(\Gamma_\infty) = \{([^i]^i, [^j]^j) \mid i \leq j\}$. Since $\mathit{Trans}(\Gamma_i) \subseteq \mathit{Trans}(\Gamma_{i+1})$ and $\mathit{Trans}(\Gamma_i) \subseteq \mathit{Trans}(\Gamma_\infty)$ for any $i \geq 1$, the class E_3 is superfinite. Hence, it is not learnable from positive examples (Gold (1967)).

8. Conclusion

We have proposed a unifying framework to discuss the learnability of translations between XML documents from a viewpoint of the formal language theory. In this paper, XML documents are modeled as well-formed sequences of brackets, and represented by typed EFSs, which have typed variables instead of ordinary variables. We have proposed a subclass of typed EFSs, called DS-REFS, in which all variables are typed with the nonempty Dyck strings or the Dyck primes. We have shown that the class of languages represented by DS-REFSs properly includes the class of languages defined by the balanced grammars. Furthermore, we have defined another subclass of the typed EFSs, called DS-RTEFSs, which represent translations over languages defined by DS-RTEFSs. The DS-RTEFSs is powerful expressions for translations, in which local variables are allowed differently from the ordinary EFSs. In the derivations of the DS-RTEFSs, all unifiers of two terms are computable even if both of the two terms include variables. This is an important and special property of DS-RTEFSs, because it is hard to consider the compatibility of types for each variable in general. Thus, we have obtained that, in the class of translations defined by DS-RTEFS, all output strings are computable from a given input string by the derivation procedure using maximally general unifiers instead of unifiers in the ordinary derivation procedure. Finally, we have shown that the class of translations defined by DS-RTEFSs is learnable from positive examples under the restrictions that the number of clauses in an EFS and the length of a clause are bounded by some constant. Furthermore, we have shown that the restriction is essential since the class of translations is not learnable from positive examples without the restriction.

One of the future works is to develop an efficient learning algorithm for the DS-RTEFSs. In particular, it is important issue to discuss the learnabilitiy of translations by *constructive* methodology in order to implement an automatic transformation system for XML documents.

References

S. Arikawa, T. Shinohara, and A. Yamamoto (1992). Learning elementary formal systems. *Theoretical Computer Science*, 95(11):97–113.

H. Arimura and T. Shinohara (1994). Inductive inference of prolog programs with linear data dependency from positive data. *Proc. Information Modelling and Knowledge Bases V*, 365–375.

J. Berstel and L. Boasson (2000). XML grammars. *Proc. 25th International Symposium on Mathematical Foundations of Computer Science*, pages 182–191.

J. Berstel and L. Boasson (2002). Formal properties of XML grammars and languages. *Acta Informatica*, 38(9):649–671.

E. Gold (1967). Language identification in the limit. *Information and Control*, 10:447–474.

K. Kiwata and S. Arikawa (1996). Introducing types into elementary formal systems. *Bulletin of Informatics and Cybernetics*, 28(1):79–89.

A. Bruggemann-Klein and D. Wood (2004). Balanced context-free grammars, hedge grammars and pushdown caterpillar automata. *Proc. Extreme Markup Language 2004*.

M. K. Rao (1996). A class of prolog programs inferable from positive data. *Proc. 7th International Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence 1160, 272–284.

T. Shinohara (1991). Inductive inference of monotonic formal systems from positive data. *New Generation Computing*, 8:371–384.

T. Shinohara (1994). Rich classes inferable from positive data: length-bounded elementary formal system. *Information and Computation* 108:175–186.

R. Smullyan (1961). Theory of formal systems. Princeton University Press.

N. Sugimoto, K. Hirata and H. Ishizaka (1996). Constructive learning of translations based on dictionaries. *Proc. 7th International Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence 1160, 177–184, 1996.

N. Sugimoto (1998). Learnability of translations from positive examples. *Proc. 9th International Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence, 1501:169–178.

N. Sugimoto and H. Ishizaka (1999). Generating languages by a derivation procedure for elementary formal systems. *Information Processing Letters*, 69:161–166.

N. Sugimoto, H. Ishizaka, and T. Shinohara (2001). An efficient derivation for elementary formal systems based on partial unification. *Proc. 4th International Conference on Discovery Science 2001*, Lecture Notes in Artificial Intelligence, 2226:350–364.

A. Yamamoto (1992). Procedural semantics and negative information of elementary formal system. *Journal of Logic Programming*, 13:89–97.

Received December 3, 2008
Revised December 25, 2008